99

FMcalc — Calculations in FrameMaker

Overview v2.2

FMcalc is an ExtendScript package which provides functions for calculations in FrameMaker.

- The formulas are stored in special markers.
- Results are stored in variables or temporary variables inserted immediately after the **#calc** marker.
- The formulas can contain mathematical constructs of significant complexity.
- With **#series** markers nearly arbitrary numbering becomes possible.
- Both marker types can be created with building blocks to avoid typing errors. Building blocks are maintained in an own panel.

This script would not have been possible without the generous help form the FrameMaker ExtendScript community [2]. In particular I want to thank Klaus Göbel, Jongware [17], Rick Quatro, Russ Ward and Markus Wiedemaier.

Note: *Output created by* **FMcalc** *is highlighted like this.*

Important

- Any operation performed by a script can not be undone[3])! Best practice is to save the document before performing script actions on it.
- In case of an error or change of Your mind you can simply Revert to Saved for the pertinent document.
- Only scripts which do not modify document contents (e.g. creating a new document) are safe in this respect.

3 This must not necessarily be the case for plug-ins represented as dll's.

Contents

Introduction

Why this script (set of scripts)?

Calculations in tables is one of the sometimes listed missing features in FrameMaker. In many cases this function could eliminate the need for an (unpopular) `OLE` link and thus reduce the program clutter in an environment.

Around 1995 `www.riess.de` provided a calculation package (API FrameScript), but this was abandoned prior to FM 6.

FrameScript — With the availability of the powerful macro language FrameScript (by Elmsoft Inc., USA) my project seemed feasible. The math capabilities of this API, however, are limited to the four species (+ - × ÷) and the concept of real[4] numbers is weak.

EScript — With FM-10 Adobe introduced the script language ExtendScript, a derivative of JavaScript [1]. This offers full access to all FM-capabilities and also has mathematical capabilities.

Project history

2002	First ideas and contact with framescript.com concerning math functions.
2004-04	Cheybyshev polynomials evaluated for defining math functions.
2010-01	More implementation considerations. Looking for other calculators.
2013-06	Consider ExtendScript rather than FrameScript.
2015-04	Naming the project **FMcalc**. Intensive revision of the proposal according to new thoughts and ideas.
2016-04	Revision of certain definitions; start programming after experience with `FMbiblio`.
2017-03	v1.0: Release to my website after 900h on the project.
2017-11	v1.1: Handling of none-numeric input from file.
2017-12	v1.2: Additional function Arc (*deg, min, sec*).
2018-03	v1.3: Installation routine and other project related changes.
2018-10	v1.4: Additional function Median.
2019-09	v1.5: Additional functions StdDev, MeanA, MeanH.
2021-09	v1.6: Allow definition of new formats {r} and {R} in Document Settings panel.

See the web-page for further development steps.

Script installation

See *Script installation* on page 44.

4 The term `Real` is used in programming languages to specify numbers with a fractional part. In contrast, integers do not bear a fractional part - they are whole numbers. While integers are always correct, but limited in magnitude, real numbers may define numbers of much greater magnitude, but limited accuracy. See *Numeric range* on page 61.

Functional overview

FMcalc does not change the structure of a FrameMaker document. Documents processed by **FMcalc** are completely compatible with other FrameMaker documents. **FMcalc** uses the existing FrameMaker marker (and variable) system, unaltered. **FMcalc** merely provides an *adapted to the task* interface to the standard FrameMaker marker (and variable) system. Documents processed with **FMcalc** can always be opened by FrameMaker, whether **FMcalc** is installed or not.

Sequence of marker processing

FMcalc markers are handled in the same sequence as with **Find > Marker of Type x**. While tables are entered during the search as they exist in the flow, text frames within anchored frames are searched after the main flow.

General mechanism

FMcalc handles only markers of type **#calc** or **#series**. Any other markers (e.g. Author or Index) are not touched.

#calc Define calculations and store result in a user variable or a temporary variable which will be inserted directly after the **#calc** marker.

#series. Define series. The evaluation of this type will store results in successive text elements using a corresponding character format.

Definition and evaluation of the special markers is initiated by appropriate menu entries or keyboard short cuts.

User variables and program constants can be maintained in an appropriate dialogue (Document settings).

All dialogues of **FMcalc** are palettes which stay open until closed. Thus the current document can be switched.

Document update

Even if you only invoke the dialogues (and don't perform any functions) the document is considered changed (indicated by the highlighted **Save-icon** in the FM-toolbar)[5)].

Comments in #calc and #series markers

Use this feature to identify the markers to make it easier finding them with the standard FM-search: `"page 17"` etc.

Variables

Variables can contain initial data (constants), but normally receive results from calculations.

Although FrameMaker variables can bear any name (except the system variables), **FMcalc** observes a naming scheme:

- Variable names are case sensitive (FrameMaker standard).
- **FMcalc** only recognises variables starting with the # symbol or the `@` symbol. User defined variables use the # symbol. At least one alphabetic character must follow:
  `#côté, #grüne_5äpfel, #Äpfel, #z`

5 FM considers the switch to master page or reference page as a document change - since FM-7.2. This is done hidden from the user when the script reads contents from the reference page.

- Special characters (blank, minus, plus, asterisk etc.) in variable names are not allowed[6]. The only special character allowed is the underscore:
  `#a_valid_name`, `#an-invalid+variable name`
- Use upper case names for user constants, for example `#VAT`.
- Reserved variables use names `@XXX`. These are numeric constants such as `@PI` or physical constants such as `@AVOGADRO`.
- Variables are used in the text where needed: *The relationship between circumference and diameter is* π = *iii*.
- **FMcalc** *markers* are only interpreted on body pages (in all flows, hence also in unnamed text flows in anchored frames).

FMcalc *variables* may be used also on master pages[7].

Examples of variables

User variables	Example contents	Reserved variables	Contents
`#VAT`	0.076	`@PI`	3.141592653589790
`#vat_total`	198.37	`@GRAVITYC`	9.806650000000000
`#anySum`	3712.45	`@RAD2DEG`	$5.729577951308230\times10^{+1}$

Vector

A variable containing more than one value (a list of items) is called a vector. They must be defined in an own statement and use a special assignment (<= rather than only =):

Example

```
#indata <= 17.5, -13, #VAT, 24.777, 11.15, @PI, 23.5e-9;
#average = MeanA (#indata) "xmpl 01";
```

Data from file

The function `GetData` allows to read lengthy lists of values from a text file and assign it to a variable. See *GetData* on page 21.

Note on handling markers

Markers are indicated in the document with the symbol **T** (if you have **View > TextSymbols** activated). It is very difficult to select these text symbols (except that for a non breaking space (␣), because they bear no width.

FMcalc allows you to place the cursor left to the marker you want to handle and **FMcalc** will select the next following **#calc** marker or **#series** marker.

Drawback

If you happen to have selected a **#calc** marker or **#series** marker and then invoke “Handle #calc markers” or “Handle #series markers”, the palette will use the *next* it finds.[8]

Solution

However, you can always move to the desired marker with the **previous** button (◀). You may benefit from the comment feature of **FMcalc**: introduce marker contents with `"ident of this marker"`.

6 FrameMaker accepts variables such as `is␣this+valid?` — **FMcalc** does not. See also *Allowed names for series* on page 53.

7 Items in master pages will appear as constant on the body pages. However, the dynamic aspect of re-calculation will still exist (similar to system variables like `Modification Date`).

8 This is true also for the selection of a marker-group which You can not distinguish visually from a single marker.

Go to Marker buttons

If the palette is already open, the buttons “Go to Marker”, labelled ⏮, ◀, ▶ and ⏭ work on the internally stored sequence of markers.

Example

- The currently selected (and used in the palette) marker be the 4th in the document.
- You place the cursor somewhere between marker 2 and 3
- ▶ will go to the 5th marker, not to the 3rd one (◀ will go to the 3rd, not to the 2nd marker).

Work with open palette

To work on an arbitrary marker while the palette is open:

- Place the cursor somewhere before the marker.
- Use button **Refresh** to adjust the palette contents.
- The cursor will now display the marker following your cursor placement (or selection). The “Go to Marker” buttons now work from this location on.
- If your cursor position is behind the last marker and you **Refresh**, then FMcalc wraps to the first marker.

#calc markers

A new marker type #**calc** is used. The contents of the marker is the formula, for example:

```
= #VAT * Prod(Left) "xmpl 02";
#total = Sum (Above) * (1 + #VAT) "xmpl 03";
#xpos = #length * Cos(#alpha) "xmpl 04";
```

Mulitple formulas in one marker

A **#calc** marker can contain more than one formula (statements). Statements are separated by a semicolon.

```
#a=#VAT*Prod(Left); #total=Sum(Above)*(1+#VAT);
#b=2.5*#a "xmpl 05";
```

Comments in formula

Between any formula elements comments can be placed to make the formula more understandable for human readers. Comments are surrounded by *straight* double quotes:

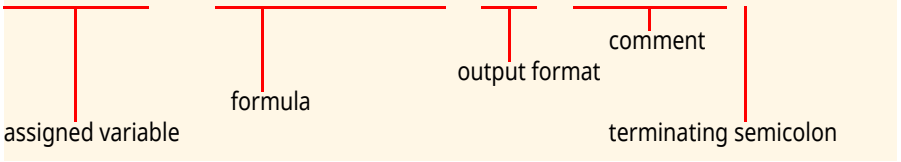
```
#xpos = #length * Cos(#alpha) "horizontal extension";
= #VAT "Value Added Tax" * Prod(Left) "xmpl 06";
```

Syntax of formula (statement)

As seen from the above examples the syntax is similar to that of arbitrary programming languages. A formula is linearised and needs parentheses for clarification.

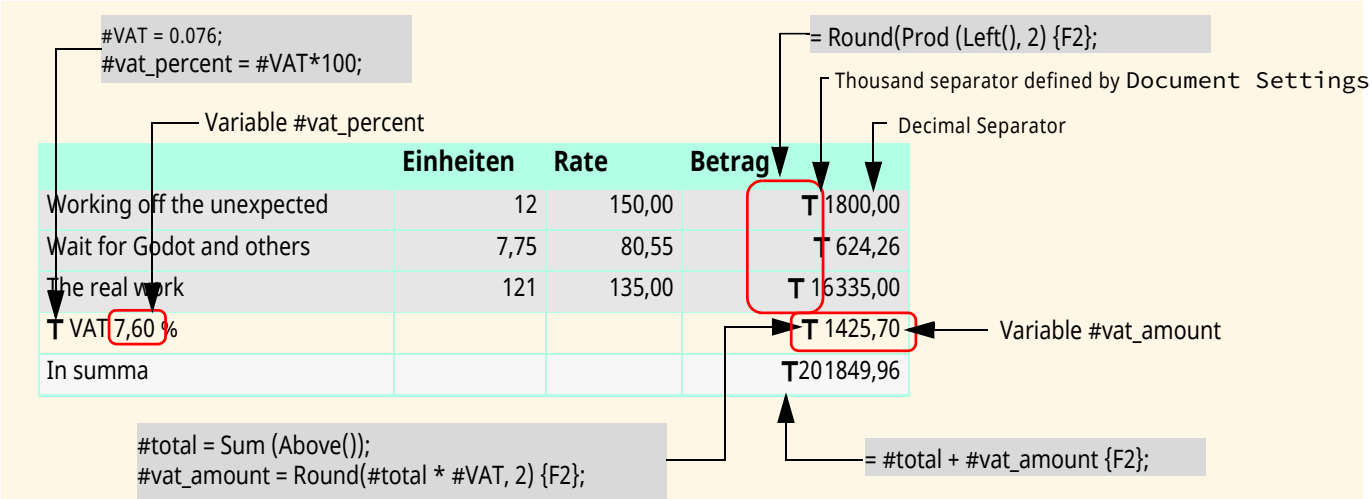
For example, $something = \sin\left(\sqrt{\pi\frac{1}{3}}\right)$ can be written in the marker as:

```
#something = Sin(Sqrt(@PI/3)) {E5} "any hokus";
```



Example table

It is assumed that Document Settings provide comma for `Decimal Separator` and thin space for `Thousands Separator`..



The constructs in the table may look very complicated for this simple task. However, after having set up the table once you can just copy/delete rows and insert new.

See more examples in `Table-calculations.fm`

Inserting results into document (output)

The standard method is assigning the result to a FrameMaker variable:

```
#something = Cos(Sqrt(@PI * 1/3)) {E5}"xmpl 08";
```

The resulting variable (`#something`) can be used the same way as any FM variable anywhere in the document: 5.20528×10^-1

Multiple statements are evaluated from left to right. Each assignment of a user variable changes its value which can be used in the next statement:

```
#T1 = Sqrt(17.); #T1 = #T1 * #T1; "#T1 = 1.7000×10^+1 "
```

Temporary variable

Formulas may assign a “temporary variable”, which is inserted directly after the marker[9] into the text[10]. This is specified by starting the formula with the equal sign:

```
= Sin(Sqrt(@PI * 1/3)) {E5}; "temporary result=
8.53844×10^-1 "
```

This marker contents can also be used anywhere. However, the resulting variables must not be moved or copied to another location, because only the original location is known to **FMcalc**.

Note: *If multiple statements in a marker define temporary variables then only the **first** one will be inserted into the text:*

```
= Sin(Sqrt(@PI)); = Atan2(17.5, 13.7); "=> 9.7974×10^-1 "
```

Note: *Temporary variables can not receive vectors. Hence the following is not valid:*

```
<= 17.9, 23.5, 11.3, 24.9 "invalid assignment of vector"
```

While *inside* **FMcalc** the decimal separator *always is a period*, in the resulting variable contents it may depend on the document settings (see *Manage document settings and debugging* on page 42).

Hence you can copy formulas across documents with different language (and most likely different settings for the decimal separator).

Example formatting

```
#t1 <= 17.9, 23.5, 11.3, 24.9; #result = MeanA (#t1)
{F2};
```

This defines a variable containing a vector which is used in the next statement[11]. If in the document the decimal separator is set to comma, then the FM variable `#result` contains the string `19,40`. `{F2}` means: decimal number with 2 decimals.

In this document we use the period: 19.40

9 If there is - e.g. from a previous evaluation run - already a variable, it will be removed first. If you need to protect a standard variable following directly the marker: put any text - at the minimum a thin space - between marker and variable.

10 The name of these variables is generated and contains the current time to be ‘unique’. These variables can be seen in FM, but not in **FMcalc**, because they are filtered from the variable list. See *Temporary results* on page 51 for details.

11 Vectors are variables which contain more than one value. They must be defined in an own statement and use a special assignment (<= rather than only =). A list of values is valid as argument for appropriate functions.

#series markers

Rationale

While the **#calc** functions provide a value in a variable, the idea for series came from the demand of 'decrementing counters', which is not possible with auto-number in paragraphs.

Ideas for usage

- Place numbering anywhere in the text, for example, to number sentences used in some legal text: Apply legal sub-number (semel, bis, ter, ...), a general form of textual numbering.
- Decrement numbers for paragraphs rather than increment as it is possible with autonumbering.
- Use circled numbers from the Unicode range Enclosed Alphanumerics for a list of call-out figures.

General mechanism

- The definitions are stored in markers of type **#series**. To a series with name *seriesName* a corresponding character format *seriesName* exists. Text which will receive the series items uses this character format (thus being replaced by the generated series items) [12].
- This character format must exist in the catalogue (or it will be created with "`All AsIs`").
- Numbering (placing the series) is done in the text flow *after* the **#series** marker.
- The text which will receive the first series item must be marked: It must contain the functional character `Optional Hyphen` (¬).
- The series evaluation is stopped at the end of the document. The same series can be started again by a marked text element. It is possible to nest series of different name.
- Calculation is initiated by appropriate menu entries.
- Insertion of series does not make sense on master pages - as it will be a constant on the corresponding body pages. **FMcalc** interprets these character formats only on body pages.

Example of series

```
Series= reverse_1; Scheme= Numeric; Start= 18.5; Incr= -4.1; Decimals= 2;
```

Series	This parameter defines the name of the series and the corresponding character format (`reverse_1`).
Scheme	Defines the type of the series: `Numeric`, `roman`, `ROMAN` or the name of a textual scheme (list of text items to be placed). There are restrictions on the following parameters depending on the Scheme.
Start	Starting value for numbered schemes or index of the first textual item for textural schemes.
Incr	Increment for the numbered schemes or increment of the index in textual schemes.
Decimals	For scheme type `Numeric` only: number of decimal places to be created for the placed values.
Resulting series	`18.50`, `14.40`, `10.30`, `6.20`, `2.10`, `-2.00`, `-6.10`, `-10.20`, …

12 See also *Allowed names for variables* on page 51.

Script invocation

Menu entries

The menu is inserted in the **Format** menu, because only this is also available when a book is active.

Unfortunately ExtendScript does not allow to define a position for the menu – it always appears at the end of the parent menu item (Format)[13].

Document menu

Format		Shortcut	See
Calculations in FrameMaker ...	Documentation	ESC q, m, d	
	(Re) calculate in document	ESC, q, c, d	page 10
	Handle #calc Markers	ESC, q, m, c	page 13
	Handle #series Markers	ESC, q, m, s	page 28
	Document Settings	ESC, q, m, z	page 35

Book menu

Format			
Calculations in FrameMaker ...	Documentation	ESC q, d	
	(Re) calculate in book	ESC, q, c, b	page 10

See also *Note about keyboard short cuts* on page 11.

Documentation

This `pdf` is displayed with your PDF viewer.

Handle #calc Markers

Insert/edit a **#calc** marker at the current location in the document. See *Handle #calc markers* on page 13.

Handle #series Markers

Insert/edit a **#series** marker at the current location in the document. See *Handle #series markers* on page 28.

(Re-)calculate in document

All **FMcalc** markers in the document are considered. First the `#calc` markers, then the `#series` markers are evaluated.

1. Markers are searched in the same sequence as with **Find > Marker of Type x**.
2. In tables Cell Numbering “Row First” or “Column First” influence the evaluation order for `#series` markers, but do not influence the cell addressing in `#calc` markers (see *Functions for cell addressing* on page 24).
3. Variables for results are defined/modified in the sequence in which they are assigned in the formulas of a `#calc` marker.

