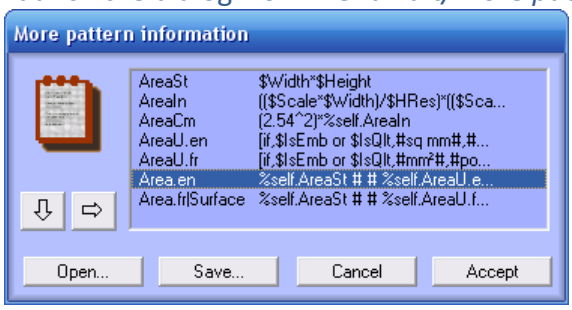
myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

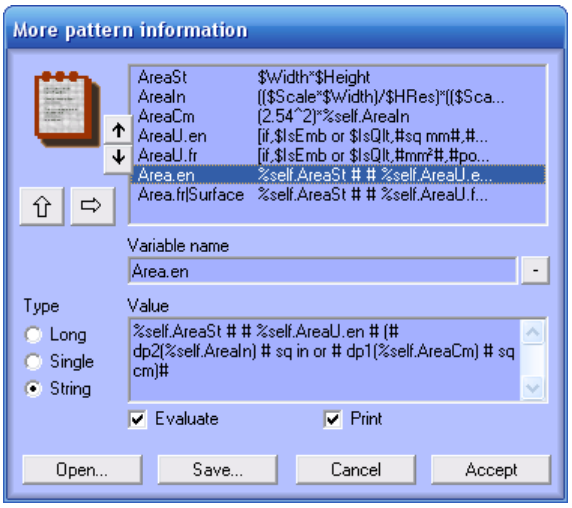
myriaCross editor version 1.51.01 and above allows you to define your own pattern information data using named variables inside named sections ; this data is stored in your pattern file and can appear when printing information and notes.
myriaCross editor version 1.51.01 and above also has the ability to evaluate mathematical expressions defined with integer and real values, operators, functions, texts, variables returning pattern-specific values and extended data variable values.

Adding pattern information data :

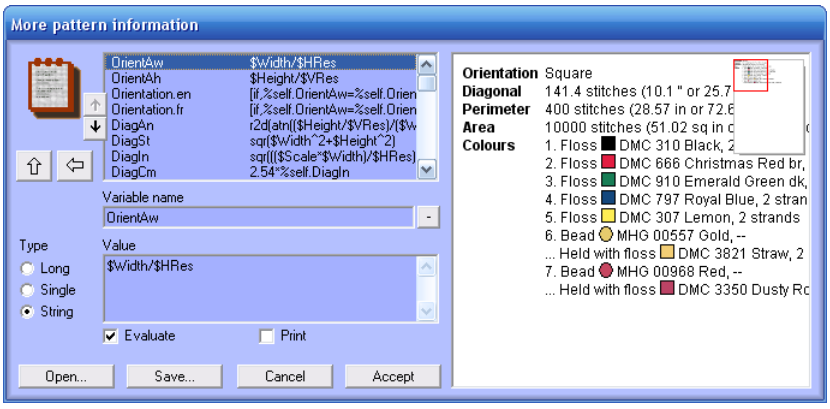
Launch the dialog from menu *Edit/More pattern information* or right-click on icon  .



The list shows defined variables with their associated values. Obviously, a new pattern will show no variables until you create some.
Click the **arrow down** button to enlarge the window and start edition. The arrow down button then changes to an **arrow up**. Clicking it will end the edition and reduce the window.



Tiny arrow buttons let you change the **order** by moving selected variable up or down in the list.
To **create** a variable, enter its ***name***, choose the data ***type*** (long, single or string) then enter a ***value***.
Check the ***Evaluate*** option for a string variable whose content must be evaluated on use. Check the ***Print*** option to include current variable in further *Information and notes* printouts.
Actually create your variable using ***+ (plus)*** button or **delete** selected variable using ***– (minus)*** button.



