User Guide

For:

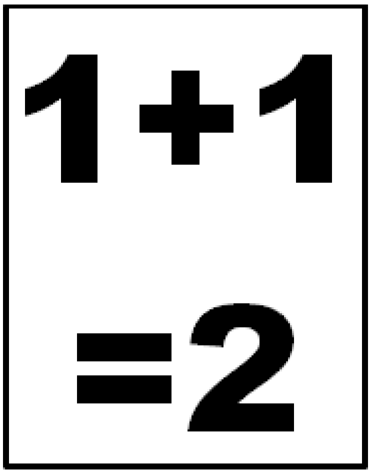
Rhyscitlema Calculator For All

Software Version 1

By:

Rhyscitlema

http://www.rhyscitlema.com/calculator-for-all



13 September 2015

Contents

1 General Information....3

2 Math Function Expression Text (MFET)....4

- 2.1 Value Structure....4
- 2.2 Supported Operators....4
- 2.3 Software-Defined Math-Constants....5
- 2.4 Software-Defined Math-Functions....5
- 2.5 Software-Defined Extra-Functions....6
- 2.6 Replacement Operator :=....6
- 2.7 Conditional Operators “ ? : ”....6
- 2.8 Per-value Operation....7
- 2.9 User-Text....7
- 2.10 Access Control....7
- 2.11 More About MFET....7

3 MFET Container and File....8

- 3.1 MFET Container....8
- 3.2 MFET File....8

4 User Interface....9

- 4.1 Software-specific features....9
- 4.2 Character vs Partly-Code-Number Text Display....10

5 Frequently Used Features....11

- 5.1 Commenting....11
- 5.2 Error Tracking....11

6 Simple Examples....12

- 6.1 Basic Example....12
- 6.2 Circles and Spheres....12
- 6.3 Quadratic Equation....12

1 General Information

Rhyscitlema Calculator For All is a software to evaluate any mathematical expression, and also use user-defined variables and functions. It is derived from the Rhyscitlema Graph Plotter 3D software.

The software evaluates variables and functions defined with their associated mathematical expressions in a text form called Math Function Expression Text (MFET). MFET is a syntax in which mathematical expressions are written. It specifies the way in which variables and functions are defined as well as the way in which they are used together with operators and operations.

By using the well diversified operators and operations together and by calling user-defined variables and functions between themselves, the MFET syntax enables the implementation of very complex and compact mathematical expressions in ways that are very Useful, Simple, Efficient, Robust and Flexible.

For now only technical information is provided.

2 Math Function Expression Text (MFET)

MFET is simply a statement of the form:
output_value_structure = mathematical_expression ;

It should be terminated by the end-of-statement character, which is the semi-colon ';'. However it can be terminated by the end of a file or the end of a text. The end-of-statement is only necessary when there are multiple MFET statements, as will be found in a MFET container.

The following sub-sections discuss the MFET syntax in more details.

2.1 Value Structure

The value structure defines the structure of the value of an expression after it has been evaluated. Particularly:

- A value structure is represented using brackets, comma-separators and names.
  - ((w,x,y),z) , {w,x,{y,z}}
- A single value is the most common value structure
  - x, (x), ((x))
- Vectors and matrices are special types of value structures
  - (w,x,y,z)
  - ((w,x),(y,z))
- A variable or function is defined using a value structure
  The brackets **{}** must be used.
  - {x, y} = (2, 3) + (4, 5) ;
- The parameter to a function is defined using a value structure
  The brackets **()** must be used.
  - f((x,y),z) = x+y+z ;

- A valid MFET is:
  - {f,g}(x,y) = (x+y, x-y) ;

  The interpretation is as follows:
  - {f, g}(x,y) is a value-structure definition of two functions: f(x,y) and g(x,y)
  - (x+y, x-y) is a mathematical expression that evaluates to a 2-value vector
  - f(x,y) is equal to (x+y, x-y)[0] and so evaluates as (x+y)
  - g(x,y) is equal to (x+y, x-y)[1] and so evaluates as (x-y)

- The brackets {} and () are always interchangeable. The only exception is that {} is used when defining the value-structure of a variable or function while () is used when defining the value-structure of a function parameter.

2.2 Supported Operators

The order in which the operators below are listed is also their **precedence order**. That is what comes lastly is evaluated firstly. For example, because the operator times * comes after the operator plus +, an expression like 2+3*4 will evaluate as (2+(3*4)).

The logical operators evaluate to 1.0 or 0.0 (true or false). A single-value operand is considered true if it evaluates to non-zero, and false if it evaluates to zero.