Note: *After recalculation you are informed about the number of markers handled.*

For details see *#calc markers* on page 49 and *#series markers* on page 53.

(Re-)calculate in book

- **FMcalc** does not handle nested books.
- All `*.fm` files in the book (also in groups or folders) are considered and handled in their DFS order[14].

13 This has been reported as bug 4110544.

14 DFS = Depth First Search. That is, a substructure (e.g. group) is entered and processed before it is left and the process continues on the upper level.

- **FMcalc** ignores the following open errors when opening a book component: old FM version, file is in use, missing fonts, referenced file not found. Hence you really use **FMcalc** only on 'clean' documents.
- **FMcalc** does not close book files after processing.
- **FMcalc** does not provide any means to transfer the user variables between the documents. They may have completely different contents in the documents.

See also Note about book operations hereafter.

Note: *After recalculation you are informed about the number of markers handled.*

Manage related lists

Both `#calc` markers and `#series` markers rely on building blocks. Often used items can be added to these lists. The lists of mathematical functions, numerical and physical constants can not be redefined. See *Document settings* on page 35.

Note about Refresh button

The following actions outside of the **FMcalc** palette require the use of the **Refresh** button:

- Manipulate **#calc** or **#series** markers with standard FM methods (that is, not within an **FMcalc** palette).
- Change text on the reference page **FMcalc**. The panel Document Settings has no Refresh button. Hence it must be closed and opened again, if it was open at the time of this manipulation.
- Rare other cases when the user is suspicious about automatic updating the internal tables.

Note about book operations

When using **FMcalc** take additional care when importing formats between book components:

- The Document setting relating to **FMcalc** (e.g. Decimal separator) may differ in the documents.
- The Document setting for superscripting and/or highlighting (character formats) may differ in the documents.
- Also the reference page **FMcalc** may differ in contents due to a different set of variables and text schemes etc.
- Although user variables in the documents may have the same name, their contents may be different.

Note about keyboard short cuts

Short cuts are active only with focus in the document. If a panel is active and you want to switch to or invoke another panel you either do that via menu or need to activate the document before typing the short cut.

(Re-)calculation

(Re-) calculation does not alter the contents of markers. If syntax errors or evaluation errors occur, evaluation is terminated

and the marker is presented in the handling dialogues for correction.

In **#calc** markers the evaluation may, however, alter the contents of variables

The purpose of **#series** markers is to alter the contents of text pieces of the corresponding character format.

Initialisation

At the start of the script the following is done:

1 Get program settings from the reference page in the template document.
2 Set up the notification mechanism to cope with changing documents and other situations.
3 Set up program constants (values of numeric constants, names of functions etc).
4 Write information about the script to the log file.
5 Set up the menu.

Dialogues

The dialogues are designed as palettes. They stay open while the user can change focus to the document and even change the active document (or to a book).

This requires to update contents in the dialogues depending on the current document (for example, the list of user variables). You use the **Refresh** button for this.

Document settings

Fresh document

A document which has never been used while FMcalc is installed in your environment does not contain any settings for FMcalc. Hence at open of this document the required settings are copied from the template document. This establishes a reference page FMcalc.

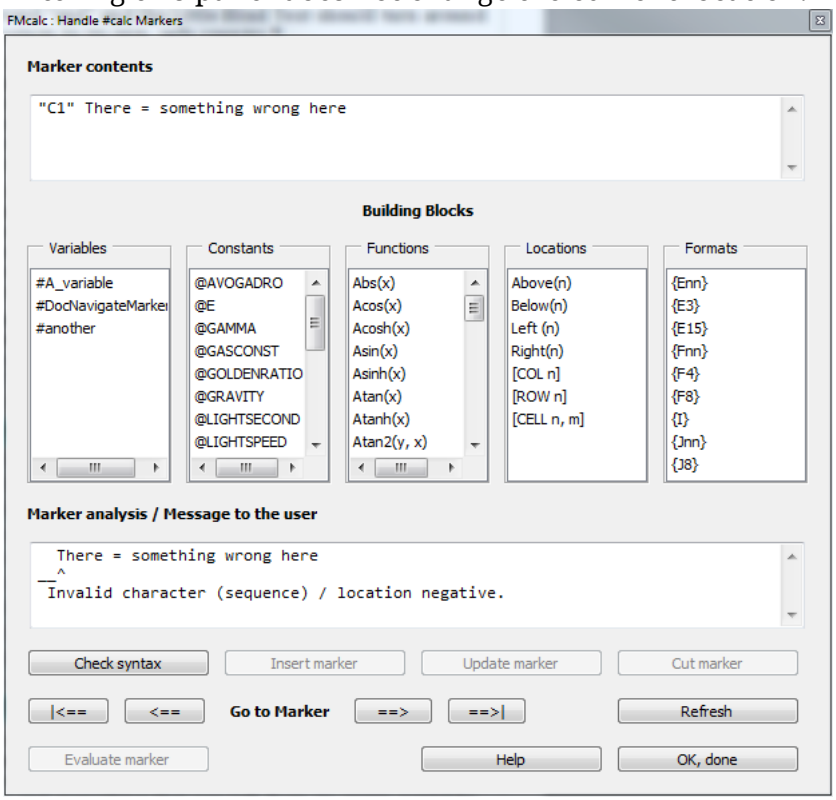
If you are in doubt whether the reference page is in sync with the version of FMcalc, delete the pertaining reference pages and close the document. At next open FMcalc will re-establish the reference page.

Used document

FMcalc takes all necessary information about document specific settings from the reference page FMcalc. The current settings can always be checked by opening the according panel (*Document settings* on page 35) with **ESC q m z**.

Handle #calc markers

Entering this panel does not change the current location.



Text fields

Marker contents

In this field the contents of the marker is assembled. Normally a formula is assigned to a variable. Names of existing variables can be used within the formula.

New **FMcalc** variables are defined if a new name occurs to the left of the equal sign and this name does not yet exist in the list of current variables.

Building blocks

Double clicking on a building block (e.g. a variable name) inserts it at the current location (or replaces a selection) in the marker contents. There it may be edited.

See *Document settings* on page 35 for managing lists.

Variables are defined in the document. In rare cases a change of the current document may require you to use the **Refresh** button.

Marker analysis / Message to the user

Syntax of formulas must be checked before the marker is placed in the document. Only after passing all tests the Insert / Update buttons become active.

This area is also used for general guidance of the user.

Error indications

For the following marker text ...

```
#foo = @PI/180.0; #angle = #foo * 33,17; = #VAT * #foo
"xmpl 10"
```

... the cursor points to the faulty position in the marker text:

```
  #angle = #foo * 33,17
                  __^
 Improper use of comma in numeric value
```

Function buttons

Check syntax

The following checks are done:

- Comments must be closed, parentheses, brackets and braces must be balanced.
- User variables or user constants must exist in the list.
- It is possible to use a variable which is defined prior to the current formula:

```
#var1 = @PI / 180; #degree_b = 312 * #var1 "xmpl 11";
```

- Function names or cell references must be correct.
- Format specifiers must be valid (starting with `E, F, I, J`).
- Operator symbols are -, +, *, /
- Comma must not be used in numeric values. As list separator it must be followed by a blank.

After correcting the marker text in the edit field you use this button again, until you get the OK message:

This activates other buttons (**Insert**, **Update**).

Note: *Errors such as completely missing operators or crossing parentheses can not be detected at this stage. They will create error messages during evaluation.*

Insert marker

The built **#calc** marker is inserted at the current document location. Hence you may need to define this location by placing the cursor in the document [15)].

Update marker

The currently selected **#calc** marker gets new contents from the dialogue.

Copy marker

No special button for this. You may copy the contents of the dialogue field and use it in a later operation.

Cut marker

This cuts the contents of the current marker into the clipboard. The marker is removed from the document. You can now use the clipboard contents for inserting a new marker or modify another marker.

Note: *There is no Undo for this action.*

⏮, ◀
▶, ⏭

Search for the first or previous **#calc** marker.
Search for the next or last **#calc** marker.
See *Go to Marker buttons* on page 6 for details.

Evaluate marker

This button is active only after syntax check *and* insert/update are activated. For details see *#calc markers* on page 49. Evaluation may reveal errors not detectable during syntax checking. This may create/update user variables and/or user constants. It also may insert a temporary variable behind the marker.

Help

The **Help** button opens this PDF document at *Handle #calc markers* on page 13.

Refresh

See *Note about Refresh button* on page 11

OK, done
Close (X top right)

Dismiss the dialogue. However, if an action was done already (e.g. insert a marker) this can not be undone.

15 The distinction between already existing and a new marker is based on the current selection. Hence do not select a single character to place a new marker. You may, however, select more than 1 character...

Elements of formulas

Syntax description	• | denotes alternatives (or), terms in brackets [ ] denote options. Literally needed brackets are denoted by these symbols: ⟦and⟧. • Possible repetition of an element or element group is denoted by the symbol ellipsis (…). For example "`decimal_figure` …".
statement	`[result] = term [format] [; statement] [;]`
vector assignment	`result <= value, value, …`
Note:	*The semicolon after the last statement is optional.*
comment	`"sequence of characters not containing double straight quote"`
result	Name of a variable receiving the numeric result.[16] If no variable name is provided in front of the equal sign than the result is inserted as text immediately after the marker. In tables these values may become 'input' to formulas by cell references.
term	`simple_term [operator simple_term]`
simple_term	`constant | variable | function_name(argument)`
operator	`+ | - | * | / | ( | )` [17]
constant	**keyword_0** | `integer | real` see *Keywords for constants (reserved variables)* on page 23.
sign	`+ | -`
decimal_figure	`0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`
integer	`[sign] decimal_figure [decimal_figure …]`[18]
real	`[sign] integer [FmtDecimal] integer [exponent]`
Note:	`FmtDecimal` *is a single character. It is defined on reference page* `FMcalc`.
exponent	`E|e integer`
variable	`#name | cell_reference`
Note:	*Within* **FMcalc** *certain rules apply for the names of variables and constants. See Variables on page 4 and Allowed names for variables on page 51.*
function_name	See the list *Functions and arguments* on page 16 [19].
value	`integer | real | variable | constant`
argument	`value | cell_reference`
argument_list	`argument [, argument_list]`
cell_reference	⟦**keyword_1**⟧ | ⟦**keyword_2**⟧ see *Functions for cell addressing* on page 24 and *Indices for cell addressing* on page 24.
format	Formatting string enclosed in braces (see *Formatting results for presentation* on page 26).

16 See also *Allowed names for variables* on page 51.
17 Operator ** (exponentiation) must be substituted by function power.
18 Integers are, however, always treated as real. Hence 5 /2 is 2.5, not 2 (as it would be with integer division).
19 Certain functions allow only certain types of arguments.

Functions and arguments

Remarks

- Function names of **FMcalc** are case sensitive. Their names start with an upper case initial (in contrast to JavaScript).
- The list of arguments uses comma, followed by blank as a separator to be coherent with `value_lists`. Function arguments may have these types:

  n — Integer (fraction of value is dismissed).

  x, y — Values (real, integer). They are, however, always treated as real. Hence 5/2 is 2.5, not 2 (as it would be with integer division).

  x_0, x_1, ... — Vector (list of values or value pairs).

  NaN — This is short for “Not a Number”. Input which can not be interpreted numerically gets this special value. It can also exist as results from functions, e.g. from `Sqrt(-5)`.

  - Math.xxx functions result in NaN for illegal arguments. This creates appropriate output (NaN)
  - In other cases a message in the dialogue is issued.
- Not all wanted functions are available as ExtendScript [1] functions. Column heading JS indicates the level of JS: 3 (similar to ES) and 6 (ECMA 6th edition, draft as 2015-03).
- The argument for the following functions is a list of values. The behaviour of these functions depends on the setting of `NaNskip` (see *Skip NaN in function arguments* on page 42): `MeanA, MeanG, MeanH, Median, Min, Max, Count, Hypot, Prod, Sum, SumProd`
- Superfluous arguments are detected during syntax check only (case 1). In evaluation superfluous arguments are just ignored (case 2):

```
#vect <= 17, 23, 31, 37; = Cbrt(#vect, 17.33) "case 1"
#vect <= 17, 23, 31, 37; = Cbrt(#vect) "case 2"
```

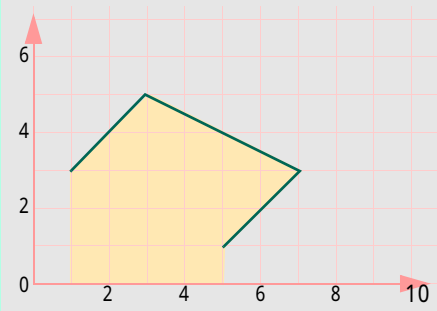
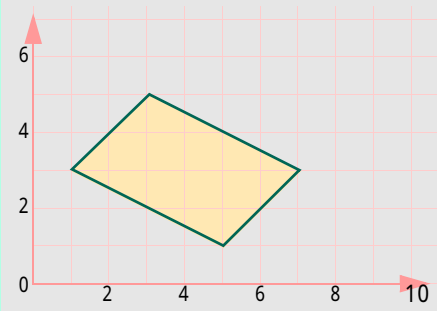
Function	Explanation	Restrictions, argument domain	JS	Example use, printciapal values
`Abs (x)`	Absolute value of x; the result has the same magnitude as x but has positive sign.		3	Abs (-33.99) ➔ 33.99
`Acos (x)`	Arc cosine of x.	-1 ≤ x ≤ 1	3	0 ≤ y π
`Acosh (x)`	Inverse hyperbolic cosine of x (areacosinus hyperbolicus)	+1 ≤ x ≤ ∞	6	0 ≤ y ≤ ∞
`Arc (deg,min,sec)`	Convert to radians	Abs (result) may be > 2π	-	Arc (229, 10, 59.2249883844) ➔ 4
`Area (x0, y0, …)`	Area between polygon and x-axis.	See *Area* on page 19	-	Area (0,0, 2,2, 4,2, 5,0) ➔ 7.00 Area (0,2, 2,4, 5,2, 2,0, 0,2) ➔ 10.0
`Asin (x)`	Arc sine of x.	-1 ≤ x ≤ 1	3	-π/2 ≤ y ≤ π/2
`Asinh (x)`	Inverse hyperbolic sine of x (areasinus hyperbolicus)	-∞ < x < ∞	6	-∞ < x < ∞
`Atan (x)`	Arc tangent of x.	-∞ < x < ∞	3	-π/2 ≤ y ≤ π/2; Atan (3.7) ➔ 1.3068
`Atanh (x)`	Inverse hyperbolic tangent of x (areatangens hyperbolicus)	-1 < x < 1	6	-∞ < x < ∞
`Atan2 (y, x)`	Arc tangent in full circle.	Two arguments. Result range is ±π	3	Atan2 (17.5, 3.7) ➔ 1.3624
`Cbrt (x)`	Cube root of x		6	Cbrt (777) ➔ 9.1933
`Ceil (x)`	Map a real number to the smallest following integer		3	Ceil (17.78); ➔ 18 Ceil (-17.78); ➔ -17
`Cos (x)`	Cosine of an angle	Argument range ±π/2	3	
`Cosh (x)`	Hyperbolic cosine	Argument range ±709	6	
`Count (x0, x1, …)`	Number of values in the list		-	Count (11, 22, 33, 44, 12, 32); ➔ 6
`Exp (x)`	Exponential function of x (e raised to the power of x, where e is the base of the natural logarithm). See *Keywords for constants (reserved variables)* on page 23:		3	Exp (@PI); ➔ 23.1407
`Factorial (n)`	Factorial (1 × 2 × 3 × … n)	n ≤ 170 for any, ≤ 17 for integer result.	-	Factorial (11); ➔ 39916800
`Floor (x)`	Map a real number to the largest previous integer		3	Floor (17.78); ➔ 17 Floor (-17.78); ➔ -18
`Fract (x)`	Fraction part of value	Result has same sign as argument	-	Fract (-17.89); ➔ -0.89
`GetData (file)`	Read data from text file.	See *GetData* on page 21 for details.	-	#T2 = ReadData ('datafile.txt');
`Hypot (x0, x1, …)`	Sqrt of the sum of value-squares	Reasonable results require at least 2 arguments	6	
`Log (x)`	Natural logarithm (logarithmus naturalis)	Argument must be positive	3	
`Log10 (x)`	Common logarithm (base 10)	Argument must be positive	6	
`Max (x0, x1, …)`	Largest of listed values	List of arguments	6	
`MeanA (x0, x1, …)`	Average of values in the list	See *Statistical function* on page 19.	-	MeanA (17.5, 33.7, 25.9, 45.8)
`MeanG (x0, x1, …)`	Geometric mean of values		-	
`MeanH (x0, x1, …)`	Harmonic mean of values		-	
`Median (x0, x1,…)`	Median of values in the list	See *Median* on page 19	-	Median (#Vector)
`Min (x0, x1, …)`	Smallest of listed values	List of arguments	6	
`Mod (y, x)`	y modulo x	Two arguments. See also *Mod* on page 21.	-	Mod (5.5, -2); ➔ -1.5
`Power (x, y)`	Exponentiation x^y	Two arguments	6	
`Prod (x0, x1, …)`	Product of values in the list	List of arguments	-	

Function	Explanation	Restrictions, argument domain	JS	Example use, printciapal values
`QSimps (x0,y0, …)`	Approximate integral of evenly or unevenly spaced data	See *QSimps* on page 21	-	QSimps (#Vector);
`Random ()`	Number value with positive sign, ≥0 but < 1, chosen pseudo randomly with approximately uniform distribution over that range. This function takes no arguments.		3	
`Rem (y, x)`	Remainder from y/x	Two arguments	-	Rem (5.5, -2); ➔ 1.5
`Round (x, n)`	n < 0: Rounding to the $\|n\|^{th}$ decimal place n=0: Rounding to the nearest integer value. n > 0: Rounding to 10^n See also *Round* on page 22		-	Round (55.55, -1); ➔ 55.6 Round (55.549, -1); ➔ 55.5 Round (55, 1); ➔ 60 Round (54.9, 1); ➔ 50 Round (1.005, -2); ➔ 1.0100
`Sign (x)`	+1 for positive x, -1 for negative x, 0 for ±0, NaN for NaN		6	
`Sin (x)`	Sinus of an angle	Argument range ±π/2	3	
`Sinh (x)`	Sinus hyperbolicus	Argument range ±709	6	
`StdDev (x0,x1, …)`	Standard deviation of values	Values are assumed to be normally distributed. See *Statistical function* on page 19.	-	
`Sqrt (x)`	Square root	Positive value	3	
`Sum (x0, x1, …)`	Sum of values	List of values	-	
`SumProd (x0,x1,…)`	Pairs of values are multiplied and the intermediate results are summed up: $x_0*x_1 + x_2*x_3 + x_4*x_5 + ...$	Even number of values	-	
`Tan (x)`	Tangent of an angle [rad]	Argument range ±π/2	3	
`Tanh (x)`	Tangent hyperbolicus	Argument range ±354	6	
`Trunc (x)`	Integer part of value (not rounded!)	Result has same sign as argument	6	Trunc (-17.89); ➔ -17

Details for special functions

Area

If the first and last value pair is identical, then the polygon is closed and the area is what it encloses.