Click the **arrow right** button to see a preview of your variables when printed ; it changes to an **arrow left**. Click it to close the preview area.
Only variables set as printable will appear.
Click ***Cancel*** button to ignore all changes or click ***Accept*** button to keep them. Click ***Save*** button to record your pattern information in a file. Click ***Open*** button to load pattern information from a file, either as new information or added to existing one. Since information files are in a readable format, you can manually define information in a file then load it as a whole (see appendix C below).
Note : changing type, value, flags (or localised named part – see later) of a selected variable automatically updates it. Entering a name that matches an existing variable automatically selects it in the list. Names cannot contain & # @ $ < > = ( ) + - * ^ / \ [ ] { } , ; : ! or ?

myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

Title	Daisies
Author	Ellen Maurer-Stroh
Copyright	2000 Ellen Maurer-Stroh
Size	84 x 79 stitches (6 x 5.6 " or 15.2 x 14.3 cn
Cloth	14 count (5.5 stitches/cm) (14ct #357 Aida,
1 stitch	0.071 x 0.071 " or 1.81 x 1.81 mm
Notes	Full Stitches - use 3 strands of floss Backstitches - use 1 strand of floss
Orientation	Landscape
Diagonal	115.3 stitches (8.24 " or 20.9 cm) at 43.2°
Perimeter	326 stitches (23.29 in or 59.1 cm)
Area	6636 stitches (33.86 sq in or 218.4 sq cm)

Here is a sample printout ; adding to standard pattern information, now appear the variables you set as printable in above dialog.
As you can see, this feature allows you to print almost all what you need.

Appendix A. Storing named extended data

Named data structure :

Section1 *Variable1* ➔ *Value*
Variable2 ➔ *Value*
…
Section2 *Variable1* ➔ *Value*
Variable2 ➔ *Value*
…

Note : your pattern information data is stored in section *MoreInfo* ; you cannot define other sections yet ; next versions may allow you to do so in order to manage new features.

Available data types for variables :

You can either define variables as **long** integers, **single** precision reals or character **string**s. Long integers are in the range -2 147 483 648 to 2 147 483 647 while single reals are in the range -3.402823e38 to -1.401298e-45 for negative values and 1.401298e-45 to 3.402823e38 for positive values. String variables can contain constant values, variables or mathematical expressions to be evaluated by the program.

A sample simple pattern information :

```
MoreInfo
 MyLong    =    1234
 MySingle  =    17.99
 MyString  =    "PAT001"
```

A sample advanced pattern information :

```
MoreInfo
 AreaStitches =    "$Width*$Height"
 AreaUS       =    "($Width/$HRes)*($Height/$VRes)"
 AreaMetric   =    "(2.54^2)*%self.AreaUS"
 Display      =    "#Area = # %self.AreaStitches # stitches#
                    #Area US = # dp3(%self.AreaUS) # sq in#
                    #Area Metric = # dp2(%self.AreaMetric) # cm²#"
```

Assuming current pattern is 40 x 50 stitches at 14 stitches per inch, and you enabled only variable *Display* for print, printed information and notes would also show :

Display Area = 2000 stitches
Area US = 10.204 sq in
Area Metric = 65.83 cm²

myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

Note : the expression *%self* refers to the current section name ; for instance, *%self.AreaUS* evaluates as *MoreInfo.AreaUS.*

Appendix B. Evaluating mathematical expressions

Available operators :