- open-close brackets **( )** and **{ }**
- brackets for subscripting **[ ]**
- comma separator **,**

- Concatenate **.,**
- Replacement **:=**
- Conditional **? :**

- Logical OR **or** (does only per-value operation)
- Logical AND **and** (does only per-value operation)
- Logical NOT **not**

- Equal = or == (or **.==** for per-value operation)
- Not equal **!=** (or **.!=** for per-value operation)
- Less than < (or **.<** for per-value operation)
- Greater than > (or **.>** for per-value operation)
- Less than or equal to <= (or **.<=** for per-value operation)
- Greater than or equal to >= (or **.>=** for per-value operation)

- Plus + (does only per-value operation)
- Minus **-** (does only per-value operation)
- Positive +
- Negative **-**
- Times ***** (or **.*** for per-value operation)
- Divide / (or **./** for per-value operation)
- To-power ^ (or **.^** for per-value operation)
- modulo (remainder) **mod** (does only per-value operation)
- Dot Product **.**
- factorial **!**
- Transpose ^**T**

2.3 Software-Defined Math-Constants

- E constant **e**
- PI constant **pi**
- square root of negative one **i**

2.4 Software-Defined Math-Functions

- 10 raised to power **E()**
- e raised to power **exp()**
- logarithm base 10 **log()**
- logarithm base e **ln()**
- square root **sqrt()**
- floor **floor()**
- fullfloor **fullfloor()**

- complex number magnitude **abs()**
- complex number argument **arg()**
- complex number real part **real()**
- complex number imaginary part **imag()**

- complex number conjugate **conj()**

- trigonometric sine **sin()**
- trigonometric cosine **cos()**
- trigonometric tangent **tan()**
- trigonometric sine inverse **asin()**
- trigonometric cosine inverse **acos()**
- trigonometric tangent inverse **atan()**

- hyperbolic sine **sinh()**
- hyperbolic cosine **cosh()**
- hyperbolic tangent **tanh()**
- hyperbolic sine inverse **asinh()**
- hyperbolic cosine inverse **acosh()**
- hyperbolic tangent inverse **atanh()**

2.5 Software-Defined Extra-Functions

- length of value-structure **length()** , evaluates to a single value
- size of vector or matrix **size()** , evaluates to a 2-value vector (rows, columns)

- print expression **print()**
- print on condition **printOnC(,)**
- print on error **printOnE(,)**
  These functions are explained under the section on Frequently Used Features.

2.6 Replacement Operator :=

Consider the following expression:

- (2 := current + 1)

The replacement operator **:=** evaluates what is on its right-hand-side (RHS), then replaces what is on its left-hand-side (LHS) with the result. '*current*' is a special variable associated with the replacement operator. It is considered special only when there is an associated replacement operator. It simply is the value of the current LHS. This causes the replacement to be an update.

2.7 Conditional Operators “ ? : ”

The conditional operators enable the use of conditional expressions. So it enables piece-wise definition of variables and functions.

The syntax is:

- (condition) ? (on true) : (on false)

which is similar to:

- (on true) if (condition) or
  (on false) otherwise

If (condition) evaluates to:

- non-zero then (on true) is evaluated and (on false) is ignored.
- zero then (on true) is ignored and (on false) is evaluated.

For example, consider the expression: 2 + ((5>6) ? 1 : -1)
The result is evaluated as 2 + (-1) because the expression (5>6) evaluates to zero – false.

The two operators have the same precedence level. So an expression like:

- 2 + ((5>6) ? 1 : (5<6) ? 0 : -1)

which is the same as:

- 2 + ((5>6) ? 1 :
  (5<6) ? 0 : -1)

Will evaluate as:

- 2 + 1 if (5>6) or
  0 if (5<6) or
  -1 otherwise

2.8 Per-value Operation

Some operators are described to do **per-value operation**. This means that the operator is applied to each of the corresponding values of the value-structures of the operands, with the output being of the same value-structure as the operands. Typically, the plus + operator does per-value operation. Ex:

- (1, 2, 3) + 4 evaluates as (1+4, 2+4, 3+4)
- (1, 2, 3) + (4, 5, 6) evaluates as (1+4, 2+5, 3+6)

An operation that can only be performed on a single-value argument (mainly the Math functions) will do a per-value operation when the operand is a value-structure like a vector or a matrix. Ex:

- sin(1,2,3) evaluates as (sin(1), sin(2), sin(3))

2.9 User-Text

A user-text is anything found inside the double quotes " ".

It is possible to display user-texts as part of the result of an evaluation. For example on evaluating the expression (2+3, "text"), the result will display as (5, text). Note however that operations on user-texts are not possible – at least for now!