Polygon is open	Polygon partially closed	Polygon is closed
		
`Area (0,0, 2,4, 5,2, 10,1);` → 2.0500×10^+1	`Area (1,3, 3,5, 7,3, 5,1);` → 2.0000×10^+1	`Area (1,3, 3,5, 7,3, 5,1, 1,3);` → 1.2000×10^+1

- The number of arguments must be even (that is x/y pairs).
- The number of x/y pairs must be > 1.

Statistical function

Average, Mean and Median

[Matthew Handy] The term **average** is a generic term: it covers lots of ways of indicating where the centre of a set of data is. It's an attempt to answer the question “what's a typical value for these data?” There is no such thing as “the” average.

The term **mean** is generally used to refer to the arithmetic mean, which is where you add up all the data values and divide by how many there are. Technically there are other means (geometric and harmonic).

Which average to use depends on the type of data you've got.

For example, consider incomes. The (arithmetic) mean tends to overstate the typical income because it gives undue weight to very high incomes. The **median**, on the other hand, is bang in the middle, so you might prefer to quote that.

Or, suppose I drive from A to B at 60 miles an hour, but drive back at 40 miles an hour. What's my average speed? 50 miles an hour, yes? No! Because I spent more time driving at 40 miles an hour (because I was driving more slowly over the same distance) than at 60 miles an hour. The **arithmetic mean** of 50 gives a misleading answer. The **harmonic mean** is what is needed here. That gives 48, which reflects the extra time spent driving at the slower speed.

Median

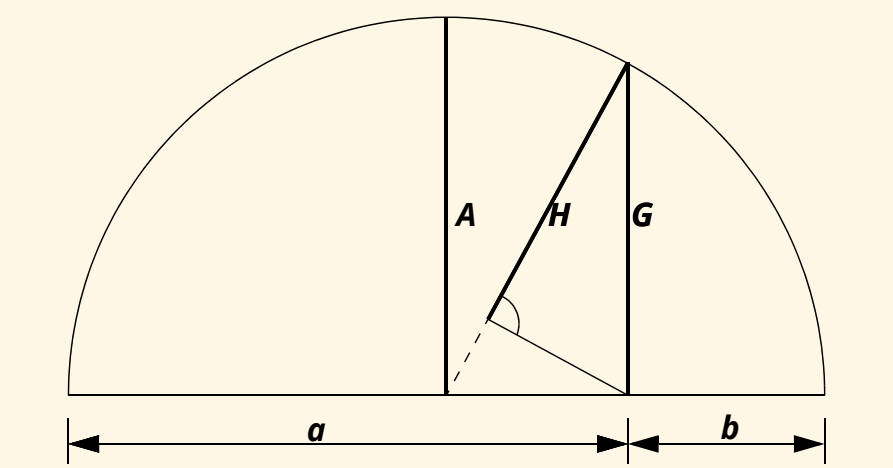
If observations of a variable are ordered by value, the median value corresponds to the middle observation in that ordered list. The median value corresponds to a cumulative percentage of 50% (i.e., 50% of the values are below the median and 50% of the values are above the median).[20)]

Pythagorean means

The Pythagorean means are the three “classic” means ***A*** (arithmetic mean), ***G*** (geometric mean), and ***H*** (harmonic mean). The figure below shows how these means on two elements ***a*** and ***b***

20 https://www.statcan.gc.ca/edu/power-pouvoir/ch11/median-mediane/5214872-eng.htm as of 2018-10-30.

could be constructed geometrically, and also demonstrates that ***H ≤ G ≤ A***.



[http://mathworld.wolfram.com/PythagoreanMeans.html].

FMcalc Function		Equation [a]	For 2 items	Comment
Arithmetic mean	**MeanA**	$\bar{x} = \frac{1}{n}\sum_{i=1} x_i$	$\frac{a+b}{2}$	The arithmetic mean of a set of values is the quantity commonly called "the" mean or the average `MeanA(17, 23, 25, 44, 66, 43, 12, 22);` ➔ `31.5000`
Geometric mean	**MeanG**	$G = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$	$\sqrt{ab}$	The easiest way to think of the geometric mean is that it is the average of the logarithmic value s, converted back to a base 10 number.`=EXP(AVERAGE(LN(A1:A200)))` ➔ values must be > 0. See *MeanG (v1,v2,...,vn)* on page 57. `MeanG (4, 8, 3, 9, 17);` ➔ `6.8138`
Harmonic mean	**MeanH**	$\frac{1}{H} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{x_i}$	$\frac{2ab}{a+b}$	The harmonic means of the integers from 1 to n for n=1, 2, ... are 1, 4/3, 18/11, 48/25, 300/137, 120/49, 980/363, ... ➔ values must be > 0. See *MeanH (v1,v2,...,vn)* on page 57. `MeanH (1, 2, 3, 4, 5, 6, 7);` ➔ `2.6997`
Relationships		$A \geq G \geq H$	$G = \sqrt{AH}$	
Standard deviation	StdDev	$S_{n-1} = \sqrt{\frac{1}{n-1}\sum_{i=1} (x_i - \bar{x})^2}$		If the data is a random sample then the denominator `N-1` is used. If the data comprise the entire population then instead of the *sample standard deviation*, the *population standard deviation* with the denominator is `N` instead of `N - 1` is used. It is rare that measurements can be taken for an entire population, so, by default, statistical computer programs calculate the sample standard deviation. `StdDev(17, 23, 25, 44, 66, 43, 12, 22);` `18.0238`

Median
See also *Median* on page 19.
	Median			The statistical median is an order statistic that gives the "middle" value $\tilde{x}$ of a sample. More specifically, it is the value $\tilde{x}$ such that an equal number of samples are less than and greater than the value (for an odd sample size), or the average of the two central values (for an even sample size).
				`Median (26.1, 25.6, 25.7, 25.2, 25.0)` `25.60`
				`Median (26.1, 25.6, 25.7, 25.2, 25.0, 24.7)` `25.40`

a. mathworld.wolfram.com and [14].

GetData

For greater flexibility than the assignment of vectors in a **#calc** marker the function `GetData` allows to read from a text file:

```
#Tmany <= GetData ('Data-2.dat');
```

Data-2.dat

This example file contains 3 lines which define 18 items of which 3 are not numeric. The symbol » denotes a TAB.

```
0.48353576660156» -456e3, 17.53» 1234.5678» 33.9e9» -
177.8
-456e3» Invalid» 0.000234e3» 17.53» 1234.5678» 33.9e9
-177.8» -456e3» Invalid» 0.000234e3» 17.53» …
```

`settings.NaNskip`

With `NaNskip = true` also non-numerics are read in and marked as `NaN`. These values are skipped in functions.

With `NaNskip = false` the input process is terminated at the non-numeric value and a message is issued.

File names

`datafile.txt` In directory of current document.

`..\\datafile.cfg` one level up (parallel to doc. dir.)

`..\\ddd\\datafile.ini` In subdirectory `ddd` which is parallel to the document directory.

`D:\\ddd\\sample.dat`Absolute path on drive `D`.

File contents

The first line may specify a comment indicator of arbitrary length. This string starts in position 1 of the line and must not be pure numeric:

Uncommented file	Commented file

`17.53 » 1234.5678`
`33.9e9`
`-456e3 » 0.00000056`
`0.000234e3 » 33.9e9 » 37.99`
	`;`
	`; Data-1.dat: demo file`
	`17.53 » 1234.5678`
	`33.9e9`

Data in a line are separated by *one* TAB (denoted above by »)

Since the decimal separator depends on the language of the document, both period or comma are allowed in data files.

Note: *Variable contents is checked against the 1024 character limit.*

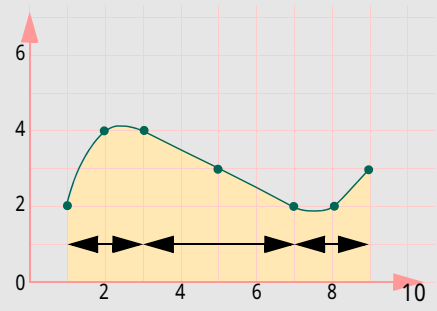
Mod

The result of modulo operator takes the sign of the divisor, not the dividend (as it is with Remainder).

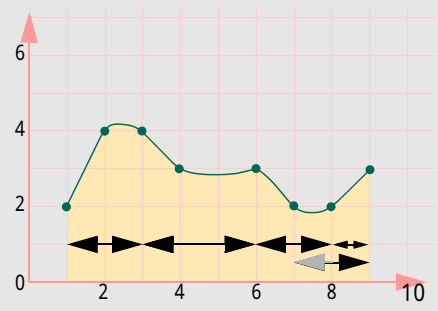
QSimps

Function Quasi Simpson approximates integration for uneven (or even) intervals by successive parabolas[21)].

Odd number of points



Even number of points



21 This function is based on a Fortran program courtesy by Bruce Cameron Reed, Physics, Alma College, Michigan <reed@alma.edu> (2016).

Odd number of points	Even number of points
All parabola segments span 3 points	All but the last parabola segment span 3 points. The last segment takes the parabola parameters from the last 3 points, but the area is calculated only between the pre-to-last and last point.
`= QSimps (1,2, 2,4, 3,4, 5,3, 7,2, 8,2, 9,3);` ➔ 2.3667×10^{+1} (Area: 2.3500×10^{+1})	`= QSimps (1,2, 2,4, 3,4, 4,3, 6,3, 7,2, 8,2, 9,3);` ➔ 2.5000×10^{+1} (Area: 2.3500×10^{+1})

Note: *To judge the accuracy of the method keep in mind that measurements always are of limited accuracy. It's not reasonable to start with 2 decimal points and expect a result accurate up to the 4th decimal...*

The following restrictions apply to the arguments:

- The number of arguments must be even (that is x/y pairs).
- The number of x/y pairs must be > 2.
- Successive x-values must not be equal. This would be a singularity.
- X-values must monotonically increase or decrease. The direction of mono tonicity must not change. That is, you can not define a closed curve analogous to the Area function.

Note: *To get the value of a closed area you can set up an 'upper' area and a 'lower' area and build the difference.*

Power

JS-function name is `pow`.

Round

JS provides only `round (x)` which returns the number value that is closest to `x` and is equal to a mathematical integer.

The process of approximating a quantity, be it for convenience or, as in the case of numerical computations, of necessity. If rounding is performed on each of a series of numbers in a long computation, round off error[22)] can become important, especially if division by a small number ever occurs.

FMcalc's **Round** function just rounds. User specified or default output setting may show more or less decimals (see *Formatting results for presentation* on page 26).

```
Round (1.005, -2); ➔ 1.0100
```

22 Round off error is the difference between an approximation of a number used in computation and its exact (correct) value. In certain types of computation, round off error can be magnified as any initial errors are carried through one or more intermediate steps.
An awful example of round off error is provided by a short-lived index devised at the Vancouver stock exchange (McCullough and Vinod 1999). At its inception in 1982, the index was given a value of 1000.000. After 22 months of recomputing the index and truncating to three decimal places at each change in market value, the index stood at 524.881, despite the fact that its "true" value should have been 1009.811 [mathworld.wolfram.com/RoundoffError.html].

Keywords for constants (reserved variables)

Keywords are all upper case. Some are available as JavaScript math properties (JS version in last column).

Keyword_0	Explanation	Value approx [a]	JS
@AVOGADRO	Avogadro's constant [mol^{-1}]	6.02214076×10^{23}	-
@E [b]	Base of natural or Napierian logarithms	2.71828182845...	3
@GAMMA [c]	$\gamma = 1 + {}^1/_2 + {}^1/_3 + {}^1/_4 + {}^1/_5 + {}^1/_6 \ldots + {}^1/_n - \log n$	0.57721566490...	-
@GASCONST	Molar or universal gas constant [$JK^{-1}mol^{-1}$]	8.3144598	-
@GRAVITY	Gravity constant on earth [m/s^2]	9.80665	-
@GOLDENRATIO	$\varphi = (1 + \sqrt{5})/2$	1.618033988749...	-
@LIGHTSPEED	Speed of light [m/sec]	2.99792458×10^8	-
@LIGHTSECOND	Light-second [m]	2.99792458×10^8	-
@LN10	ln (10)	2.302585092994...	3
@LN2	ln (2)	0.693147180559...	3
@LOG2E	base-2 logarithm of e	1.442695040888...	3
@LOG10E	base-10 logarithm of e	0.434294481903...	3
@MVOLUME0	Molar volume of ideal gas at 1bar and 0°C [m^3mol^{-1}]	$2.2710\ 980\times10^{-2}$	-
@MVOLUME25	Molar volume of ideal gas at 1 bar and 25°C [m^3mol^{-1}]	2.4789598×10^{-2}	-
@PI	Ludolf's constant	3.141592654...	3
@RAD2DEG	Radians to Degrees conversion	57.2957795130...	-
@SQRT1_2	Square root of ½	0.707106781186...	3
@SQRT2	Square root of 2	1.414213562373...	3
@SQRT3	Square root of 3	1.73205080756888	-

a. Sources: Wikipedia; [14] and [16]
b. Euler's number
c. Euler-Mascheroni constant.

Using cell contents (input)

Note: *These functions are only valid for* **#calc** *markers within tables.*

- Only cells in body rows are considered for relative and absolute addressing. Cells in the heading rows or footing rows[23], however, may contain a #calc marker.
- Relative addressing takes the current cell (containing the **#calc** marker) as reference (index = 0). Its contents can only be retrieved with absolute addressing.
- For absolute addressing the top left body cell is CELL 1, 1.
- Cell may only contain a single value, not a vector!
- Since the decimal separator depends on the language of the document, both period or comma are allowed in cells.

Restrictions on cell contents

- Only the first paragraph in a cell is considered. This allows to comment on the numeric value in successive paragraphs.
- From cells with autonumbered paragraph format only the part behind the autonumbering is considered.
- While markers may exist in the first paragraph, ordinary variables must not – most would be interpreted as `NaN`.

Functions for cell addressing

The current cell contains the marker with the current formula. Only cells in the table body are considered for formula input. Cells in the table header or table footer can only display results via **FMcalc** variables.

Keyword_1	Explanation
`Left (n)`	n elements (cells) to the left of the current cell .
`Right (n)`	n elements (cells) to the right of the current cell .
`Above (n)`	n elements (cells) above the current cell .
`Below (n)`	n elements (cells) below the current cell .

`n` must not be negative. If `n` is left out or zero, then all elements to the left etc. are considered.

Example

```
#cost = Prod(Left(2)); #total = Sum(Above())"xmpl 12";
```

Cell ranges

Empty cells are interpreted as `NaN`[24]. Whether these are skipped (e.g. when summing up) depends. See *Skip NaN in function arguments* on page 42.

Also the table boundary terminates the cell range. For example, if the table has 6 columns and the **#calc** marker in the first column requests `Right(6)`, then only 5 items are handled.

Indices for cell addressing

Relative and absolute addressing for a single cell[25] within table body:

Keyword_2	Explanation
ROW *n*	Cell *n* in same column[a] as the active cell (with **#calc** marker)
COL *m*	Cell *m* in same row[a] as the active cell (with **#calc** marker)
CELL *n, m*	Absolute addressing with *row, cell number*.

23 Although heading rows and footing rows are repeated in multi-page tables, FM does not repeat markers in it.
24 Same apply to cells with non-numeric contents in the first paragraph.
25 Cell numbering “Row First” or “Column First” has no influence. This is relevant only for the automatic ¶ numbering.

a. This may sound mixed up - but look at the table on the next page.

The following depicts the numbering scheme of Excel, which is mimicked in **FMcalc**. The first line in a cell is the absolute address. The second line is relative to the current (filled red) cell.

Excel scheme

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	R1C1 R[-2]C[-2]	R1C2	R1C3	R1C4	R1C5 R[-2]C[2]
Row 2		R2C2 R[-1]C[-1]		R2C4 R[-1]C[1]	
Row 3			Current cell		
Row 4		R4C2 R[1]C[-1]		R4C4 R[1]C[1]	
Row 5	R5C1 R[2]C[-2]				R5C5 R[2]C[2]

FMcalc cell numbering

FMcalc provides relative addressing only for the same row or column as the current cell.

FMcalc scheme

	Col 1	Col 2	Col 3	Col 4	Col5
Row 1	CELL 1, 1	CELL 1, 2	ROW -2	CELL 1, 4	CELL 1, 5
Row 2			ROW -1 Above (2)		
Row 3	COL -2 Left (2)	COL -1	Current cell	COL 1 Right (2)	COL 2
Row 4			ROW 1		
Row 5	CELL 5, 1	CELL 5, 2	ROW 2 Below (2)	CELL 5, 4	CELL 5, 5

If cell addressing points to a cell outside the table body then this creates an error during syntax checking.