xor	Exclusive bitwise or	(5 xor 2 returns 7	7 xor 2 returns 5)
	Exclusive logical or	(0 xor 0 = 0	0 xor -1 = -1
		-1 xor 0 = -1	-1 xor -1 = 0)
or	Bitwise or	(5 or 2 returns 7)	7 or 2 returns 7)
	Logical or	(0 or 0 = 0	0 or -1 = -1
		-1 or 0 = -1	-1 or -1 = -1)
and	Bitwise and	(5 and 2 returns 0)	
	Logical and	(0 and 0 = 0	0 and -1 = 0
		-1 and 0 = 0	-1 and -1 = -1)
>=	Greater than or equal to	(5 >= 2 returns -1	2 >= 5 returns 0)
<=	Lower than or equal to	(5 <= 2 returns 0	2 <= 5 returns -1)
>	Greater than	(5 > 2 returns -1	2 > 5 returns 0)
<	Lower than	(5 < 2 returns 0	2 < 5 returns -1)
<>	Not equal to	(5 <> 2 returns -1	5 <> 5 returns 0)
=	Equal to	(5 = 2 returns 0	5=5 returns -1)
+	Add	(5 + 2 = 7)	
-	Substract	(5 – 2 = 3)	
mod	Modulo	(5 mod 2 = 1, remainder of integer divide)	
**	Integer divide	(5 \ 2 = 2, integer part of divide)	
***	Multiply	(5 * 2 = 10)	
/	Divide	(5 / 2 = 2.5)	
^	Power	(5 ^ 2 = 5 * 5 = 25)	

Note : operators *xor*, *or*, *and*, *mod* **must** be surrounded by spaces.

Available functions :

abs	Absolute value	(abs(2.3) = 2.3	abs(-2.3) = 2.3)	
int	Integer value	(int(2.3) = 2	int(-2.3) = -3)	
fix	Integer value	(fix(2.3) = 2	fix(-2.3) = -2)	
sgn	Sign	(sgn(2.3) = 1	sgn(0) = 0	sgn(-2.3) = -1)
sqr	Square root	(sqr(3) = 1.732)		
exp	Exponantial (base e)	(exp(1) = 2.718)		
log	Logarithm (base e)	(log(1) = 0	log(2.7182818) = 1)	
log10	Logarithm (base 10)	(log10(5) = 0.698	log10(50) = 1.698)	
rnd	Random value	(rnd(any) = real between 0 and 1)		
sin	Sine	(sin(pi/6) = 0.5)		
cos	Cosine	(cos(pi/6) = 0.866)		
tan	Tangent	(tan(pi/6) = 0.577)		
asn	Arc sine	(asn(0.5) = pi/6)		
acs	Arc cosine	(acs(0.866) = pi/6)		
atn	Arc tangent	(atn(0.5776) = pi/6)		
not	Logical/bitwise not	(not(-1) = 0	not(0) = -1)	

myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

Note : functions *sin*, *cos*, *tan* require an argument in radians ; functions *asn*, *acs*, *atn* return an angle in radians. Use following functions to convert between radians and degrees :

d2r	Degrees to radians	(d2r(180) = 3.1415926)
r2d	Radians to degrees	(r2d(3.1415926) = 180)

Available formating functions (conversion to string) :

dp0	No decimal places	(dp0(0.5776) = "1")
dp1~dp8	1 to 8 decimal places	(dp1(0.5776) = "0.6", dp2(0.5776) = "0.58")
quote	Add quote marks	(quote(0.5) = "«0.5»", quote("ab") = "«ab»" quote("5 + 1") = "«6»")
a2s	Convert anything to string	(a2s(10) = "10", a2s(1,2) = 1.2, a2s("ab") = "ab")

Note : you cannot blend formating functions and operators ; for instance, the expression *dp1(3.25)*3* will return 0 while *dp1(3.25*3)* will return 9.8 that is correct.

Specifying texts :

Use this syntax : *#TextString#* ; if you need use character # inside the text string, specify &#.
Example 1: *#Result is # 5+3 # inches#* returns *Result is 8 inches*.
Example 2 : *#Result &#1 is # 5+3 # inches#* returns *Result #1 is 8 inches*.
You can also define special texts that will print as pictures :

Use syntax *!Box:<RGB colour>* to create a coloured box.
Use syntax *!Dot:<RGB colour>* to create a coloured filled circle.
Use syntax *!Pnt:<RGB colour>* to create a coloured point.
Use syntax *!Lin:<RGB colour>* to create a coloured line.
Use syntax *!Arr:<RGB colour>* to create a coloured arrow.

