CodeSheet Manual

by José Luis Dorado Ladera [1]

January 9, 2019

[1]odarod86j@gmail.com

Contents

Chapter 1

The sheet

1.1 Extensions and folders

- Save your CodeSheet documents with the extension *.csheet* and your tables with the extension .ctable. CodeSheet software will ignore other documents.
- You can organize your documents using subfolders if you want.
- To open your sheet, select the root folder which contains the files of your character.

1.2 Attributes

- An attribute is a numerical variable of your character.
- We set the value of an attribute using declarations.
- We strongly recommend using only letters of the alphabet and the _ character for the attributes' names. You can also use numbers at the end. For example, attribute1 or attribute2. In any case, the first letter of the attribute MUST be a letter of the alphabet. An attribute name can NOT have spaces.

1.2.1 Declarations

- An attribute declaration is one line of a .csheet document with a specific syntax. It is used to modify the value of an attribute. We also use declarations in macros, this declarations are called non-attribute declarations.
- An attribute declaration has two parts. The first part provides the name of the attribute you want to modify. The second part is the modification and consists of a set of parameters.
- The two parts of an attribute declaration are divided by the ':' character.
- It is NOT possible to write two or more attribute declarations in only one line. One attribute declaration by line.

- Sometimes we use the expression 'declaration' to refer only to the mentioned second part, i.e., the expression after the ':' character. In this case is a non-attribute declaration and is usually used in macros.

Examples:

- strenght: 29 + 5 + 7
- dex: 29 + dex_enhancement - dex_penalties
- wisdom_bonus: \table(wisdom, table_abilitymodifiers) - wisdom_bonus_penalties

Suppose that we have the following declarations:

- attribute1: 5
- attribute2: attribute1 + 2
- attribute3: 3
- attribute4: attribute2 + attribute3

then the values of these attributes are:

- attribute1 = 5
- attribute2 = 7
- attribute3 = 3
- attribute4 = 10

Bucles of declarations

You should take care because you can make a bucle. In this case, you will not be able to use the attributes in the bucle. For example, the next declarations make a bucle:

- attribute1: attribute2
- attribute2: attribute3
- attribute3: attribute1

Multiple declarations of an attribute

We can have more than one declaration of an attribute. In this case, the value of each declaration is added to calculate the final value of the attribute. This declarations of the same attribute could be in different *.csheet* files.

Examples: if we have the next declarations:

- attribute1: attribute2 + 5

- attribute1: 3 + 4
- attribute1: attribute2
- attribute2: 2

then the value of the attribute1 is 16. However, this behavior can be changed using the parameters \min() and \max().

You can find more information in the **Parameters** chapter.

1.3 Tables

You can think that a table is as a function. The table gets an *input* and generates an *output*. To use a table you need to call this table as a parameter.

Example: baseAttack : \table(level , baseAttackMedium)
In this example, the attribute baseAttack is equal to the *output* of the baseAttackMedium table according to the level *input*.

You can find more information in the **Parameters** chapter.

1.4 Macros

A macro is a set of instructions. With each instruction you make one action with the software CodeSheet. You can create variables, call this variables or attributes of your sheet, print something in the display, execute another macro, use the elemental structures of a programing language such as if, for, while or arrays, enable and disable *.csheet* files, and much more.

You can find more information in the **Macros** chapter.

1.5 Enable and disable *.csheet* files

A *.csheet* file can be enabled or disabled. If a *.csheet* file is disabled, then the declarations and macro codes declared in this file are not effective.

Example: Suppose that we have this three files:

---

base.csheet

```
level: 2
STR: 18
STRbonus: (STR−10)/t:2
baseAttack: level
```

---

sword2.csheet

```
STR: 2
swordAttack: baseAttack + STRbonus + 2
swordDamage: 2d6 + STRbonus + 2
\MACRO: swordAtt
  ?swordAttack
  ?swordDamage
\ENDMACRO
```

---

strengthBeltPlus4.csheet

```
STR: 4
```

---

then the values of the attributes and the macro actions are:

variables↓ / enabled files→	**only *base.csheet***	***base.csheet* and *sword.csheet***	**All are enabled**
level	2	2	2
STR	18	20	24
STRbonus	4	5	7
baseAttack	2	2	2
swordAttack	0	9	11
swordDamage	0	2d6+7	2d6+9
swordAtt action	nothing happens	show attribute tables	show attribute tables

Read the next chapters to understand the details.

1.5.1 \ALWAYS ENABLED

You can force a file to be always enabled. The file will always remain enabled even though the user executes a command to disable it. To do that you must include this line in the file:

\ALWAYS ENABLED

This line can not be inside a macro's declaration.

1.6 Comment lines

You can add comment lines in your *.csheet* files. If a line starts with the character #, then the CodeSheet software is going to ignore this line. You can also use these comments in your macros.