Relative addressing

`[0]`	The current cell. This can not be accessed.
`[COL 1]`	Cell in same row, just after the current cell
`[COL -1]`	Cell in same row, just before the current cell
`[ROW 9]`	Ninth cell below the current cell.
`[ROW -3]`	Third cell above the current cell.

Absolute addressing

Be aware to insert a blank after the comma (syntax requirement).

`[CELL 1, 1]`	First cell in first body row.
`[CELL 2, 3]`	Third cell in second body row.
`[CELL 9, 9]`	In a table with less than 9 body rows or columns this will create an error, because the cell can not be reached.

Example

```
#items <= [ROW-3], [ROW-4] "create vector"; #cost =
Prod(#items) "xmpl 13 - all items multiplied together";
#items <= [COL 2], [COL 3], [COL 7]; "create vector";
#subtotal = Sum(#items) "xmpl 14";
```

Formatting results for presentation

Inserting temporary results

Temporary results (generated variables) are placed immediately after the generating **#calc** marker. See *Temporary results* on page 51.

These variables must not be copied or moved, because **FMcalc** keeps only the place behind the generating marker.

Specify formatting

The results presented in a variable can be formatted by a special notation at the end of the statement: {*format_string*}

Format string Gen. [a]	Exmpl.	Explanation	-1234.567 repres. with exmpl def.
En	E4 e12	Scientific notation with mantissa length n+1 (plus a sign if necessary). The scientific notation always presents a "normalised" mantissa (one non-zero figure prior to the decimal point).	-1.2345×10^{+3}
Fn	F2 f4	Fixed point floating number: decimal value with n decimal places.	-1234.57
I	I i	Integer (with sign). No length specification allowed.	-1234 54321
Jn	J8 j7	Integer with leading zeroes yielding a total length of n figures (sign is ignored)	00001234
R	R	Upper case Roman numerals. These can not be negative or > 4000. No length specification allowed.	MCCXXXIV
r	r	Lower case Roman numerals. These can not be negative or > 4000. No length specification allowed.	mccxxxiv

a. n denotes an integer number without sign.

Example

```
#total = Round(Sum(Above),-2) {F2}"xmpl 15";
```

Formatting details

No formatting specified

If no formatting is specified, the default formatting as specified on reference page `FMcalc` is applied. Integers are always presented with all of their figures.

Scientific notation {E}

Scientific notation uses a character format for superscripting to provide the usual appearance: 1.234567×10^{-4}. In formula definitions the simple form can be used: `1.234567E-4` (or `e`).

Fixed point floating number {F}

This is the standard representation in text using a decimal symbol: 12 345.67. See also *Thousands separator* on page 27.

Integers {I}

Integers are whole numbers without a decimal part. Values in the range of "safe integer" display all figures: -9 007 199 254 740 991. See also *Thousands separator* on page 27. Values above the range of "safe integer" use scientific notation: 9.007199254740990×10^{+15} [26].

Integers {J}

Integers with leading zeros: 00012. This can only be a value without sign.

26 The last figure is incorrect, should be 2. This seems to be the result of conversion.

Roman numerals {R} {r}
The value range is restricted to 1 ... 4000. Outside this range the formatting is not agreed upon.

Decimal separator
Within the application **FMcalc** the decimal separator is the period[27]. Hence formulas are set up with this:

```
#VAT = 15.6 "percentage of Value Added Tax for country x";
```

Numeric values in table cells and data files my use either the period or the comma for the decimal separator.

For the output to be placed in FM variables the setting for the decimal separator (`FmDecimal`) is observed[28]. Thus during 'input' and 'output' this separator may be converted.

Thousands separator
Insertion of thousands separator follows typographic rules: it is inserted only, if more than 4 figures left to the decimal point exist: `12'423`, but `2423`. [29] See *Document settings* on page 42.

This formatting is applied only to I and F formats.

Note:
If Thousands Separator is not defined, nothing is inserted.

Highlighting
Independently of the output format the user may specify a highlight for all output generated by **FMcalc**. See *Document settings* on page 42. In this document it is this colour.

Space not sufficient

- If for the J-format space (n) is to small for the value the last figure is replaced by a question mark:
  For example {J6}: ➔ 001234; {J3}: ➔ 12?
- Values > 1.0×10^{20} are *not* formatted with I-format. The JS functions `Ceiling and Floor` force an exponential display:

```
17.234567123456790123e14 ➔ 1723456712345679
17.234567123456790123e15 ➔ 1.723456712345680×10+16
```

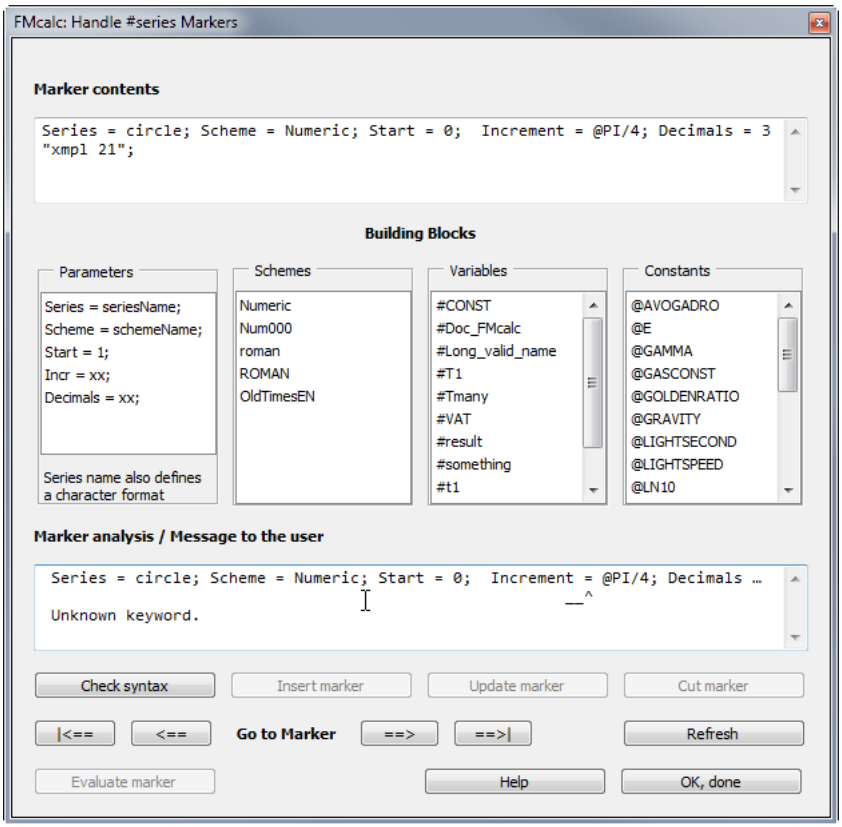
27 This distinction allows to use the comma as list separator which eases processing.

28 The decimal separator (period or comma) is not taken from the system locale, because it may be document specific. It is defined on reference page FMcalc. On 'input' both forms are accepted by means of a regular expression.

29 As default Thin Space (`\st`) and Non-Breaking Hyphen (`\+`) are used to prevent the number from breaking across lines. These are independent of font as thy are not characters but FM functions.

Handle #series markers

Entering this panel does not change the current location.



Text fields

Marker contents

In this field the contents of the marker is assembled. The series marker needs various parameters, which are separated by a semicolon. The name of the series 'links' to the items in the text by a character format of the same name.

A **#series** marker does not define any variables. Variables can only be used in this context.

Note: *Although the order of parameters defining the series can be arbitrary it makes sense to use the order given in the Parameters list.*

Building blocks

Double clicking on a building block (e.g. a variable name) inserts it at the current location in the marker contents. There it may be edited.

See *Document settings* on page 35 for managing the lists of building blocks.

Variables are defined in the document. In rare cases a change of the current document may require you to use the **Refresh** button.

Marker analysis / Message to the user

The syntax of the marker text is checked before the marker is placed in the document. Only after passing all tests the Insert/ Update buttons become active.

This area is also used for general guidance of the user.

Error indications

For the following marker text …

```
Series = MySeries; Start = -12.5; Incr = -1; Scheme =
roman; "xmpl 20"
```

… the cursor points to the faulty position in the marker text:

```
  Series = MySeries; Start = -12.5; Incr = -1; Scheme =
...
                                    __^
 Negative Start not valid for scheme = roman/ROMAN.
```

Function buttons

Check Syntax

The following checks are done:

- Comments must be closed, parentheses must be balanced.
- User variables or user constants must exist in the list of building blocks. They are defined in a **#calc** marker. These are always evaluated before any **#series** marker is evaluated.
- Constants must exist in the list of building blocks. For example `@PI`.
- Valid operator symbols (for parameters `Start` or `Incr` only) are `-`, `+`, `*`, `/` and `(`
- Comma must not be used in numeric values.

After correcting the marker text in the edit field you use this button again, until you get the OK message.

This activates other buttons (**Insert**, **Update**).

Note: *Errors such as completely missing operators or crossing parentheses can not be detected. They will create error messages during evaluation.*

Insert marker — The built **#series** marker is inserted at the current document location. Hence you may need to define this location by placing the cursor in the document [30].

Update marker — The currently selected **#series** marker gets new contents from the dialogue.

Copy marker — No special button for this. You may copy the contents of the dialogue field and use it in a later operation.

Cut marker — This cuts the contents of the current marker into the clipboard. The marker is removed from the document. You can now use the clipboard contents for inserting a new marker or modify another marker.

Note: *There is no Undo for this action.*

⏮, ◀ Search for the first or previous **#series** marker.
▶, ⏭ Search for the next or last **#series** marker.
See *Go to Marker buttons* on page 6 for details.

Evaluate marker — This button is active only after syntax check *and* insert/update have been activated. For details see *#series markers* on page 53. Evaluation may reveal errors not detectable during syntax checking.

Help — The **Help** button opens this PDF document at *Handle #series markers* on page 28.

Refresh — See *Note about Refresh button* on page 11

OK, done
Close (X top right) — Dismiss the dialogue. However, if an action was done already (e.g. insert a marker) this can not be undone.

Creating series in text

- Define a **#series** marker at a place after which the series will be managed. It must be prior to the place holders.

30 The distinction between already existing and a new marker is based on the current selection. Hence do not select a single character to place a new marker. You may, however, select more than 1 character...

- You are prompted to define a character format with the same name as the series. It may be “all as-is”, because only the name is relevant.
- For each item of the series create a place-holder (e.g. xxx) and apply the corresponding character format.

Note: *If the place holder is followed by space, this must be replaced by a non-breaking space (***CTRL+space***). Otherwise the space will disappear*[31]*.*

- For the first item to be placed (start of series) the place holder must contain the special symbol ¬ (optional hyphen). This symbol is only visible in your document with View option **Text Symbols** = ON.

Note: *If this start-indicator is not present, then numeric series will display all items as* `NaN` *and text series will display all items as* `UnDef`.

- You can directly evaluate the marker after syntax check and insertion/update. This will modify the place holders to the series items.
- The replacement of the place holders stops at the end of the document (book components are independent).

Contents of #series marker

The **#series** marker must contain the following parameters. They are separated by a semicolon. Hence there can be only one definition within a series marker.

Parameter	Example definition	Comment
`Series`	`counter_7`	Name of the series. This is identical to an (existing) character format, which is used to define the location of the output. This value is of course case sensitive.

`Scheme`	`Numeric`
	`roman`
	`ROMAN`
	`Num000`
		Presentation format of the output (case sensitive). The value 17 can thus be presented as `17`, `xvii`, `XVII` or `0017`. Default = Numeric. For details see *Other parameters* on page 31

	`Text_scheme`	For details see *Parameter Scheme* on page 31
`The following parameters depend on the scheme. See` *Other parameters* `on page 31`		
`Start`	`17.5` `23`	Start value, to be applied to the first occurrence of the series format. Default = +1.
`Incr`	`#T1 * 2` `-1`	Formula defining the next item to be output. Default = +1.
`Decimals`	`5`	Number of decimal figures. Relevant for `Numeric` and Num000 scheme only. Default = 0 (Numeric) or 4 (Num000).

See also *Examples of series* on page 32.

Series name

Naming the series and the corresponding character format should follow a scheme to guide the document editor. For example the name could start with `Series_` or `Counter_`; but it

31 This is the consequence of **Smart Spaces**. Hence if this option is OFF, spaces will not disappear.

should not start with @ or # to avoid confusion with **FMcalc** constants or variables.

FMcalc allows only names containing alphanumeric characters and the underscore symbol. That is, the same syntax as for user variable names, but not starting with #.

Note: *It is important to know that in FrameMaker only the last applied character format is known (and displayed in the status line). Hence the special format used for series must be applied to the place-holder value* as the last character format.

Parameter Scheme

Text schemes define a series of text items which are used for the first, second etc. occurrence in the relevant character format. These user defined text schemes may have names such as `Greek_chars`, `Weekdays`, `Months`, `EnumEN` or `Rainbow`.

See *Manage text schemes* on page 40.

Other parameters

Scheme	Start	Incr	Decimals
Numeric	- A formula may be used instead a numeric value (see *Formula in parameters* hereafter). - Result may be a positive or negative integer or real (with decimals). - Default value for both parameters is +1.		Whole number (integer) Default is 0
Num000	- A formula may be used instead a numeric value (see *Formula in parameters* hereafter). - Result is truncated to whole number. - Default value for both parameters is +1.		This parameter is interpreted as "number of places". Items are always treated at integer. Default is 4.
	Result must be > 1	Result may be ±n.	
roman ROMAN [a]	- A formula may be used instead a numeric value (see *Formula in parameters* hereafter). - Result is truncated to whole number. - Default value for both parameters is +1.		Ignored
	Result must be > 1	Result may be ±n.	
text scheme	- A formula may be used instead a numeric value (see *Formula in parameters* hereafter). - Result is truncated to whole number. - Default value for both parameters is +1.		Ignored
	Result must be > 1.	Result may be ±n[b].	

a. Negative results during evaluation are displayed as NaN, because negative values are not defined in the Roman numeral system.

b. n is checked against the number of items in the scheme.

Formula in parameters

- Numeric values may be integer or real. They are, however, always treated as real. Hence 5 /2 is 2.5, not 2 (as it would be with integer division). Only the final result will be truncated to an integer.
- **FMcalc** variables may be used in place of a numeric value.
- Basic arithmetic functions (+, -, *, /) are allowed, also parentheses to clarify precedence.
- Not allowed are mathematical functions (`xxx()`) and mathematical and physical constants (`@xxx`). If you need such, define an appropriate user variable or user constant in a **#calc** marker.

Series are evaluated after any **#calc** evaluations have been done. Hence the variables used in the series are always the last current values of the **#calc** evaluations.

Interpreting the examples

In the examples in this section the series are indicated by ‘pseudo-character formats’, that are colours. For reference the colour is also applied to the name of the series.

Examples of series

Please note the special symbol (⊤) in the first item of the series to (re-) start the series. For the purpose of visualising this symbol also in pdf, a visible copy of it (special font) is inserted in front of the real one in these examples.

This symbol must be in the range of the character format used for the series[32]. This symbol is only visible in your document with **View > Text Symbols** ON.

A simple idea

```
Series = circle; Scheme = Numeric; Start = 0;  Incr = @PI/4; Decimals = 3 "xmpl 21";
```

If we have the marker here ⊤, the first item in this series will be ⊤0.000 the next one will be 0.785, then comes 1.571and it continues with 2.356. And so on and so forth.

You may restart the series: ⊤0.000, then 0.785and so on.

Note: *The increment can not refer to a previous item to support a geometrical series. The following is not valid (with the intention to get 1, 7, 19, 43, ...):*

```
Start = 1; Incr = (*2 + 5); "invalid increment"
```

Reverse numbering in a table

```
Series = reverse_2; Scheme = Numeric; Start = 18; Incr =-4 "xmpl 22";
```

The marker may be placed in the table head	Just text to demonstrate a table.
⊤18	This value is the start value
14	decremented by 2
10, 6	and so on
2	the last positive
-2	the scheme extends beyond zero
-6	etc.

The series may continue outside the table. See *Finito la commedia* on page 34 To get the series again elsewhere, you use the same character format and to restart it using the special symbol for the first item.

Textual numbering

This is a series for your ultimate planning without stress:

```
Series = zeiten; Scheme = OldTimesDE; Start = 1; Incr = 1 "xmpl 23";
```

For the definitions of the scheme see *Manage text schemes* on page 40.

⊤in alten zeiten	This starts the textual numbering.
früher	And here it is continued.
vor kurzem	Shortly ago.

32 If this symbol is *not* within the scope of the character format the values placed will all be NaN for the first use of the series or just a continuation of the last use.

heute	Today.
morgen	Tomorrow.
nächste woche	Next week.
in zukunft	In the future.
manjana	It may happen.
in der ewigkeit	In eternity.
in alten zeiten	Restart due to exhausted list.

Circled numbers

Assume call-out figures used in a graphic which are described with the following legend.

```
Series = circled; Scheme = CirledNumbers; Start = 1;
Incr = 1 "xmpl 24";
```

For the definitions of the scheme see *Manage text schemes* on page 40.

① Gimmick to do all what you can not do yourself.

② Ultimate gimmick to impress your girlfriend.

③ Top secret gimmick used to impress your CEO.

To work correctly this series requires a character format (or the ¶-format) defining a Unicode font which includes the Block `Enclosed Alphanumerics` (code range `U+2460 … U+2473`), for example `Calibri` or `Consolas`.

Numbering of legal articles

```
Series = legal_1; Scheme = Legal; Start = 2; Incr = 1
"xmpl 25";
```

For the definitions of the scheme see *Manage text schemes* on page 40.

Legal text often uses a sub-numbering with Latin counting, with the speciality of leaving out the first term (`semel`). The example is from the Swiss federal Constitution (BV 1999):

Art. 22	Waffenplätze
Art. 22bis	Zivilschutz
Art. 22ter	Eigentumsgarantie
Art. 22quater	Raumplanung

Esoteric example

This example uses the constant @PI and the variable #VAT which has value 7.6).[33)]

```
Series = artificial; Start = #VAT; Incr = 3.14; Scheme =
Numeric; Decimals = 2 "xmpl 26";
```

7.600	This was the VAT value in Switzerland for a long time.
10.74	Although artificial, it comes close to other countries VAT value.
13.88	And even this one is superseded by many countries.