Available complex functions :

[If, <condition>, <expression if true>, <expression if false>]

Evaluates *<condition>* then returns evaluated *<expression if true>* if condition returns *True* or any value but 0, or returns evaluated *<expression if false>* if condition returns *False* or 0.
Example : [if, X>5, Y+1, Y-1]
Note : expressions must not contain commas unless you get unpredictable result; if you need use one or more commas inside a text expression, call a variable that contains them.

[Choose, <condition>, <expression if 1>, <expression if 2>, …, <expression if n>]

Evaluates *<condition>* then returns evaluated expression whose position is evaluated *<condition>*. If lower than 1 or greater than the count of specified expressions, returns 0.
Note : same remark about the use of commas.

[Repeat, <variable>, <min>, <max>, <expression>]

Repeats the evaluation of *<expression>* for *<variable>* whose value varies between *<min>* and *<max>* by step of 1 or -1 (automatically deduced from *<min>* and *<max>* values).
Variable must be *%0*, *%1*, *%2*, …, or *%9* ; this variable can be used in *<expression>* and even in variables called by *<expression>*.
Examples :
*X = %1 * 10 # #* and *Y = [Repeat, %1, 1, 5, %self.X]* evaluates as *10 20 30 40 50.*
*X = %1 * 2 # #* and *Y = [Repeat, %1, 5, 1, %self.X]* evaluates as *10 8 6 4 2.*
*Y = [Repeat, %1, 1, 5, %1*2 # #]* evaluates as *2 4 6 8 10.*

Note : same remark about the use of commas. You can also use *%1p* that evaluates as *%1+1* or *%1m* that evaluates as *%1-1* inside variables called by *<expression>* but not inside *<expression>* itself.

[ExitRepeat]
Leaves most recent running repeat loop and returns an empty string or 0 depending on calling expression. If there is no loop currently running, the function does nothing.
Example : *X = [If, %1 > 5, [ExitRepeat], 10*%1 # #]* and *Y = [Repeat, %1, 1, 15, %self.X]* evaluates as *10 20 30 40 50* ; cases when *%1* equals 6 to 15 do not evaluate at all.

[DefArray, %<type><number>, <min>, <max> (, %<type><number>, <min>, <max>)]
Defines an array of values. Type is Lng (for Long), Sng (for Single) or Str (for String). Number is between 0 and 3 (you can define up to 4 arrays for each type). Min and max position expressions are evaluated then the array is defined accordingly. You can define more than one array at once. Returns nothing.
Example : *X = [DefArray, %Lng0, 1, 10]* defines an array of long values at position 1 to 10.
Y = [DefArray, %Lng0, 1, %self.Size] defines an array with computed max position.

[SetArray, %<type><number>, <position>, <value> (, %<type><number>, <position>, <value>)]
Stores evaluated *<value>* in specified array at specified *<position>*. You can store more than one value at once. Returns nothing.

Available constants :

Pi	Single	3.1415926
EmptyStr	String	"" (empty string)

Available pattern variables :

$Title	String	Current pattern title
$Author	String	Current pattern author
$Copyright	String	Current pattern copyright message
$Company	String	Current pattern author company
$Website	String	Current pattern author site
$Email	String	Current pattern mail address
$Width	Long	Current pattern width in grid squares
$Height	Long	Current pattern height in grid squares
$HRes	Long	Current pattern horizontal resolution in squares per inch
$VRes	Long	Current pattern vertical resolution in squares per inch
$Scale	Single	Current pattern scale factor (embroidery or quilting : 10, else :1)
$IsCrs	Long	Current pattern is cross stitch (0=false, -1=true)
$IsEmb	Long	Current pattern is embroidery (0=false, -1=true)
$IsQlt	Long	Current pattern is quilting (0=false, -1=true)
$IsKnt	Long	Current pattern is knitting (0=false, -1=true)
$IsLhk	Long	Current pattern is latch hook (0=false, -1=true)
$WhatIs	Long	Current pattern content type (1=cross stitch, 2=embroidery, 3=quilting, 4=knitting, 5=latch hook)
$WhatIsEx	Long	Same values plus 6=scrapbook if containing charms only
$EdTime	String	Current pattern elapsed editing time.