2.10 Access Control

A MFET statement can be preceded by the text **\private** so to mean private access. This ensures that a variable or function is not accessed outside the scope in which it is defined. This scope is the MFET container - the collection of MFETs for which the particular MFET is part of.

2.11 More About MFET

- By default a vector is a **column vector**.

- A function parameter overrides any user-defined variable or function, which overrides any software-defined variable or function.

3 MFET Container and File

3.1 MFET Container

A MFET container is a collection of MFETs. When a MFET container is evaluated, its very first statement is evaluated and the result is returned (maybe to display it next). It is *only* this first statement that is not and must not be a variable or function definition.

Below are examples of MFET containers:

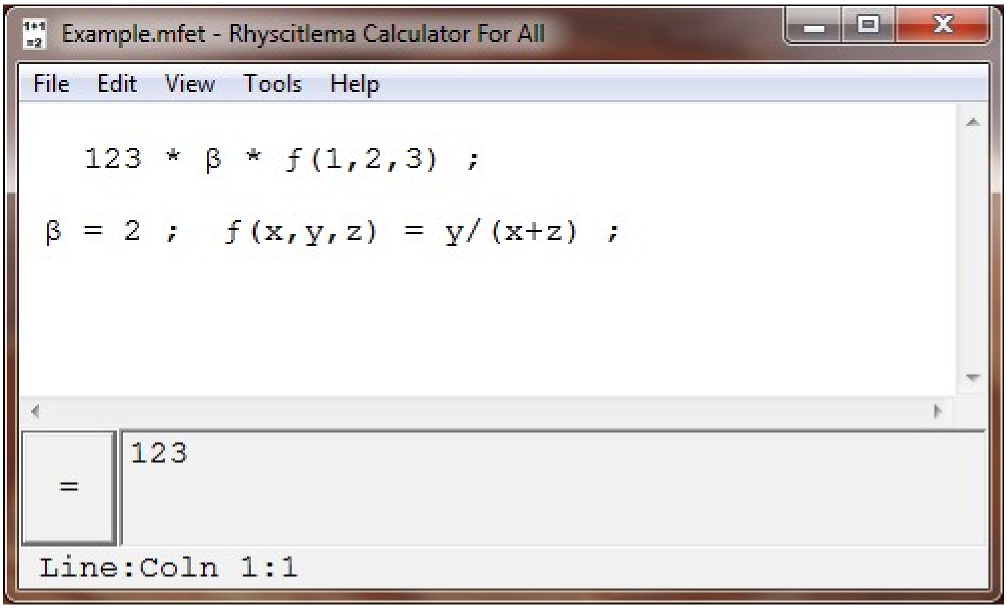
1.
    - The container is empty or has only space-type characters.
2. 1+1
    - There is only the text '1+1'. This is the only and therefore the first statement. It is evaluated to '2'.
3. 2a
    - The first statement is '2a' which is evaluated to an error due to 'a'.
4. 2a ; a=3
    - There are two MFETs: '2a' and 'a=3'. The first is '2a' which is evaluated to 6.
5. ;
    - There are two MFETs. Both are either empty or have only space-type characters.

3.2 MFET File

A MFET file is a file storing a MFET container.

A MFET file can be used as a library. Such a file will contain pre-developed variables and functions that are called in other variables and functions outside the file (unless they have private access). The same file can even also be used in other applications.

4 User Interface



```
  123 * β * f(1,2,3) ;

β = 2 ;  f(x,y,z) = y/(x+z) ;
```

```
123
```

```
Line:Coln 1:1
```

The user interface largely does text and file editing. Features that are common to most text file editing software will not be discussed.

When a MFET file is loaded its content is displayed in the large user entry text field. The content is then automatically evaluated and the result is displayed in the small result text field.

4.1 Software-specific features

Below is the list of software-specific features along with where to activate them:

- Menu → Tools → Evaluate:
    - Does the same thing as the equal '=' button.

- Menu → Tools → User Interface:
    - The layout of the user interface is provided by a certain MFET container described as the User Interface Definition Text. The option 'Show' displays this text in the large user entry text field. The option 'Load' gets the content of the large user entry text field and evaluates it as the User Interface Definition Text. That is it evaluates the MFET container, not in the usual way (as the purpose is not to get a single result), but instead by evaluating each of the primary variables. It then updates the displayed user interface.

- Menu → Edit → Go To:
    - Repositions the caret. The provided entry is a MFET entry.

- Menu → Edit → Convert Text:
    - These are features to choose whether to show a text in its normal character form, partly-code-number form, or fully-code-number form. Refer to the sub-section below.