Numbering with leading zeros

```
Series= LeadingZeros; Scheme= Num000; Start= 21; Incr= -
3;
```

Leading zeros are sometimes requested by people addicted to old-style programming habits.

0021	The first item to be calculated.

33 For ‘approximation’ German slang knows the proverb: Pi mal handgelenk (Pi times wrist) obviously a Dada-istic expression..

0018 The second item to be calculated.
0015 The third item to be calculated.
0012 The next item to be calculated.
0009 The last item to be calculated.

Finito la commedia

And here we continue the sequence `reverse_2`: `-10`.

Document settings

Both **#calc** markers and **#series** markers rely on building blocks. The implemented mathematical functions, numerical and physical constants can not be redefined.

Tabs in this dialogue

Manage user variables and constants on page 35.
Manage input locations on page 38.
Manage output formats on page 39.
Manage text schemes on page 40.
Manage document settings and debugging on page 42.
See also *Note about Refresh button* on page 11.

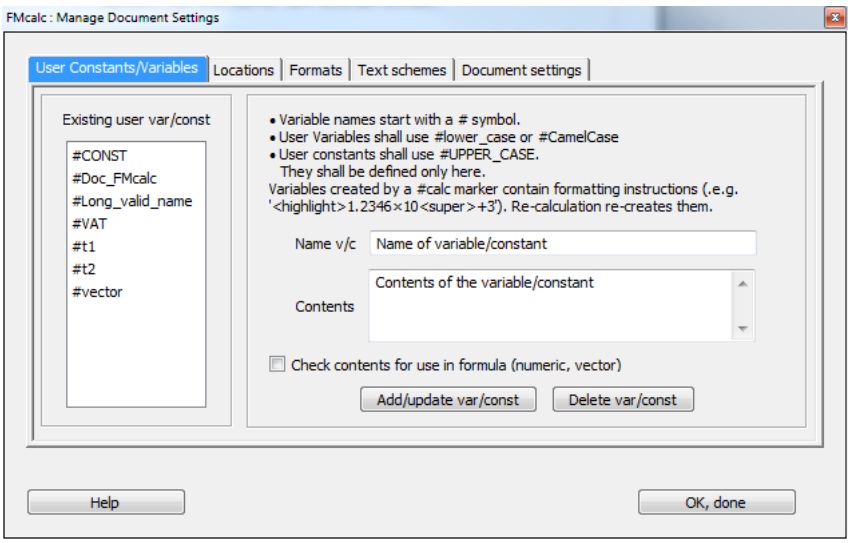
Manage user variables and constants

Help button

The **Help** button opens this PDF document at *Document settings* on page 35.

Cancel
Close (X top right)

Dismiss the dialogue without further action. However, if an action was done already (e.g. define a variable) this can not be undone.



User variables and user constants

Note: *If panel* **Handle #calc markers** *or panel* **Handle #series markers** *is open and a variable/constant is defined or deleted, then the corresponding list of variables in the panels is not updated.*

Mandatory naming rules

- **FMcalc** only recognises user variables starting with the # symbol.
- The only special character allowed is the underscore: `#a_valid_name`.

Helpful rules

User variables use names in all lower-case (after the initial # symbol) or CamelCase. A user variable can be defined in this panel in advance — without any significant contents. It may, however, also contain just numeric data and/or comment — same as a user constant. This will exist only as long as it is not overwritten by the execution of an **FMcalc** marker.

```
#UserVar_1 = 0 "value will be modified by #calc marker(s)"
```

User constants use names in all upper-case (after the initial # symbol). A user constant must contain only numeric data and it may contain a comment[34] (see *Comments in formula* on page 7), for example:

```
#CONST = 17.71 "The mysterious figure in the galaxis"
```

User constants are intended to be used only to the right side of an = sign in a formula.

Define a new variable

1 Type the desired name into the field **Name**.
2 Type the definition of the constant or variable into the field **Contents**:
   - Independently from the document setting use only the period as decimal separator: `17.45, 3114.765e99`
   - You may set up a vector (list of values), for example:

```
17.45, 33.9, #Var1, 99.17, @PI, 123.4e6
```

3 To check the validity of your input set the check box **Check contents for use in formula (numeric, vector)**. You will be diverted to *Check contents of variable or vector* on page 37.
4 Without this verification use button **Add/Update var/const** to update the list (after the syntax check).

Modify an existing variable

1 Select the user variable (or user constant) in the list of **Existing user var/const** by double clicking it.
2 The name is transferred into the edit field **Name v/c**. The contents (definition) of the user variable (or user constant) is transferred to the field **Contents**.
3 Modify the contents of the field.
4 To check the validity of your input set the check box **Check contents for use in formula (numeric, vector)**. You will be diverted to *Check contents of variable or vector* on page 37.
5 Use button **Add/Update var/const** to update the list (after the syntax check).

Delete a variable

1 Select the user variable (or user constant) in the list of **Existing user var/const** by double clicking it.
2 Use button **Delete var/const** to remove the variable from the list and the document.

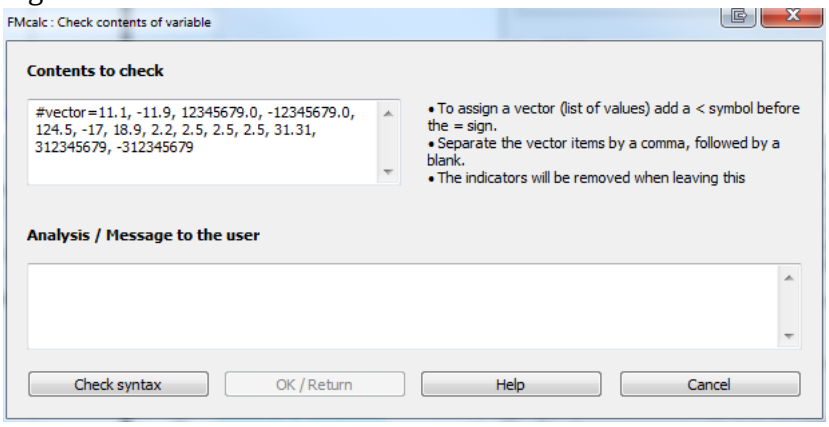
Note: *There is no Undo for this action.*

34 Of course such a comment will occur in the text if the variable is used outside **FMcalc**, that is as ordinary FM variable in text.

Check contents of variable or vector

This dialogue is entered if in **Manage user variables and constants** the respective check box is set.

Note: *An = sign is automatically inserted in front of the text to be verified[35]. This will be removed when going back to the calling dialogue.*



Button **Check Syntax** has the same function as in *Handle #calc markers* on page 13.

Assignment looks like a vector

This message is issued in two cases:

1 Either you really want to define a vector[36], then you are requested to enter a “<” in front of the = sign and **Check Syntax** again.

2 Or you intend something else. Then the statement must be modified.

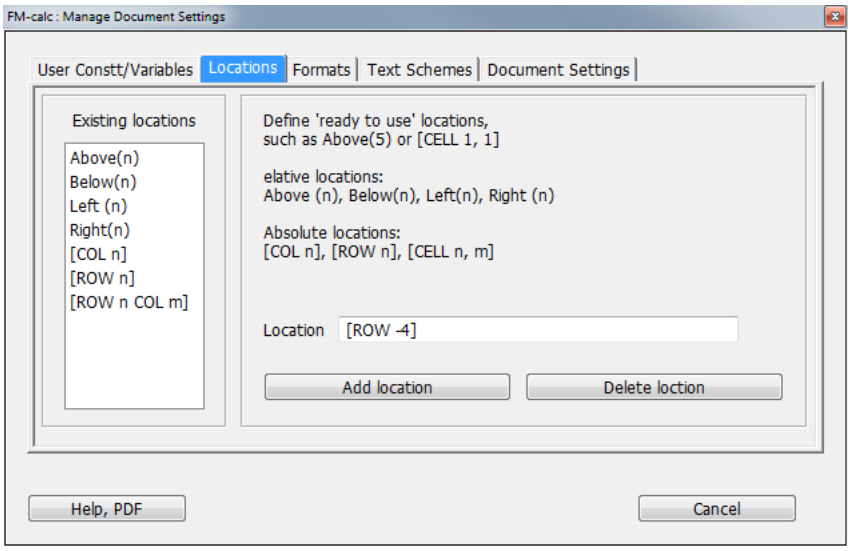
Or you simply want to provoke an error. Then use **Cancel** and force the storage in dialogue Document Settings with **New/ update var/const**.

After verification use button **OK/Return**.

35 Thus the functions to check the contents can be re-used from the marker-tests.

36 Since internal representation is different to ordinary values, this must be known.

Manage input locations



Note: *The 7 displayed location templates can not be deleted.*

Note: *If panel* **Handle #calc markers** *is open and a location is defined or deleted, then the corresponding list of locations in the panel is not updated.*

Define a new location

1 Type the definition in the edit field and use the **Add location** button.

2 Syntax is checked. However it can not be detected at this time whether (for example) a cell range is outside a table.

Modify an existing location

1 Select the item in the list by double clicking it. It will be transferred into the **Location** field.

2 Modify the definition in the edit field and use the **Add location** button.

3 If you do not want the original definition any more delete it from the list.

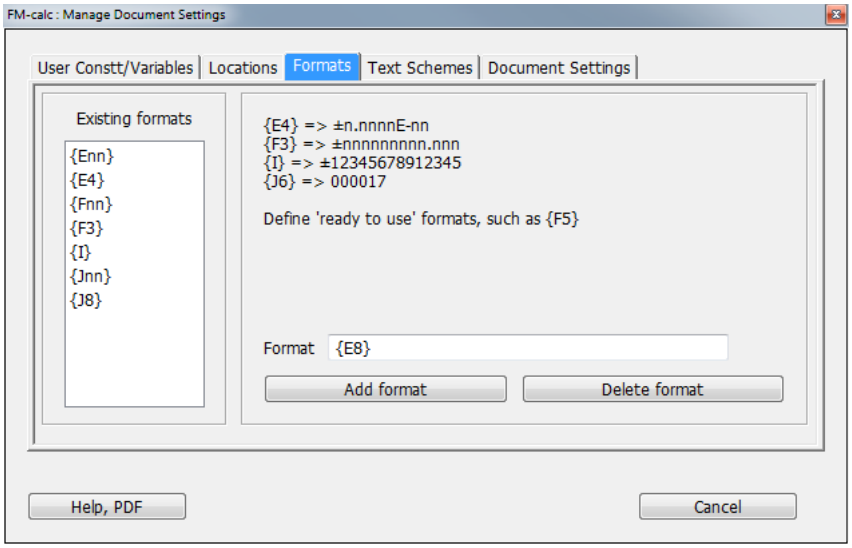
Delete a location

1 Select the item in the list by double clicking it.

2 Use button **Delete location** to remove the user item from the list.

Note: *There is no Undo for this action.*

Manage output formats



Note: *The 7 displayed format templates can not be deleted.*

Note: *If panel* **Handle #calc markers** *is open and an output-format is defined or deleted, then the corresponding list of formats in the panel is not updated.*

Significance indicator

{E*n*} n specifies the number of decimal places of the mantissa to be displayed. Displaying the value $-2.9979245765432\times10^{8}$ by various formats:
{E0} ➔ $-3.\times10^{8}$
{E8} ➔ -2.99792458×10^{8}

{F*n*} n specifies the number of decimal places. For example, displaying the constant `@PI` with various formats:
{F0} ➔ 3.
{F8} ➔ 3.14159265

{I} Integers are always presented with all figures.

{J*n*} Integer with leading zeroes. *n* specifies the number of figures including the leading zeros. If at output creation the number is to large to fit into *n* figures, the last figure is replaced by a question mark:
{J6}: ➔ 001234
{J3}: ➔ 12?

Define a new format

1 Type the definition in the edit field and use the **Add format** button.
2 Syntax is checked.

Modify an existing format

1 Select the item in the list by double clicking it. It will be transferred into the **Format** field.
2 Modify the definition in the edit field and use the **Add format** button.
3 If you do not want the original definition any more delete it from the list.

Delete a format

1 Select the item in the list by double clicking it.
2 Use button **Delete format** to remove the user item.

Note: *There is no Undo for this action.*

Manage text schemes

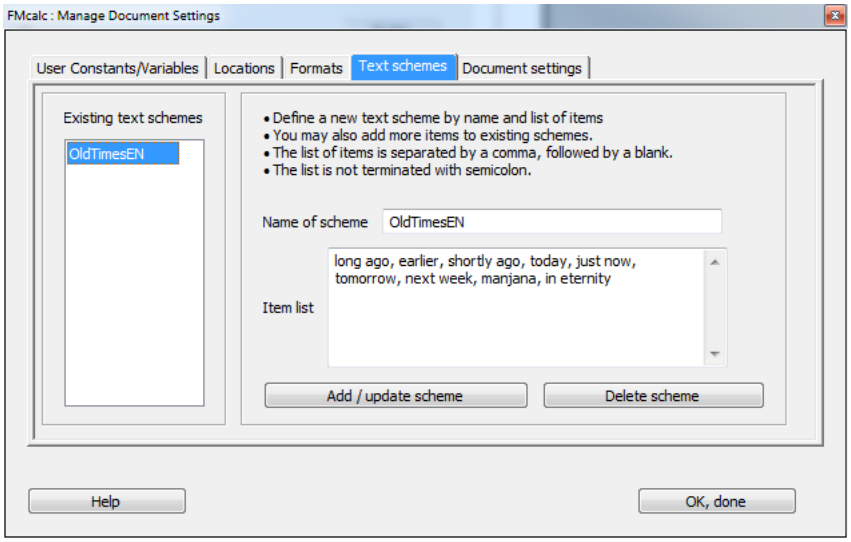
The definition of text schemes are kept on reference page `FMcalc` of the current document. Hence they are not ‘global’ to **FMcalc**.

Text schemes

Text schemes provide a list of items. For example, a series named `Latin` may be defined with these items:

```
Latin = semel, bis, ter, quater, quinquies, sexies,
septies, octies, nonies, decies, …
```

Note: *Comments are not possible. The list is not terminated by semicolon. With this example the fourth occurrence of the character format with name* `Latin` *will receive the string* `quater`.



Note: *If panel* **Handle #calc markers** *is open and a text scheme is defined or deleted, then the corresponding list of schemes in the panel is not updated.*

Define a new scheme

1 Type the desired name into the field **Name of scheme**. Observe rules for the names to be able to deduct the purpose from it.
   - Start with an upper case character.
   - Continue with upper case and lower case characters and/or figures.
   - The only special character allowed is the underscore (`Counter_something_12`).
2 Type the list of the series items into the field **Item list**:
   - Separate the items by a comma, followed by at least one blank.
   - Do not terminate the list with a semicolon or other punctuation.
3 Use button **Add / update scheme** to add the scheme to the list.

Note: *The order of the items in the definition is the order in which the items will be used. If the given list is exhausted, it will be repeated (*`nonies, decies, …, semel, bis`*).*

Modify an existing scheme

The standard schemes `Numeric`, `roman` and `ROMAN` can not be modified. They represent the ordinary number sequences `1 2`

`3 4 …, i ii iii iv …` and `I II III IV …` Hence they are not listed in **Existing text schemes**.

1 Select the scheme in the list of **Existing text schemes** by double clicking it.
2 The name is transferred into the edit field **Name of scheme**. The contents (definition) of the scheme is transferred to the field **Item list**.
3 Modify the contents of the field. For example, add items and/or reorder them manually.
4 Use button **Add /update scheme** to update the definition.

Delete a scheme

1 Select the scheme in the list of **Existing text schemes** by double clicking it.
2 Use button **Delete scheme** to remove the user item from the list.

Note: *There is no Undo for this action.*

Examples text schemes

Note: *Comments (text enclosed in straight quotes) is not possible here. Since these are not formula statements, the list is not terminated by a semicolon.*

CircledNumbers

```
①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧, ⑨, ⑩, ⑪, ⑫, ⑬, ⑭, ⑮, ⑯, ⑰, ⑱, ⑲, ⑳
```

To work correctly this scheme requires a character format (or the ¶-format) defining a Unicode font which includes the Block `Enclosed Alphanumerics` (code range `U+2460 … U+2473`), for example `Calibri` or `Consolas`.

EnumEN1

```
1st, 2nd, 3rd, 4th, 5th, …
```

Please note that the definitions can not bear a character format. Hence 1^{st}, 2^{nd} … are not possible.

EnumENverbal

```
first, second, third, fourth, fifth, sixth, seventh, …
```

GreekLC

```
α, β, γ, δ, ε, ζ, η, θ, ι, κ, λ, μ, ν, ξ, ο, π, ρ, σ, τ, υ, φ, χ, ψ, ω
```

Legal

```
semel, bis, ter, quater, quinquies, sexies, septies, octies, nonies, decies, …
```

MonthnamesA

```
Jänner, Feber, März, April, Mai, Juno, Juli, August, September, Oktober, November, Dezember;
```

OldTimesDE

```
in alten zeiten, füher, vor kurzem, heute, morgen, nächste woche, in zukunft, manjana, in der ewigkeit
```

Rainbow

```
red, orange, yellow, green, blue, indigo, violet
```

WeekdaysEN

```
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
```

hebrew_rainbow

```
Series = hebrew_rainbow; Scheme = קשת;
```

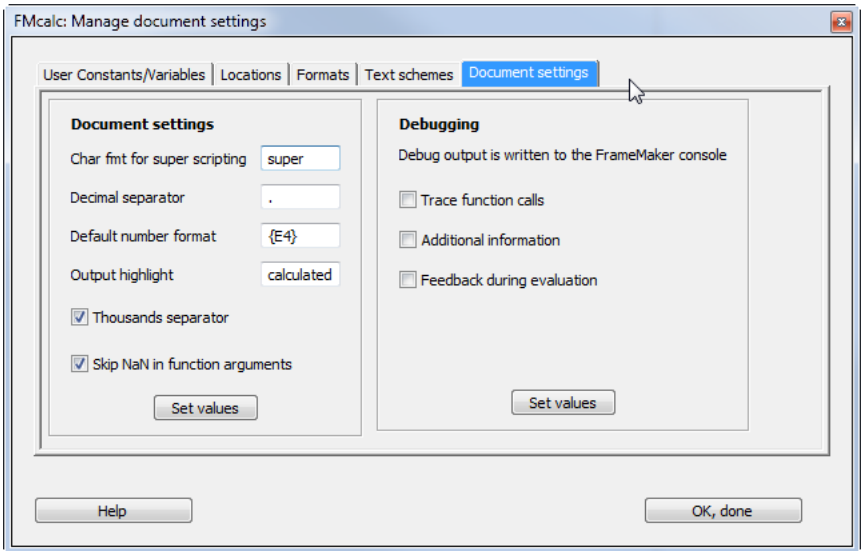
קשת = אדום, כתום, צהוב, ירוק, כחול, אינדיגו, סגול

Both on the reference page (above line) and for the character format you need a font capable of this language.