$PropPal:i;j;k		Current pattern palette property		
	Long	Count of palette entries	0; 0; 0	
	Long	Count of sub entries	ME; 0 ; 1	
	Long	Symbol number	ME; 0 ; 2	
	String	Symbol font	ME; 0 ; 3	(example : Arial)
	Long	Sub entry is defined	ME; SE; 0	(0=false, -1=true)
	Long	Sub entry type	ME; SE; 1	(0=Thread, 1=Bead)
	String	Sub entry range key	ME; SE; 2	(example : DMC)
	String	Sub entry range info	ME; SE; 3	(example : DMC Cotton)
	String	Sub entry number	ME; SE; 4	(example : 310)
	String	Sub entry description	ME; SE; 5	(example : Black)
	Long	Sub entry RGB colour	ME; SE; 6	
	Long	Sub entry strands count	ME; SE; 7	(0 to 6)

Note : The variable name is *$PropPal:* (2 points at end with no space). *i*, *j* and *k* are parameters separated with a semicolon ; they can be either *@section.variable*, *%self.variable* or *%0* to *%9*. *ME* stands for main palette entry position number (1 for 1st colour, 2 for 2nd and so on). *SE* stands for palette sub entry position number (between 1 and 4). Using complex function *Repeat* allows you to scan the whole palette.

Examples :

$PropPal: 0; 0; 0 returns the number of colours in the palette (palette entries).

$PropPal: 5; 0; 1 returns the number of defined sub entries in 5th palette entry.

$PropPal: 3; 2; 4 returns the colour number of 2nd sub entry in 3rd palette entry.

$PropCS:col;row		Current pattern cross stitch property	
	Long	Cross stitch (column, row) type table entry pointer	

Note : The variable name is *$PropCS:* (2 points at end with no space). *col* and *row* are parameters separated with a semicolon ; they can be either *@section.variable*, *%self.variable* or *%0* to *%9*. Using complex function *Repeat* allows you to scan the whole pattern.

$PropCT:i;j		Current pattern cross stitch type property	
	Long	Count of entries	0; 0
	Long	Top left quarter stitch palette pointer (*TLQ*)	TE; 1
	Long	Top right quarter stitch palette pointer (*TRQ*)	TE; 2
	Long	Bottom left quarter stitch palette pointer (*BLQ*)	TE; 3
	Long	Bottom right quarter stitch palette pointer (*BRQ*)	TE; 4
	Long	Petite stitch flag	TE; 5

Note : The variable name is *$PropCT:* (2 points at end with no space). *I* and *j* are parameters separated with a semicolon ; they can be either *@section.variable*, *%self.variable* or *%0* to *%9*. *TE* stands for table entry position number (1 for 1st entry, 2 for 2nd and so on). Petite stitch flag is bit-coded (1 for *TLQ*, 2 for *TRQ*, 4 for *BLQ*, 8 for *BRQ*) ; example : 5 means *TLQ* and *BLQ* are patite stitches while *TRQ* and *BRQ* are quarter stitches. Using complex function *Repeat* allows you to scan the whole table.

$PropBS:i;j		Current pattern backstitch property	
	Long	Count of backstitches	0; 0
	Long	Palette pointer	TE; 0
	Long	Start X	TE; 1
	Long	Start Y	TE; 2
	Long	End X	TE; 3