There is only one button, the equal '=' button. This evaluates the content of the large user entry text field as a MFET container and displays the result in the small result text field. Evaluation can also be performed by pressing the keys **Shift+Enter**.

4.2 Character vs Partly-Code-Number Text Display

This is a feature for file editing. It is especially useful to:

- Efficiently use the Find-Replace feature
- Provide a character through its code number
- Know the code number associated to a character

A character is stored in a computing device in the form of a number. The displayed character is just the meaning given to that number. For example the character '0' is stored as the hexadecimal number 30, corresponding to the decimal number 3*16+0 = 48. This is the *Code Number.*

Here, a character with a 4-digit hexadecimal code number of XXXX is represented as **\uXXXX**. For example the character '0' is represented as \u0030. This is described as the fully-code-number representation.

In certain contexts such as software programming, some characters have alternative representations. Particularly:

- The tab character is represented as \t (code number \u0009)
- The carriage-return character is represented as \r (code number \u000D)
- The line-feed or newline character is represented as \n (code number \u000A)

In regard to this feature of the user interface, a character that is said to be displayed as partly-code-number is a character displayed in one of the following forms:

- The **normal character display**
  - The '0' character will display as '0'
  - The newline character will display a new line
- The **alternative representation** if it is available
  - The '0' character will display as '0' since it has no alternative representation
  - The newline character will display as \n (and there will be no new line)
- The **fully-code-number representation**
  - The '0' character will display as \u0030
  - The newline character will display as \u000A

The feature “Set Partly is Fully Code Number” is a feature to redefine the partly-code-number representation of a character, so that using the feature “Character to Partly Code Number” will keep most characters unchanged while showing the fully-code-number representation of the affected characters only. To use this feature a text containing the targeted characters in their fully-code-number representation is selected.

These features can be applied to any text selected, including texts in the find-replace windows.

5 Frequently Used Features

5.1 Commenting

The character **#** is used to comment a single line. That is everything from the character up to the encounter of a newline is ignored.

#{ and **}#** are used for *bulk-commenting*. That is everything from the specific starting point to the specific stopping point is ignored. A }# only closes a matching #{. Therefore a bulk-comment can contain another bulk-comment.

Priority is given to the first encountered commenting. That is for example: #...#{ or #...}# on a single line, is a valid and single comment instance (the line is commented). Also #{...#...}# is a valid and single bulk-comment instance (the bulk of text is commented).

A commented portion becomes a space character.

5.2 Error Tracking

Three software-defined functions are provided for error tracking.

1) print(expression) - Print
This evaluates the expression inside the brackets and prints the result as an error message.

2) printOnC(on_condition, if_condition) – Print On Condition
This evaluates the second argument *if_condition*. If the result is non-zero then the first argument *on_condition* is evaluated and its result is printed as an error message. Else everything continues as if there was only the first argument and no printOnC().

A similar expression is: (if_condition) ? print(on_condition) : on_condition

3) printOnE(if_error , on_error) – Print On Error
This evaluates the first argument *if_error.* If it turns into an error (like division by zero) then the second argument *on_error* is evaluated and its result is printed as an error message. Else everything continues as if there was only the first argument and no printOnE().

6 Simple Examples

6.1 Basic Example

<u>MFET container:</u>

```
   123 * β * ƒ(1,2,3) ;

 β = 2 ;  ƒ(x,y,z) = y/(x+z) ;
```

<u>Result of evaluation:</u>

```
123
```

6.2 Circles and Spheres

<u>MFET container:</u>

```
  Result ;

 r = 1 ;            # radius
 C = 2*π*r ;        # Circumference of a circle
 A = π*r^2 ;        # Area of a circle
 S = 4*π*r^2 ;      # Surface of a sphere
 V = (4/3)*π*r^3 ; # Volume of a sphere
 π = pi ;           # 'pi' is software-defined

 Result = (C , A) , (S , V) ;
```

<u>Result of evaluation:</u>

```
((6.283, 3.142),
 (12.566, 4.189))
```

6.3 Quadratic Equation

<u>MFET container:</u>

```
  x1 , x2 , f(x1) , f(x2) ;

 x1 = ( -b + sqrt(b^2 - 4*a*c) ) / (2a) ;

 x2 = ( -b - sqrt(b^2 - 4*a*c) ) / (2a) ;

 f(x) = a x^2 + b x + c ;

 a = 2; b = -2;  c = 1;

 # Solving for x where f(x) = 0
 # Change the coefficients a, b and c
```

<u>Result of evaluation:</u>

```
(0.5 + 0.5i, 0.5 - 0.5i, 0, 0)
```