אדום Hebrew rainbow, 1st item (red)
כתום Hebrew rainbow, 2nd item (orange)
צהוב Hebrew rainbow, 3rd item (yellow)

Manage document settings and debugging

The dialogue presents the current values, which may be those copied over from the template (defaults). You adjust them to your needs:



Document settings

The following items influence the presentation of numeric data in output (contents of user variables or directly inserted into the text).

Char fmt for superscripting — Character format for superscripting. This format must exist in the character catalogue. If it does not exist, it will be created. Default is `super`.

Decimal separator — Character used to separate integer from decimal part. Default is *period*. Depends on the document[37]. This is only used during 'input' and 'output' (document text). Internally **FMcalc** always uses the period (text in markers).

Default number format — Default output format for real values. Default is `{E4}`.

See also *Significance indicator* on page 39.

Output highlight — Character format to highlight the evaluated output values. Default (`super`) is taken from the template. If it does not exist in the catalogue you are prompted for a setting.

Thousands separator — When checked, integers and floating point numbers are formatted according to typographic rules: If there are more than 4 figures left to the decimal point, groups of 3 figures are separated by a thin space.

Skip NaN in function arguments

```
#Input <= 11.1, 22.2, 33.3, invalid, 55.5, 66.6;
#Something = Sum (#Input);
```

NaNskip set (true, default)	NaNskip not set (false)
none-numeric arguments are skipped	none-numeric arguments terminate the list of values prematurely
`#Something` ➔ `188.7`	`#Something` ➔ `66.6`

"Debugging functions

You can switch on or off the following functions in dialogue **Format > FMcalc > Document Settings > Document Settings**.

37 Language and country custom for the document may influence this. Hence it is not deducted from the OS-locale.

If one of these is set, then information is written to the FM Console.

Trace function calls — An indented list of the call sequence of all functions will be created.

Note: *The following two settings may dramatically increase the amount of console output and hence slow down the process.*

Additional information — Also lowest level functions (in particular math functions) will be traced and few functions provide additional information.

Feedback during evaluation — Check this box to get information during evaluation of markers:

- Marker text as it is defined.
- Marker text evaluated: Constants, formulas etc. are resolved to numeric values.
- This allows to see a step-by-step evaluation (see *Example of analysis* on page 49).

Debugging

Nothing is perfect[38]. If the script fails (reports a script error in the FM console log file) try to locate the error:

- You may want to create a test document with just the marker creating problems.
- Programmers need to know what happened *before* the error. Hence be able to describe the last steps before the script issues the error.
- Try to reproduce the error in another document. If it does not occur there, what might be the difference of the two?
- The following are common sources for a failing script (not just **FMcalc**:
  - Interfering actions of the user: e.g. fiddling around with the reference page `FMcalc`, thus ruining its structure.
  - Manipulating the **#calc** or **#series** markers outside of the script dialogues.
  - Accidentally moving a script component (see *Script components* on page 46).
- If the problem is in evaluation and you can not get to glimpse what's the reason for that, you may switch on **Feedback in Evaluation** in the document settings.
- After running the script producing the error close FM and put the following items together in a ZIP archive):
  - File `%appdata%\Adobe\FrameMaker\vv\consfile.txt`
  - The explanation, what You did before the error occurred.
  - If possible, the document producing the error. Hence it is good practice to create a narrowed down document which still creates the error.
- Please be aware that the author is not that young anymore and you may need to seek help in the FM/ExtendScript community. That's also a reason for distributing the full source code ...

38 The script comprises nearly 200 functions with about 4500 lines of code.

Script installation

1 Close FrameMaker if necessary.

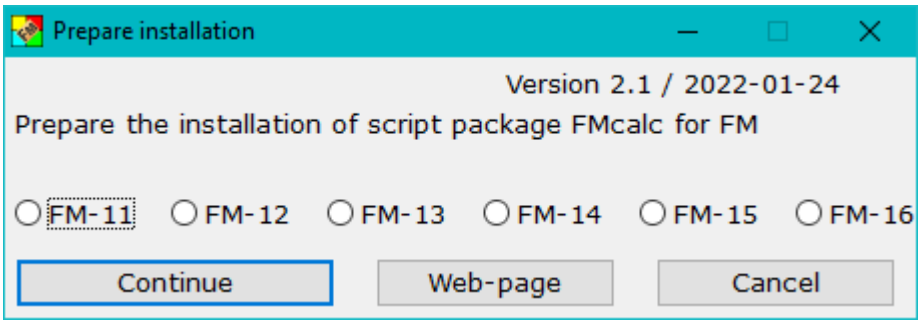
2 Download the Inst-**FMcalc**.zip from my web-page

3 UnZip the file to the desktop.

4 Execute the file `PrepareInstallation.exe` as Admin.

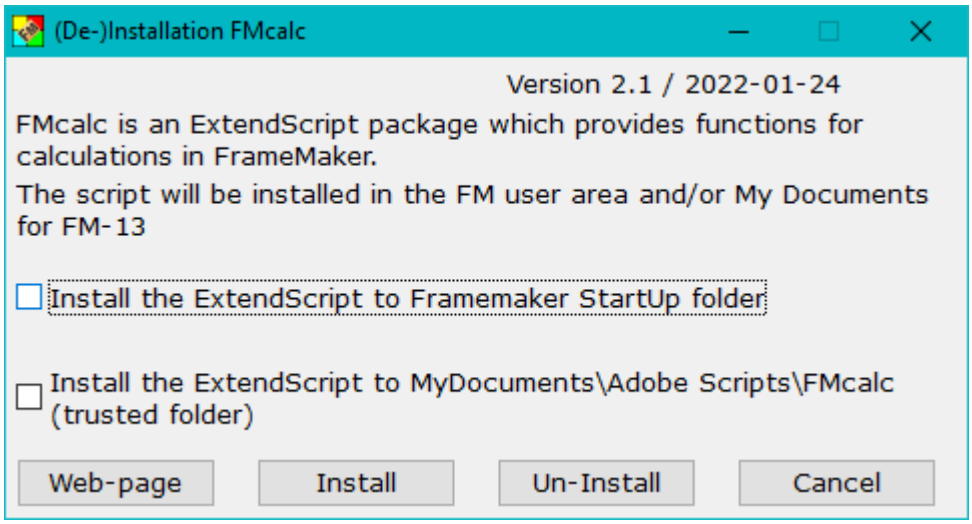
Note: *Windows Defender may react to this program with a virus warning. This is a false positive. You may check this with an upload to Virustotal. See Avoid Windows Defender intervention on page 45.*

5 The last six FM-versions present in your `%appdata%/Adobe/FrameMaker/` are listed in the dialogue (on your system there may be only one).



The language of the installation dialogues adapts to the UI language of your current FM installation (de, en, fr).

6 Select the appropriate FM installation and use **Continue** . This opens the installation program:



7 Select the appropriate options:

- Use the first option to start the script automatically at FM-start.
- Use the second option if you wish to start the script on demand from a trusted source. You can define the script in **File > Scripts ... > Catalog** (Script Library) as Favourite: use the Add button and navigate to `My Documents \Adobe Scripts\`**FMcalc**`\`**FMcalc**`.jsxbin`.

8 With **Install** the script and associated files will be transferred into the relevant locations.

9 A message indicates the end of the installation. The desktop icons will be removed.

De-install the script

To de-install the script, start the installation program from the short cut in the **Start-menu > D+DD**. Then use the **Un-Install** button.

This removes the script only from the locations defined in the options. You can later use the installation program again.

To get completely rid of the script you need to remove the following in addition to the de-installation:

- The short cut in the start menu
- Directory `%appdata%\D+DD\`**FMcalc**

Avoid Windows Defender intervention

1. In Windows settings navigate to **Windows Security > Virus & Threat protection settings > Manage Settings**
2. Go to **Exclusions**
3. Add the folder you need to exclude:
   `C:\Users\`*username*`\AppData\Roaming\D+DD`

Details of the D+DD script installation

- Unpacking creates a program (`Prepinstallation.exe`) and a directory (`InstallThis`).
- `Prepinstallation.exe` should be executed with admin rights (right-click and select), otherwise a message will appear indicating this.
- `Prepinstallation` adds the information about the current directory (e.g. `C:\Users\Klaus\Desktop\InstallThis`) and the installation directory of the selected FM version to the `InstallThis\config.ini` file.
- Then `InstallThis\Inst-`**FMcalc**`.exe` is executed, which creates a directory `D+DD\`**FMcalc** in the Windows Start directory and copies everything from `InstallThis` there. In addition, a short cut is created to this directory.
- This directory contains `Inst-`**FMcalc**`.exe`, which is now executed and the unpacked files that are no longer used (e.g. on the desktop) are deleted.
- `Inst-`**FMcalc**`.exe` asks where the script should be copied to (`Startup` folder or `MyDocuments\Adobe Scripts\`**FMcalc**). The selected option must also be selected during a de-installation in order to de-install at the correct location.
- The script consists of the main module **FMcalc**`.jsx` and the modules in the **FMcalc** directory.
- Various scripts (e.g. the installation of the tool bars) also write something in the `maker.ini` in the user area in the section `[D+DD]`.

Implementation details

Known issues

Panel and console intertwined

If a panel is invoked and this opens *together* with the console:

- Closing the console window also closes the panel and the next invocation of the panel creates “Unable to create Window” ➔ Restart FM at least twice before next start.
- As long you do not close the console window things work.
- The reason for this “slave effect” is not known.

Script components

All script components must reside in the same directory.

`FMcalc.jsx`	The main script
`FMcalc.pdf`	This document
`FMcalc_Ancillary.jsx`	Ancillary functions
`FMcalc_dialogC.jsx`	Palette for handling #calc markers
`FMcalc_dialogDS.jsx`	Palette for maintaining building blocks and document settings
`FMcalc_dialogS.jsx`	Palette for handling #series markers
`FMcalc_dialogV.jsx`	Dialogue for verifying variable contents
`FMcalc_Evaluation.jsx`	Functions related to the evaluation of markers
`FMcalc_Globals.jsx`	Definition of global variables, objects and arrays.
`FMcalc_Markers.jsx`	Functions for handling markers.
`FMcalc_Math.jsx`	Mathematical functions.
`FMcalc_OpenPdf.exe`	Open a PDF at a specified location. This is used in `FMcalc_tScript.bat`
`FMcalc_tpl_en.fm`	Templates for the reference page.
`FMcalc_tpl_de.fm`	(In FM-10 to allow the script to run
`FMcalc_tpl_fr.fm`	in a wide range of FM versions).
`FMcalc_tScript.bat`	Generated batch file to open the Help pdf (this document).
`FMcalc_en.xml`	Dialogue texts in English (default)
`FMcalc_de.xml`	Dialogue texts in German
`FMcalc_fr.xml`	Dialogue texts in French

FM Startup folder

To start the script with the start of FM, the following files must be copied to the **Startup** folder (*vv* = 10 ... 14):
`%appdata%\Adobe\FrameMaker\vv\Startup\`

`FMcalc.jsx`	The main component.
`FMcalc`	Directory containing all other components.

User interface

Dialogues

The dialogues are palettes to stay open while the user can change focus to the document. They were designed using [12] and then developed further in the program editor.

UI language support

All texts suitable for translation are externalised in XML modules `FMcalc_ll.xml`. Some texts contains place holders `%01` … `%09`. These messages are completed by function `BuildMsg`:

Definition

```
<EvaluateSTextScheme msg_01= "Series «%01»: Not enough items «%02» for Start value = «%03»." />
```

Call

```
sMsg1 = KLD_C.BuildMsg (KLD_C.UItxt.EvaluateSTextScheme.@msg_01, oSeries.Name, nTextItems, oSeries.Start);
```

Message

```
Series «Gugus»: Not enough items «5» for Start value = «7»
```

FMcalc automatically takes the texts from the file corresponding with the FM UI language (en, de, fr).

Note: *The French translation done with the help of Deepl for Windows (https://www.deepl.com/app).*

Script internals

Script internals, such as the items in a formula or series statement are *not translated*. Translation would disable the exchange of documents between FM-language-versions (FM users know this problem from the TOC / IVZ / TDM etc. suffixes of generated files).

See also *Translation issues for documents using FMcalc* on page 54.

Handling defaults

Defaults are kept on the reference page `FMcalc` of the current document. If this reference page does not yet exist it is copied from the template file `FMcalc_tpl_ll.fm` (`ll`: de, en, fr).
See *Manage document settings and debugging* on page 42.

Adapting the template

Your company standards may require to adapt certain elements in the reference page of the template. The following table gives you an example:

Standard template ref page item	Swiss-German environment

`FmtSup = super`
`FmtDecimal =.`
`FmtReal = {E4}`
`FmtHigh = calculated`
`FmtThousand = true`
`NaNskip = true`
	`FmtSup = exponent`
	`FmtDecimal =,`
	`FmtReal = {F2}`
	`FmtHigh = berechnet`
	`FmtThousand = true`
	`NaNskip = true`

When modifying the template be careful not to ruin the structure of paragraph formats! They are not in the catalogue.

Program relevant values

The following are hard coded, can not be changed by the user:

Character	Purpose
`. (period)`	Decimal separator in numeric values (within **FMcalc**).[a]
`,␣ (comma, followed by blank)`	Separates list items (e.g. in text schemes)
`; (semicolon)`	Terminates statements. Hence multiple statements may exist in a **#calc** marker.

a. Input from table cells or data file are checked with regular expressions which allow both period or comma

The following constants are not exposed to the user. They are replacements for JavaScript constants not present in ExtendScript. They may be used in functions which substitute methods not present in ES.

Keyword_0	Explanation	Value approx	JS
`@EPSILON`	[a]	$2.220446049 \times 10^{-16}$	6
`@MAX_SAFE_INTEGER`	$2^{53} - 1$ [b]	9007199254740991	6
`@MIN_SAFE_INTEGER`	$-(2^{53} - 1)$	-9007199254740...	

a. JavaScript: Number.EPSILON represents the difference between one and the smallest value greater than one that can be represented as a Number.

b. Used in function `IsSafeInteger` (not exposed to the user).

#calc markers

Markers defining ordinary variables may all be located at the beginning of the document. This may ease maintenance of the document.

Markers defining a temporary result must of course be at the proper location!

Evaluating #calc markers

Not all details are given here.

1 Check syntax again, because the marker may have been modified outside **FMcalc**. In case of an error, present the marker dialogue again.

2 Remove comments and separate statements at the semicolon. The syntax checking assures already balanced parentheses[39].

3 Scan each statement from left to right. Special characters trigger actions:

- `=` Memorise location. If nothing is left to the symbol, create a temporary variable.
- `@` Replace name of constant by its contents and skip this.
- `#` If this is behind the location of the = sign: replace name of variable by its contents and skip this.
- `A...Z` Check whether a function name starts. If so, replace it by the name of the internal function name (`Abs` ➔ `Math.abs; Cbrt` ➔ `M_Cbrt`) and skip the result.
  Cell ranges (e.g. `Left(n)`) are evaluated (input read) ... and skip the result
  Invalid starting chars (e.g. `B`) are already recognised by the syntax check.
- `[` Find the contents of the cell addressing and put it into a scalar or vector. Replace the construct by the value list and skip.
- `,` Comma must be followed by at least one blank to indicate that a next numeric value follows to form a vector.
- `{` Remove the format information and put it aside.
- `...` The following are just skipped, because they form valid elements: operators (-+*/) and numeric values.

4 Use `eval` to evaluate the formula string. This may return errors in which case the marker dialogue is presented.

5 Format result accordingly.

6 Place result in variable.

Note: *Corrective actions are left to the user (see for example Check syntax on page 14.*

Example of analysis

The following lines show the steps:

```
"F01" #vat_amount = Round(((Sum(Left(3))) + #item + [CELL 5, 5]) * #VAT, 2) {F2};
#vat_amount = Round(((Sum(Left(3))) + #item + [CELL 5, 5]) * #VAT, 2)
#vat_amount = M_Round(((Sum(Left(3))) + #item + [CELL 5, 5]) * #VAT, 2);
#vat_amount = M_Round(((M_Sum(Left(3))) + #item + [CELL 5, 5]) * #VAT, 2)
#vat_amount = M_Round(((M_Sum(17.0, 23.7, 37.9)) + #item + [CELL 5, 5]) * #VAT, 2)
```

39 However, syntax errors such as ...(...; ...)... can not be detected with this procedure.

```
#vat_amount = M_Round(((M_Sum(17.0, 23.7, 37.9)) + 12.34 + [CELL 5, 5]) * #VAT, 2)
#vat_amount = M_Round(((M_Sum(17.0, 23.7, 37.9)) + 12.34 + 56.78) * #VAT, 2)
#vat_amount = M_Round(((M_Sum(17.0, 23.7, 37.9)) + 12.34 + 56.78) * 0.08, 2)
```

Interpreting input

'Within' **FMcalc** values are kept in simple form, such as `-17.5` or `-17.456e-009`. However, within a variable to be placed in text (including temporary variables, a sophisticated notation is used:

- The minus sign is replaced by "non breaking hyphen -" both for the sign of the value the sign of an exponent.
- The exponent symbol (`e` or `E`) is presented as `×10<super>`
- Superfluous zeros in the exponent are removed.