	Long	End Y	TE; 4
$PropFK:i;j		Current pattern French knot property	
	Long	Count of French knots	0; 0
	Long	Palette pointer	TE; 0
	Long	X	TE; 1
	Long	Y	TE; 2
$PropBD:i;j		Current pattern bead property	
	Long	Count of beads	0; 0
	Long	Palette pointer	TE; 0
	Long	X	TE; 1
	Long	Y	TE; 2
$PropES:i;j		Current pattern embroidery stitch property	
	Long	Count of embroidery stitches	0; 0
	Long	Palette pointer	TE; 0
	Long	X	TE; 1
	Long	Y	TE; 2
$PropQS:i;j		Current pattern quilting stitch property	
	Long	Count of quilting stitches	0; 0
	Long	X	TE; 1
	Long	Y	TE; 2
$PropObj:i;j		Current pattern object property	
	Long	Count of objects	0; 0
	Long	Object type	TE; 0
	Long	Left	TE; 1
	Long	Top	TE; 2
	Long	Width	TE; 3
	Long	Height	TE; 4
	String	Name (Provider item name)	TE; 5
	String	Reference (Provider item number)	TE; 6
	String	Description (Provider item type)	TE; 7
	String	Provider name	TE; 8
$LngStr:ID;i;j;k;l		String from current language file	

Note : *ID* is mandatory, this is the string identification number ; *i, j, k* and *l* are optional help string parameters *%1*, *%2* ; *%3* and *%4* recpectively.

Available processing variables :

%0 to *%9*	Long or Single	Variable of a *Repeat* function
%Lng<number>:<position>	Long	Item in an array of long values
%Sng<number>:<position>	Long	Item in an array of single values
%Str<number>:<position>	Long	Item in an array of string values

Extended data variables :

The general syntax is *@SectionName.VariableName* but you can use *%self.VariableName* to refer to the current section name ; for instance, *%self.Area* evaluates as *MoreInfo.Area.* Later versions may contain more sections for additional features ; at that time, the general syntax will be useful. Obviously, if specified variable is not found in specified section, it will be evaluated to zero.

More examples :

1+2	returns 3
*(1+2)*3*	returns 9
Sqr(3)	returns 1.7320508
Sqr(2+1)	returns 1.7320508
(2.5+1.5)(3-1)*	returns 8
*2.54*$Width/$HRes*	returns pattern width in centimeters
2@Size.Width*	returns 2 times the value stored in section *Size*, variable *Width*.

Appendix C. Information files format (.mcmia)

Encoding :

Statement	***Meaning***
myriaCross editor	*Identifier*
Section=MoreInfo	*Section name, always MoreInfo*
Version=1	*Format version*
Count=<count-of-variables>	*Count of following variable definitions*
For *Count* variables :	
Lng:<variable-name>=<value>	*Long*
Or Lng/P:<variable-name>=<value>	*Long Printable*
Or Sng:<variable-name>=<value>	*Single*
Or Sng/P:<variable-name>=<value>	*Single Printable*
Or Str:<variable-name>=<value>	*String*
Or Str/E:<variable-name>=<value>	*String Must be evaluated*
Or Str/P:<variable-name>=<value>	*String Printable*
Or Str/EP:<variable-name>=<value>	*String Printable Must be evaluated*

Note : Long and single variables need not be evaluated, thus, Lng/E and Sng/E will do nothing more though recognized by the program. For multiline string values, use sequence \0D\0A to indicate a new line ; example : *Str:MyString=Here is a\0D\0Amultiline\0D\0Astring.* You can add remark lines but only after Count statement line ; to do so, use ‘ as first character ; example : *‘Comment line*

Localising your variables :

The program prints all the variables you set as printable using the variable name as printed information name. If you want your variables to be printable in more than one language, here is how to proceed :

Consider this sample variable :

Str/EP:Area=$Width*$Height # stitches#

In order to print in both English and French, define 2 variables as follows :

Str/EP:Area.en=$Width*$Height # stitches# (English variable)

Str/EP:Area.fr|Surface=$Width*$Height # points# (French variable)

All variables set as printable having no language specified will be added to printouts.

Variables having specified language matching current interface language will be added if found; if not found, variables having specified language matching replacement language declared in current language file will be added. Variables for other languages will be ignored.