The only exception for this are vectors. These use unformatted values. It is assumed that such variables are not placed in the text, but only used in formulas.

Hence during input from variables and cell contents this nice formatting must be removed for further processing.

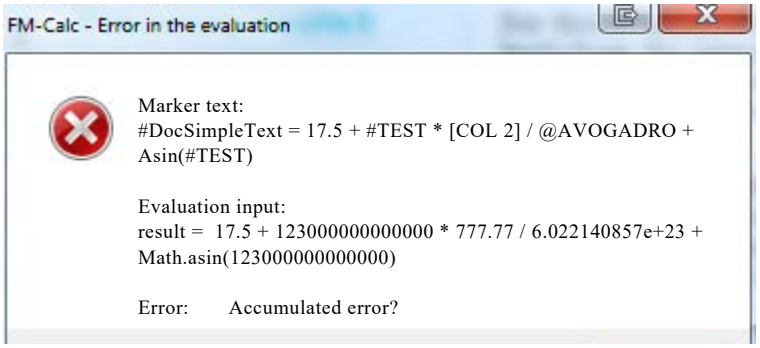
Evaluation

The extended formula will consist only of numeric values, operators and function calls.

This is evaluated with the JS function `eval()`, which may report errors. In this case the user gets the marker dialogue to check the marker text.

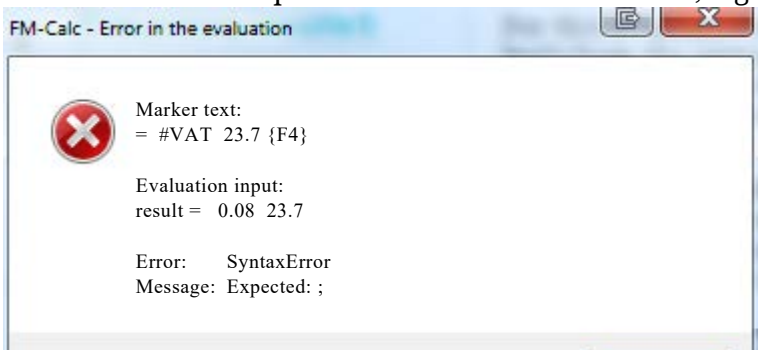
Error message from evaluation

If either or both **DialogC** or **DialogS** are open then these error messages are displayed in the area **Marker analysis**. If no panel is open these messages appear in an alert box:



Of course it needs fantasy to see the problem: The **Asin** function is defined only for the input range -1 to +1. Hence the huge value presented via `#TEST` is invalid.

Other errors are reported from the **eval** function, e.g.



Syntax Error

The JavaScript interpreter often gives a false hint about the error. In the example above it expects a semicolon to terminate the statement after `0.08`. But you have just forgotten an operator between the two values. Also the following is a false hint:

Eval input: `result = Math.atan2(23.7 17.5)`
Message: `Expected: )`
What's really missing is the comma between the arguments, not a parenthesis after the first argument.

ReferenceError — More than one = sign. Undefined variable encountered. Watch for `NaN`, `undefined`, `null` in the *Evaluation input*.

RangeError — Function argument is not in the allowed value range.

TypeError — Function argument is not of proper type. This is an error of the program author. Report it!

Temporary results

For this a variable with a generated name is used. The name is built from the characters `#zz`, followed by the ISO 8601 representation of the current time, for example:

```
#zz2016-09-27T10:28:46.060Z
```

Before such a variable is placed behind the marker, the old one (if present) is removed, because otherwise the new one would just precede the old one. Hence these variables must not be copied or moved.[40)]

Allowed names for variables

Valid variable names are checked with the following regular expression in `FMcalc_dialogDS.jsx:ButtonAddVarConst()`, `FMcalc_Markers.jsx:CheckTermElementsC()` and in `FMcalc_Ancillary.jsx:ContainsVector()`

```
#[A-Za-z\u00C0-\u017F][\d_A-Za-z\u00C0-\u017F]* *$
```

This comprises Unicode groups Basic Latin (aka ASCII), Latin-1 Supplement and Latin Extended-A.

To allow, for example, Cyrillic names, these definitions must be extended:

```
#[A-Za-z\u00C0-\u017F\u0400-\u04FF][\d_A-Za-z\u00C0-\u017F\u0400-\u04FF]*
```

Numeric values

Numeric values are checked with the following regular expression in `FMcalc_Ancillary.jsx:ContainsNumber` and `ContainsVector`.

```
^[-+]?[0-9]+\.?([0-9]+)?([eE][-+]?[0-9]+)?
```

Examples and their evaluation with `eval()`:

```
1e9                    true   1000000000
1e9 and 17,5           true   1000000000
and 17,5               false  not starting with a figure
-and 17,5              false  no figure after sign
e-9                    false  not starting with a figure
-123.4G-17             true   -123.4 Note the truncation!
-11.3                  true   -11.3
-123.4e-17             true   -1.234e-15
12345678901234567890   true   12345678901234567000
1234567890.1234567890  true   1234567890.12346
-123.4E-17             true   -1.234e-15
-123.4e-17             true   -1.234e-15
-123.e-17              true   -1.234e-15
```

Formatting results

For the following table Decimal separator was set to comma and the Thousands separator was not set.

40 No more used temporary variables after (Re)-evaluating are removed from the variable catalogue.

Input	I	J4	F3	F0	E3	E7
0	0	0000	0,000	0	0,000×10<sup>+0	0,0000000×10<sup>+0
1	1	0001	1,000	1	1,000×10<sup>+0	1,0000000×10<sup>+0
-10	-10	-10	-10,000	-10	-1,000×10<sup>+1	-1,0000000×10<sup>+1
0.0000123	0	0000	0,000	0	1,230×10<sup>-5	1,2300000×10<sup>-5
-3.14159265358979	-3	-3	-3,142	-3	-3,142×10<sup>+0	-3,1415927×10<sup>+0
1723456700	1723456700	1723456700	1723456700,000	1723456700	1,723×10<sup>+9	1,7234567×10<sup>+9
apfel	NaN	0NaN	NaN	NaN	NaN	NaN
33.7	33	0033	33,700	34	3,370×10<sup>+1	3,3700000×10<sup>+1
12345600000	12345600000	12345600000	12345600000,000	12345600000	1,235×10<sup>+10	1,2345600×10<sup>+10
1.23456e-50	0	0000	0,000	0	1,235×10<sup>-50	1,2345600×10<sup>-50
1.7976931348623157e+308	1.7976931348623157e+308	1.7976931348623157e+308	1,7976931348623157e+308	1,7976931348623157e+308	1,798×10<sup>+308	1,7976931×10<sup>+308
2.2250738585072e-308	0	0000	0,000	0	2,225×10<sup>-308	2,2250739×10<sup>-308
1.8e+308	Infinity	Infinity	Infinity	Infinity	Infinity	Infinity

#series markers

Implementation

A **#series** marker defines the series. A character format marks the text which will be ‘filled’ by the series items.

(Re)-start series

A special character (¬ optional hyphen) is present in the first text with the indicating character format. This can easily be inserted with a keyboard short cut (**CTRL+minus**).

Evaluating #series markers

1 The first **#serie**s maker is searched in the document and evaluated.

Since also these markers may have been modified with the standard UI they are checked again (In a **#series** marker statements are restricted to use constants, variables, but not functions and operators). Also the existence of the corresponding character format is checked.

2 Looping through the document (in parallel to the series items) the locations of the used character format are found and replaced by the respective series items.

3 If in the found text the special “first item” character (¬ optional hyphen) is found, then the series is restarted.. The special symbol is re-inserted at begin of the output.

4 Of course the end of the document terminates the insertion loop.

5 Then the next defining marker is worked off.

Consequences

The defining marker must be located before the use of series. IMHO this is not a restriction.

Defining **#series** markers may all be located at the beginning of the document to ease maintenance of the document.

Allowed names for series

For demonstration purpose the pattern for scheme names has been extended to allow also Hebrew characters:

```
^[A-Za-z\u0590-\u05FF][_\w\u0590-\u05FF]* *$
```

See also *Translation issues for documents using FMcalc* on page 54.

Translation issues for documents using FMcalc

There are already translation issues with standard markers of type index, glossary, glossary term etc. Hence translators already know to carefully follow the advice of the customer.

Both the translator and the receiving customer may not have the script **FMcalc** available. Hence only the standard UI of FM can be used for the document.

#calc markers

There is not much to translate in these. Only the comments may be considered.

In rare cases it may be useful to translate the names of variables:

- Existing names must be exchanged in the document catalogue of variables.
- Names must be exchanged in the formula statements. A certain name may exist in various markers.
- Names must be exchanged on the reference page `FMcalc` in section `[Ref-Variables]` (not the surrounding text in magenta!).
- An example may be the user constant `#VAT` (Value added tax) which in a German document may be `#MwSt` (*MehrwertSteuer*).

#series markers

The items of a series are mostly text, thus targets for translation. Also the name of a series may be subject to translation. The following must be observed:

- Whether the name of the series (and the corresponding character format) is to be changed, must be decided. It exists both in a marker and on the reference page.
- The entries in the reference page `FMcalc` in section `[Ref-Schemes]` must be adapted. The order of items in a line must not be changed.
- If the receiving customer can not use **FMcalc** to re-evaluate the series' in the text, than these must be translated. The character format (and the special character in the first item of a series[41]) must not be destroyed.

RTL languages

Starting with FM-13 (aka Edition 2015) RTL languages such as Hebrew are supported.

In **FMcalc** it is essential not to change the format name of the paragraph (`Ref-schemes`). It may however use any capable font (here it is `Lucida Sans Unicode`):

```
Rainbow = red, orange, yellow, green, blue, violet
```

This is not a sentence, but a sequence of words which are worked off always starting after the = sign. There is no terminating period. The items must be separated by ordinary commas, followed by a blank. The translation into an RTL language will take the whole paragraph into account:

קשת = אדום, כתום, צהוב, ירוק, כחול, אינדיגו, סגול

41 This character (`\x04` - optional hyphen) can be anywhere within the scope of the character format.

Math functions in ExtendScript

Since ExtendScript is based on ECMA Script 1999, the set of mathematical constants and functions is limited.

Constants

```
Math.E  2.71828182845905
Math.LN102.30258509299405
Math.LN20.69314718055995
Math.LOG10E0.43429448190325
Math.LOG2E1.44269504088869
Math.PI3.14159265358979
Math.SQRT1_20.70710678118655
Math.SQRT21.4142135623731
```

Functions

```
x= 1.0471975511966
y= 0.62831853071796
Math.acos(x) NaN
Math.acos(y) 0.89140640004395
Math.asin(x) NaN
Math.asin(y) 0.67938992675095
Math.atan(x) 0.80844879263002
Math.atan2(x,y) 1.03037682652431
Math.cos(x) 0.5
Math.sin(x) 0.86602540378444
Math.tan(x) 1.73205080756888
Math.ceil(x) 2
Math.floor(x) 1
Math.max(x,y) [42]1.0471975511966
Math.min(x,y) 0.62831853071796
Math.exp(x) 2.84965390822636
Math.log(x) 0.04611759718129
Math.pow(x,y) 1.02940044538511
Math.round(x+y) 2
Math.sqrt(x) 1.02332670794649
Math.random()0.59190368652344
```

Random numbers (from a loop)

```
  0.00221252441406
  0.33302307128906
  0.33287048339844
  0.48353576660156
```

Additional math functions defined in JavaScript

Many math functions defined in ECMA Script 2011 or even ECMA 2015 Script (which are implemented in JS now) are not available in ES and must be emulated by a PolyFill. See [3].

Acosh (x)

```
Math.acosh(-1); // NaN
Math.acosh(0);  // NaN
Math.acosh(0.5) // NaN
Math.acosh(1);  // 0
Math.acosh(2);  // 1.3169578969248166
```

Function can be emulated by

```
return Math.log(x + Math.sqrt(x * x - 1));
```

Asinh (x)

```
Math.asinh(1);  // 0.881373587019543
Math.asinh(0);  // 0
```

Function can be emulated by

42 `Math.max` and `Math.min` can have only 2 arguments. In JS 2011 and later the number of arguments is nearly unlimited.

```
if (x === -Infinity) { return x; }
else { return Math.log(x + Math.sqrt(x * x + 1)); }
```

Atanh (x)

```
Math.atanh(-2);  // NaN
Math.atanh(-1);  // -Infinity
Math.atanh(0);   // 0
Math.atanh(0.5); // 0.5493061443340548
Math.atanh(1);   // Infinity
Math.atanh(2);   // NaN
```

Function can be emulated by

```
return Math.log((1+x)/(1-x)) / 2;
```

Cbrt(x)

Function returns the cubic root.

```
Math.cbrt(-1); // -1
Math.cbrt(0);  // 0
Math.cbrt(1);  // 1
Math.cbrt(2);  // 1.2599210498948734
```

Expm1(x)

Function returns e^x - 1

Note: *This function is not used in* **FMcalc**.

```
Math.expm1(-1); // -0.6321205588285577
Math.expm1(0);  // 0
Math.expm1(1);  // 1.718281828459045
```

Hypot(v1,v2,...,vn)

Function returns the square root of the sum of squares of its argument:

```
Math.hypot(3, 4);         // 5
Math.hypot(3, 4, 5);      // 7.0710678118654755
Math.hypot();             // 0
Math.hypot(NaN);          // NaN
Math.hypot(3, 4, 'foo'); // NaN, +'foo' => NaN
Math.hypot(3, 4, '5');   // 7.071067811865476, +'5' => 5
Math.hypot(-3);           // 3, the same as Math.abs(-3)
```

Log10 (x)

Function returns the base 10 logarithm of a number, that is ∀x > 0, Math.log10(x) = $\log_{10}$(x) = the unique y such that 10^y=x.

```
Math.log10(2);      // 0.3010299956639812
Math.log10(1);      // 0
Math.log10(0);      // -Infinity
Math.log10(-2);     // NaN
Math.log10(100000); // 5
```

Function can be emulated by:

```
return Math.log(x) / Math.LN10;
```

Log1p (x)

Function returns the natural logarithm of 1 + a number, that is ∀x > -1 , Math.log1p(x) = ln(1+x)

Note: *This function is not used in* **FMcalc**.

```
Math.log1p(1);  // 0.6931471805599453
Math.log1p(0);  // 0
Math.log1p(-1); // -Infinity
Math.log1p(-2); // NaN
```

Function can be emulated by:

```
return Math.log(1 + x);
```

Log2 (x)

Function returns the base 2 logarithm of a number, that is

∀x > 0, Math.log2(x) = log2(x) = the unique y such that 2^y=x

Note: *This function is not used in* **FMcalc**.

```
Math.log2(3);    // 1.584962500721156
Math.log2(2);    // 1
Math.log2(1);    // 0
Math.log2(0);    // -Infinity
Math.log2(-2);   // NaN
Math.log2(1024); // 10
```

MeanG (v1,v2,...,vn)

The calculation of the Geometric Mean may appear impossible if one or more of the data points is zero (0). In these cases, however, the convention used is that a value of either '1', one half the limit of detection, or some other substitution is allowed for each zero or "less than" value, so that the information contained in these data is not lost.

For example, the US Food and Drug Administration in its shellfish sanitation program regulations requires the substitution of a value that is one significant digit less than the detection limit [i.e. "less than 2" becomes "1.9"]. Because of how geometric mean is calculated, the precise substitution value generally does not appreciably affect the result of the calculation, and ensures that all the data remains usable.

[https://www.waterboards.ca.gov/water_issues/programs/swamp/docs/cwt/guidance/3413.pdf]

➔ In FMcalc any value ≤ 0 is replaced by 1 during the calculation of the geometric mean.

MeanH (v1,v2,...,vn)

Since the reciprocal of 0 is undetermined, zero values are replaced by 1 during the calculation of the harmonic mean.

Pow (b, x)

Function returns the base to the exponent power, that is, b^x.

```
Math.pow(7, 2);    // 49
Math.pow(7, 3);    // 343
Math.pow(2, 10);   // 1024
// fractional exponents
Math.pow(4, 0.5);  // 2 (square root of 4)
Math.pow(8, 1/3);  // 2 (cube root of 8)
Math.pow(2, 0.5);  // 1.4142135623730951 (square root 2)
Math.pow(2, 1/3);  // 1.2599210498948732 (cube root 2)
// signed exponents
Math.pow(7, -2);   // 0.02040816326530612 (1/49)
Math.pow(8, -1/3); // 0.5
// signed bases
Math.pow(-7, 2);   // 49 (squares are positive)
Math.pow(-7, 3);   // -343 (cubes can be negative)
Math.pow(-7, 0.5); // NaN (negative numbers don't have a
real square root)
```

Note: *Due to "even" and "odd" roots laying close to each other, and limits in the floating number precision, negative bases with fractional exponents always return NaN*

```
Math.pow(-7, 1/3); // NaN
```

Round (x, n)

Rounding to the n'th decimal. See [11]

```
Round (55.55, -1);        // 55.6
Round (55.549, -1);       // 55.5
Round (55, 1);            // 60
Round (54.9, 1);          // 50
```

```
Round (-55.55, -1);       // -55.5
Round (-55.551, -1);      // -55.6
Round (-55, 1);           // -50
Round (-55.1, 1);         // -60
Round (1.005, -2);        // 1.01
```

Sign (x)

Function returns the sign of a number, indicating whether the number is positive, negative or zero.

```
Math.sign(3);     //  1Math.sign(-3);    // -1
Math.sign('-3');  // -1
Math.sign(0);     //  0
Math.sign(-0);    // -0 displayed as 0 by ES
Math.sign('foo'); // NaN
Math.sign();      // NaN
```

Function can be emulated by

```
{ x = +x; // convert to a number
  if (x === 0 || isNaN(x)) {
    return x;
  }
  return x > 0 ? 1 : -1;
}
```

Trunc (x)

Function returns the integral part of a number by removing any fractional digits.

```
Math.trunc(13.37);    // 13
Math.trunc(42.84);    // 42
Math.trunc(0.123);    //  0
Math.trunc(-0.123);   // -0
Math.trunc('-1.123'); // -1
Math.trunc(NaN);      // NaN
Math.trunc('foo');    // NaN
Math.trunc();         // NaN
```

Function can be emulated by

```
return x < 0 ? Math.ceil(x) : Math.floor(x);
```

Additional math functions for FMcalc

I wanted to have functions which are not available in any current JS implementation. Especially function for vectors (lists of variables) are completely missing in JavaScript.