Note : you can also use *Variable Name|Printed name* with non localised variables.

myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

Sample file :

myriaCross editor
Section=MoreInfo
Version=1
Count=4
Str/E:AreaSt=$Width*$Height
Str/E:AreaUS=($Width/$HRes)($Height/$VRes)*
Str/E:AreaMetric=(2.54^2)%self.AreaUS*
Str/EP:Area=%self.AreaSt # stitches (# dp2(%self.AreaUS) # sq in or # dp1(%self.AreaMetric) # sq cm)#

Preset information files :

Filename *(in My documents\myriaCross editor\MoreInfo)*	English variable	French variable
en-fr-Property-Area	*Area*	*Surface*
en-fr-Property-Diagonal	*Diagonal*	*Diagonale*
en-fr-Property-Orientation	*Orientation*	*Orientation*
en-fr-Property-Perimeter	*Perimeter*	*Périmètre*
en-fr-Properties-Orientation-Diagonal-Perimeter-Area	*All at once*	*All at once*
en-fr-Property-Objects List	*Objects*	*Objets*
en-fr-Property-Stitch Types	*Types* *Stitches*	*Types* *Points*
br-de-en-fr-nl-Property-Edit Time	*Edited for*	*Edité durant*
br-de-en-fr-nl-Property-Content	*Content*	*Contenu*
br-de-en-fr-nl-Property-Colours	*Colours*	*Couleurs*
br-de-en-fr-nl-Properties-Backstitches-FrenchKnots-Beads	*Backstitches* *French knots* *Beads*	*Piqûres* *Nœuds* *Perles*
br-de-en-fr-nl-Property-Alternate Sizes	*Sizes*	*Tailles*

Area *displays pattern area in stitches, square inches and square centimeters.*
Diagonal *displays diagonal size in stitches, inches and centimeters plus angle in degrees.*
Orientation *displays either « Portrait », « Landscape » or « Square ».*
Perimeter *displays pattern perimeter in stitches, inches and centimeters .*
Objects *displays the corner, size and name of every object in the pattern.*
Types *displays either the count of cross stitch quarter stitches combinations.*
Stitches *displays quarter stitches combinations.*
Edited for *displays the total elapsed editing time.*
Content *displays either « Cross stitch », « Embroidery », « Quilting », « Knitting » or « Latch Hook ».*
Colours *displays the list of colours in the palette.*
Backstitches, ***French knots*** *and* ***Beads*** *display a count of respective items by colour.*
Sizes *displays alternate sizes for current cross stitch pattern.*

A preset file explained :

We will now explain the content of preset file ***en-fr-Property-Area*** :

1 myriaCross editor
2 Section=MoreInfo
3 Version=1
4 Count=7

```
 5 'Compute Required Values
 6 Str/E:AreaSt=$Width*$Height
 7 Str/E:AreaIn=(($Scale*$Width)/$HRes)*(($Scale*$Height)/$VRes)
 8 Str/E:AreaCm=(2.54^2)*%self.AreaIn
 9 Str/E:AreaU.en=[if,$IsEmb or $IsQlt,#sq mm#,#stitches#]
10 Str/E:AreaU.fr=[if,$IsEmb or $IsQlt,#mm²#,#points#]
11 'Define Printable Variables
12 Str/EP:Area.en=%self.AreaSt # # %self.AreaU.en # (# dp2(%self.AreaIn) # sq in or # dp1(%self.AreaCm) # sq cm)#
13 Str/EP:Area.fr|Surface=%self.AreaSt # # %self.AreaU.fr # (# dp1(%self.AreaCm) # cm² ou # dp2(%self.AreaIn) # "²)#
```