Note: *Functions with a vector input (list of values) must copy the arguments to a local array* ***if the list of values must be passed to a subsidiary vector function****. See* `M_StdDev`.

Area (x_0, y_0, ...)

Area below the polygon to the x-axis. If the polygon is closed, then the area is the inner of the polygon. See also *Area* on page 19.

Count (x_0, x_1, ...)

Number of the arguments. This helper function allows a variable length argument list for the vector functions.

Factorial (n)

The factorial is calculated in a simple loop. For integer output n must be ≤ 19. To avoid overflow (with result Infinity) n must be ≤ 170.

Fract (x)

Decimal part of the value.

```
return x - Math.trunc(x)
```

GetData (file)

This is not a math function. It reads data into a user variable.

Max (x_0, x_1, ...)

Get the maximum of the argument values. Programmed with loop through arbitrary number of arguments.

MeanA (x0, x1, ...)

Arithmetic mean of a value list

MeanG (x0, x1, ...)

Geometric mean of a value list

MeanH (x0, x1, ...)

Harmonic mean of a value list

Median (x0, x1, ...)

https://www.statcan.gc.ca/edu/power-pouvoir/ch11/median-mediane/5214872-eng.htm

Min (x_0, x1, ...)

Get the minimum of the argument values. Programmed with loop through arbitrary number of arguments.

Modulo (y, x)

Emulated by

```
result = Math.abs(y) % Math.abs(x);
  if (x > 0) {return result);
  } else {
  return -result;
  }
```

Note: *The result of modulo operator takes the sign of the divisor, not the dividend (as it is with Remainder).*

QSimps (x_0, y_0, ...)

Quasi Simpson method for even and uneven intervals. This function is based on a Fortran program by Bruce Cameron Reed, Physics, Alma College, Michigan <reed@alma.edu>. He courteously sent it 2016 to me. See *QSimps* on page 21.

Remainder (y, x)

Emulated by

```
return y % x; "this is what JS does with % "
```

Sum (x_0, x_1, ...)

Sum of the arguments. Programmed with loop through arbitrary number of arguments.

SumProd (x_0, x_1, ...)

Programmed with loop through arbitrary *even* number of arguments.

Accuracy

The reference for comparing the accuracy of calculations is the Windows 7 built-in calculator.

ExtendScript and JavaScript operate with double precision values (see *Numeric range* on page 61) while the reference program Kalkulator works with Extended precision providing nearly double number of figures and exponent range.

FMcalc function	Result from ExtendScript[a]	Result from reference [b]
12345679 * 999999999	$1.234567898765430\times10^{+16}$	12345678987654321
12345679 * 99999999999	$1.234567899987650\times10^{+18}$	1234567899987654**321**
Acos (π/5)	$8.914064000439470\times10^{-1}$	0.8914064000439457**1** (K)
Asin (π/5)	$6.793899267509500\times10^{-1}$	0.679389926750950**91** (K)
Atan (π/3)	$8.084487926300220\times10^{-1}$	0.808448792630022**03** (K)
Atan2 (π/3, π/5)	1.030376826524310	1.0303768265243124**6** (K)
Cos (π/5)	$8.090169943749480\times10^{-1}$	0.809016994374947**42** (K)
Sin (π/5)	$5.877852522924730\times10^{-1}$	0.587785252292473**13** (K)
Tan (π/5)	$7.265425280053600\times10^{-1}$	0.726542528005360**89** (K)
MeanA (values) [c] using $\sum x$	$1.190071428673610\times10^{+1}$	1.19007142857142**9**e+1
MeanA (values) [c)] using $\sum \frac{x}{n}$	$1.190071428567170\times10^{+1}$	1.19007142857142**9**e+1
Cbrt (2)	1.259921049894870	1.259921049894873164767**2**...
Exp (π/3)	2.849653908226360	
Hypot (3, 4, 5)	7.071067811865480	7.07106781186547524400**84**...
Log (π/3)	$4.611759718128950\times10^{-2}$	0.04611759718129048274**8**1...
Log10 (2)	$3.010299956639810\times10^{-1}$	0.30102999566398119521**37**...
Power (π/3, π/5)	1.029400445385110	1.02940044538511177579556**79**...
Sqrt (π/3)	1.023326707946490	1.02332670794648848847**95**...

a. Differences to the reference are highlighted in the reference if larger than rounding lets assume.

b. K indictes calculations done with Kalkulator 2.5 (Andrzej Wrotniak, `http://www.wrotniak.net/works/kalkulator/`). This is because some functions are not available on the Windows calculator.

c. The 14 values for the test are
11.1, -11.9, 12345679.0, -12345679.0, 124.5, -17, 18.9, 2.2, 2.5, 2.5, 2.5, 31.31, 312345679, -312345679
According to the results the second method (summing x/n) is much better and hence implemented.

Comparison of numeric results

The numeric results from the Equations Editor and the **FMcalc** package show no difference, at least in these short formulas:

Equation editor (number crunch)		FMcalc
Formula	**Result**	**Result {E12}**
$a = e^{27}$	$a = 5.3204824\times10^{11}$	$5.320482406018\times10^{+11}$
$a = \sin\frac{\pi}{123}$	$a = 0.025538627$	$2.553862673256\times10^{-2}$
$a = \log(713)$	$a = 6.5694814$	6.569481420414
$a = \operatorname{atan}(0.025538627)$	$a = 0.025533077$	$2.553307689150\times10^{-2}$

Numeric range

The following explanations are from the ECMAScript reference.

Boolean

Boolean types interpret 0 as **false** and all other as **true**.

Integer

An integer is a whole number (has no decimal point) ranging from `-2147483647` to `2147483647`.

Some ECMAScript operators deal only with integers in the range -2^{31} through $2^{31}-1$, inclusive (see above), or in the range 0 through $2^{32}-1$, inclusive (`0` to `4294967295`).

Real

A real number represents the double-precision 64-bit format IEEE 754 values as specified in the IEEE Standard for Binary Floating-Point Arithmetic[43)].

Single precision provides a range of $\pm \approx 10^{-44.85}$ to $\approx 10^{38.53}$. The range for double precision is $\pm \approx 10^{-323.3}$ to $\approx 10^{308.3}$.

In double precision the 52 mantissa bits result in about 15 decimal digits for the representation of a number.

Zero is a special value denoted with an exponent field of all zero bits, and a fraction field of all zero bits. Note that -0 and +0 are distinct values, though they both compare as equal.

In addition there are definitions for **NaN** (not a number), **negative Infinity** (**-∞**) and **positive Infinity (+∞)**.

NaN

A `QNaN` (Quiet NaN) is a **NaN** with the most significant fraction bit set. `QNaN's` propagate freely through most arithmetic operations. These values pop out of an operation when the result is not mathematically defined.

An `SNaN` (Signalling NaN) is a **NaN** with the most significant fraction bit clear. It is used to signal an exception when used in operations. `SNaN's` can be handy to assign to uninitialized variables to trap premature usage.

Coding style

I tried to create maintainable code because it may well happen that it need to be handled by someone else (I'm now 75).

Prefixes for variables

As far as reasonable I followed the rules of Hungarian_notation:

Prefix	Property	Example
a	array	`aMenuItems, aMatch, aPieces, aItemsArray`
b	boolean	`bIsOK, glbl.bDoingBook`
KLD_C	global	`KLD_C is an object with many items: KLD_C.oCurrentDoc, KLD_C.aoSeriesMarkers, KLD_C.APP_NAME`
i	integer	`indexOfChar, iLength, iParam`
l	locals	`localArray, localText`
o	object	`oDoc, aoStatemnts, oBookComp`

43 For an exact definition see for example [10]

Prefix	Property	Example
r	real	`rResult`
re_	regular expression	`re_SeriesName, re_IniItem`
s	string	`sMarkerName, sParam`
UItxt	global	`KLD_C.UItxt is an object, the items of which are strings, defined in XML files for one language per file. KLD_C.UItxt.SetUpMenus.@menu00.toString()`
settings	global	`KLD_C.settings is an object, the items of which are strings or booleans: settings.FmtSup, settings.bDebug`
w	window	`wPalC, wPalV`

Naming of variables and functions

Since the prefixes are lower case, the actual name starts with upper case.

Naming constants

Global constants are named with upper case:

```
APP_NAME = "FMcalc";    // the name of the game
FMCALC_VERSION = "1.3"; // current version of script
LIST_SEP_CHARS = ", ";  // List separator
MAX_SAFE_INTEGER = 9007199254740991;
```

Note: *In the script these constants are not declared* `const`*, but as global variables* `KLD_C.xxx`*. The* `const` *construct would require to reset the ESTK environment for each run of script. This is to much an obstacle in debugging.*

Naming of functions

Function names start with an upper case letter - similar to ExtendScript methods. I tried to use verbal names: `FindXyz`, `FormatValue`, `HandleSeriesMarker`, `PutRefPageItems`, `IsInArray`, etc. That helps to be in line with “Maintainable JavaScript” by Nicholas C. Zakas.

The ‘casing’ in ES is anyway quite mixed up:

- ES objects are named with upper case initial. The only exception is `app`.

```
bodyPage = Doc.FirstBodyPageInDoc;
```

- ES methods = functions on objects start with upper case:

```
bkFMbiblioMenu.DefineAndAddCommand ()
if (app.ActiveDoc.ObjectValid() === 0) { …
```

- ES properties start with upper case:

```
docFile = app.ActiveBook.Name;
bodyPage = oDoc.FirstBodyPageInDoc;
bColour = oDoc.FirstBodyPageInDoc.PageBackground;
```

- ScriptUI Properties start with lower case:

```
Checkbox.maximumSize
DropDownList.selection
```

- JS Methods start with lower case:

```
helpFile = File($.fileName.replace (/\.jsx(?:bin)?$/i , ".pdf"))
nCitations = gasFndCitations.length;
specLogFile.writeln (gasFndCitations[i]);
specLogFile.close ();
```

- For some functions there are two versions:

```
alert ("ES");
Alert ("JS");
```

```
  Console ("FM-console");
```

Comments in general

I try to describe the function:

```
          kText = localText.substring(indexOfChar); // rest of the statement
          index = ContainsUserVar (gasUserVariables, kText, true);  // … of variable
in array
          sName =  gasUserVariables [index];
          oVar = goCurrentDoc.GetNamedVarFmt(sName);
         sValue = oVar.Fmt;                    // current contents (not empty due to …
          sValue = Unformat (sValue) ;            // 'un-format' for internal use
          iLength = sValue.toString().length;
          kText = kText.replace (sName, sValue);  // replace only from indexOfChar on
```

Large functions receive ‘header comments’

```
case "#":                                  //--- user variables -----------------
##########
```

Comments in functions

Every function header has a short description. Not all comment elements are present in all routines.

```
function QSimpson (arXvalues, arYvalues, nValues) { //=== Quasi Simpson integration
// Quasi Simpson integration for uneven intervals by successive parabolas
// Arguments arXvalues: abcissa values; values must either inc. or dec. monotonic;
//                      successive values must not be equal (x' is infinite)
//           arYvalues: ordinate values
//           nValues:   number of points (may be even or odd)
// Returns   Function returns the approximate integral rIntegral
// Called by eval in EvaluateStmntsC
// Calling   CreateMatrix, InvMatrix3
// Reference Fortran program AREA by Bruce Cameron Reed, … <reed@alma.edu>, 2014
// Attention At least 3 x/y pairs must be defined.
// Comment   Arrays aX and aY are used within function for 3-point sets of data
//           rAlpha,rBeta,rGamma: parabolic fitting coefficients (a,b,c)
//             of paper by Cameron Reed
//           bEven = +1 if even number points, bEven = -1 if odd number points
var i, j, k, m, bEven, aX = [], aY = [], aMatrix = CreateMatrix (3), rAlpha, rBeta,
rGamma,
    aMatrixInverted = CreateMatrix (3), rBlock, rIntegral = 0.0;

...
} //--- end QSimpson
```

Why global variables

You can read in nearly any JavaScript book that global variables are of evil (for example [15]). Nevertheless I decided to use them, because otherwise I would need a large number of arguments in most functions.

Naming of variables and functions on page 62 clearly identifies the global variables. In addition all global variables are initialised explicitly in module `FMcalc_Globals.jsx`.

Prototype definition for `oTextScheme` and `oStatemnt` are also in `FMcalc_Globals.jsx`.

Test set-up

Debugging pains

Debugging a script with a menu is cumbersome:

- When you start a script via FrameMaker, the ESTK isn't involved at all. This happens also when the script was started via ESTK and installs menus - and then the functions are invoked by a menu item.
- In these cases breakpoints do not function at all.
- Hence I have a function `DebugMenu` in the script, which sets up a list rather than a FrameMaker menu.
- The script is registered if it contains `SetupNotifications` and `Notify` functions It stays registered until the close of FM. Hence during development manual un-registration is necessary before each modification and restart of the script.
- I got problems in later development with the Notify method and hence reduced it to a single trig-ger:`FA_Note_PostActiveDocChange`

FM environment

I have tested in 10.0.3.419, 13.0.5.547 and 14.0.0.361.

The ExtendScript scripting engine

Version 4.5.5; build version 79.535558; Build date 2013/03/20-12:12:04

ScriptUI version 6.2.2

Checking the code

I have checked each function with JsHint.com.

Function	cc	new	Replaced by ... to reduce Cyclomatic Complexity (see Wikipedia)
PutRefPageItems	14	3	GetRefPagePara, PutRefPageScheme, PutRefPageSettings
GetRefPageItems	25	4	GetRefPageCategory, GetRefPageSetting
NumberOfArguments	18	9	itself, CheckRequiredArgs
EvaluateS	33	8	itself, EvaluateSFillObject, EvaluateSNumeric, EvaluateSRoman, EvaluateSTextScheme
EvaluateParameterS	23	8	itself, WriteFeedback
EvaluateStmntsC	47	22	itself, WriteFeedback
CheckTermElementsS	30	21	itself, CheckTermElementSfillStmtStructure

Initialisation files

FMcalc_tpl.fm

This FM-10 file contains the reference page `FMcalc` which keeps the defaults and also the collected lists. This reference page is copied to the current document if it does not yet exist there.

On this reference page the data is distinguished by ¶-formats, which are used only locally - not put into catalogue:

¶ format	Example contents
Ref-Comment [a]	Setting for FMcalc concerning this document 2016-05-17 Klaus@Daube.CH `[ProgramItems]`
Ref-Setting	`FmtSup = super`
Ref-Variables	`#apples`
Ref-Locations	`Above(n)`
Ref-Formats	`{Enn}`
Ref-Series	`series1 = in old times, earlier, shortly ago, today, tomorrow, next week, in eternity`

a. For better recognition this format uses italic coloured text

Project statistics

2017-11-02

Number of Modules12 + 3 language modules
Number of Functions222
Number of program lines5125
Number of comment lines1310
Lines in comment blocks117
Number of empty lines592
Number of lines7144 (
Project time [h] 2016712.50
Project time [h] 2017223.50
Documentation pages68

(Re-) Calculate in document

This document in FM-16 (2022)
68 #calc markers, 4 #series markers
With full debug output25 min, 4700 lines
With no debug output12 sec

Sources and references

[1] Vamitul on https://forums.adobe.com/thread/1725631 states 2015-05-10: ExtendScript is based on ECMAScript version 3 as of 1999, that is the same one used by Internet Explorer 5.

[2] Community: FrameMaker Scripting | Adobe Community. Available: https://forums.adobe.com/community/framemaker/extendscript (2015-04-02).

[3] ECMA script http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[4] Scripting Guide for FM-13: http://help.adobe.com/en_US/framemaker/scripting/framemaker_scripting.pdf

[5] FrameMaker 12 Object Reference.chm: http://www.jongware.com/idjshelp.html

[6] ScriptUI tutorial by Peter Kahrel: http://www.kahrel.plus.com/indesign/scriptui.html (pdf avalable there).

[7] ADobe FDK documentation for FM-13: Programmers Guide, Reference.

[8] http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/scripting/pdfs/javascript_tools_guide.pdf

[9] Debra Herman's blog "Extending FrameMaker": http://extendingframemaker.blogspot.ch

[10] IEEE Standard 754 Floating Point Numbers: http://steve.hollasch.net/cgindex/coding/ieee-float.html

[11] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/round

[12] Rapid ScriptUI by Steven Bryant greatly helps to set up dialogues, although an installation of Illustrator, InDesign or Photoshop is required. It has not been updated after 2009: http://scriptui.com/default.aspx (USD 50.-).

[13] Axel Rauschmayer, Speaking JavaScript: An In-Depth Guide for Programmers. O'Reilly 2014, isbn=1449364993]. HTML version is free at http://speakingjs.com/ but can not be searched.

[14] M. Abramowitz and I. Stegun. Handbook of Mathematical Functions: With formulas, graphs, and mathematical tables. New York: Dover Publications, 1972 (based on 10th printing of NBS edition with corrections).

[15] D. Crockford, *JavaScript: The good parts: Unearthing the Excellence in JavaScript*, 2nd ed. Farnham: O'Reilly, 2013.

[16] Physikalisch-Technische Bundesanstalt: *Das neue Internationale Einheitensystem (SI)*. Braunschweig, 2017.

[17] Jongware by Theunis de Jong (1966–2020). Although based on FM-12 this chm-file with the object-definitions is invaluable in scripting work.