Line 1 is the file content identifier ; entering something else will make the program fail loading the file.
Line 2 is the data section name ; entering something else will make the program fail loading the file.
Line 3 is the file format version of the content ; entering something else will make the program fail loading the file ; next versions may enhance the format and change the version number.
Line 4 is the count of variables in the file ; comment lines are not taken into account.
Line 5 is a comment line ; it makes the file a little bit more readable ; add as many comment lines as you wish, this is for your own comfort. You cannot include any comment line before 4th line though. The program currently includes no comment lines when saving.
Line 6 computes the pattern area in stitches :
$Width is the count of pattern columns
$Height is the count of pattern rows.
Line 7 computes the actual pattern area in square inches :
$HRes is the count of stitches per inch in the column direction
$VRes is the count of stitches per inch in the row direction
$Scale is a scale factor required for compatibility with all pattern content types :
Cross stitch, knitting, latch hook : 1 stitch per grid square
Embroidery, quilting : 10 stitches per grid square
($Scale*$Width)/$HRes is the pattern width in inches
($Scale*$Height)/$VRes is the pattern height in inches
Line 8 computes the actual pattern area in square centimeters :
Knowing that a length in centimeters equals 2.54 times a length in inches
2.54^2 means 2.54*2.54 ; both width and height must by multiplied by 2.54
To avoid repeating code, the variable uses the result of variable *%self.AreaIn* that means variable *AreaIn* in the same section.
Lines 9 and 10 define units to display the pattern area in stitches depending on the pattern content : embroidery or quilting : square millimeters, otherwise : stitches
Line 9 defines units for the English version
Line 10 defines units for the French version
Line 11 is another comment line
Lines 12 and 13 define printable variables blending texts and numbers.

Line 12 defines printable variable *Area* for the English version

%self.AreaSt	Number : area in stitches
# #	Text : 1 space
%self.AreaU.en	Unit text for area in stitches
# (#	Space plus text
dp2(%self.AreaIn)	Number : area in inches with 2 decimal places
# sq in or #	Space, unit text, space
dp1(%self.AreaCm)	Number : area in centimeters with 1 decimal place
# sq cm)#	Space, unit text

Line 13 defines printable variable *Surface* for the French version

%self.AreaSt	Number : area in stitches
# #	Text : 1 space
%self.AreaU.fr	Unit text for area in stitches
# (#	Space plus text
dp2(%self.AreaIn)	Number : area in inches with 2 decimal places
# cm² ou #	Space, unit text, space
dp1(%self.AreaCm)	Number : area in centimeters with 1 decimal place
# "²)#	Space, unit text

Private named sections :

Our chess and puzzle games contain private sections to handle specific behaviours. Sections *Object()* are editable from menu *Edit/Objects properties* ; other sections are not available for edition but could be in future versions of the program.

Object(1)	**Specific data for main object ➔ pattern**
Lock	Pattern locks (see below)
V3dBi	Regions data for view as biscornu
V3dCu	Regions data for view as cube
V3dCy	Regions data for view as cylinder
V3dPe	Regions data for view as perspective
V3dSp	Regions data for view as sphere

Object(<handle>)	**Specific data for an object by handle**
X, Y	Object startup position
X1, Y1	Alternate object position
Lock	Object locks (see below)

ScreenButton<#>	**Definition of a pattern button**
Left, Top	Position
Width, Height	Size
Picture	Image
Shape	Shape (see below)
Action	Action on click (see below)
[X1, Y1, X2, Y2]	Action range
Tooltip.<lang>	Tooltip text for language *lang*

myriaCross editor – lesson 17 : Defining more pattern information and evaluating mathematical expressions

Lock bit values :

1	Disallow object edition (no toolbar, no edit menu, no context menu) And automatically unselect object on mouse up
2	Disallow object selection (no edit, no move, fully locked)
4	Disallow object outline generation
8	Disallow object move
16	Disallow object outline smoothing
32	Disallow object shadow generation
1 * 65536	Disallow pattern resize
2 * 65536	Disallow pattern palette standardization
Other	Currently unused

Shape codes :

0	Rectangle
1	Rounded rectangle
2	Ellipse
3	Handle transparent colour (0xFF00FF)
4	Keep non-transparent areas only

Action codes :

1	Change lock for all objects
10	Memorize all objects position as X, Y
11	Memorize all objects position as X1, Y1
20	Rotate objects found in range X1, Y1, X2, Y2
30	Restore all objects position from X, Y
31	Restore all objects position from X1, Y1
40	Set all objects random position using X, Y