Example:

```
# character level
level: 2

# hit points
hp: (CONbonus + 5)*level
```

```
#### FULL ATTACK MACRO ####
\MACRO: fullAttack
  ## sword attack ##
  execute swordAttack
  # if it is a critical hit
  if die >= 19 \then
    ## critical sword attack ##
    execute criticalSwordAttack
  ## bite attack ##
  execute biteAttack
  # if it is a critical hit
  if die = 20 \then
    ## critical bite attack ##
    execute criticalBiteAttack
  \end
\ENDMACRO
###########################
```

1.7 Broken lines

In a *.csheet* file you can break a line in two or more lines to make easier write and understand the code. To do that you have to add '\\' at the end of a line. When you add '\\' at the end of a line, this line and the next line are automatically joined, ignoring the spaces in the second line.

Example: These lines in a *.csheet* file:

```
attack : 5 + \windowOption(First Bonus,\\
                              plus one   :1,\\
                              plus two   :2,\\
                              plus three :3) \\
           + level
\MACRO: printAttribute
  to print: This is \\
            the attack's value = $$attribute$$ <br>
  print
\ENDMACRO
```

are completely equivalent to these ones:

```
attack : 5 + \windowOption(First Bonus,plus one   :1,plus two   :2,plus three :3) + level
\MACRO: printAttribute
  to print: This is the attack's value = $$attribute$$ <br>
  print
\ENDMACRO
```

Chapter 2

Parameters

In this section we are going to revise the different parameters you can use to modify declarations. We separate the parameters using the '+' character, or the '-' character if we want to add the negative value of the following parameter.

We can also make products and divisions, and use parenthesis. This information is shown at the end of this section.

2.1 Attributes

We can use other attributes as a parameter.

Example: `attribute1: attribute2 + 3`

2.2 Numbers

We can use numerical values as a parameter.

Example: `attribute: 3 + 2 + 43 + 35`

2.3 Dice rolls

We can roll dice, i.e., we can generate random numbers as we do when we roll dice. Then, the value of a dice roll parameter is the value of a random number. To do that, we use the character '%'. You must indicate the number of dice you want to roll and how many sides have these dice with the next syntax:

%[NUMBER OF DICE]d[NUMBER OF SIDES]

Examples:

- `%2d6`
- `%1d20`

- %20d6
- %3d7

You can also use a declaration to indicate the number of dice and sides. In this case you must use brackets:

%([DECLARATION])d([DECLARATION])

Example: Suppose that we have the next attributes' declarations:

- level : 10
- charismaBonus : 8
- attack : %1d20 + charismaBonus
- fireball : %(level)d6
- superFireball : %(level + 5)d6 + level + charismaBonus
- tremendousFireball : %(level+charismaBonus+2)d(charismaBonus+2) + level

then the attributes' values are:

- level = 10
- charismaBonus = 8
- attack = 1d20 + 8
- fireball = 10d6
- superFireball = 15d6 + 18
- tremendousFireball = 20d10 + 10

where 1d20, 10d6, 15d6 and 20d10 are random numbers generated with the same probability distribution as the corresponding dice rolls.

2.3.1 Dice rolls' messages

When you generate a random number using a dice roll parameter (i.e. using the '%' character), the system print a message writing the result of this dice roll. This message is written by the system and allows you to prove that you has generated a random number according to a dice roll and what has been the result.

2.3.2 Dice rolls' labels

If you want that the system print a label with a dice roll result to identify it easily, you must use this syntax:

%[DICE ROLL LABEL]:[NUMBER OF DICE]d[NUMBER OF SIDES]

Example: Suppose that we have the next attributes' declarations:

- level : 10
- charismaBonus : 8
- numberRays : %1d4
- ray_attack_roll : %ray attack : 1d20 + charismaBonus
- ray_damage_roll : %ray damage : (level)d6

then when we call this attributes, the system print the following messages (remember that the results are random, these are only examples):

- numberRays:

  1d4 -> 2
- ray_attack_roll:

  ray attack : 1d20 -> 14
- ray_damage_roll:

  ray attack : 10d6 -> 33

2.3.3 What to do to avoid the system printing of dice rolls' results

If you want to generate a random number using a dice roll parameter but you do not want that the system prints the result, you can use '%%' instead of '%'. This is useful, for example, if you are a Master in a rol play game. You may want to generate dice rolls, however, you do not probably want to show the results to your players.

Example: Suppose that we have the next attribute's declaration:

- ray : %%1d20 + 6

then when we call this attribute, the system do NOT print the dice roll result.

2.4 Variables

We can use the variables's values of the macro which solve a declaration. To do that, we use the next syntax:

![VARIABLE'S NAME]

It is unusual to use variables in an attribute's declaration, however, it is necessary for non-attribute declarations. For more information about variables and macros, go to the Macros' Chapter.

Example: The declaration

!bonus1 + !bonus2 + strenght + %2d6

is equal to the sum of the variables bonus1 and bonus2, the attribute strenght, and the dice result of 2d6.

2.5 \min() and \max()

There is the possibility to change the behavior of how we calculate an attribute's value when it is used as a parameter. Instead of adding each declaration's value to the final value of the attribute, we can find the maximum or minimum declaration's value. To do that you must use the next syntax:

\min([ATTRIBUTE'S NAME])

or

\max([ATTRIBUTE'S NAME])

Example: Suppose that we have the next attributes' declarations:

- attribute1 : attribute2 + 2
- attribute1 : 3
- attribute1 : 5
- attribute2 : 4
- attribute3 : \min(attribute1)
- attribute4 : \min(attribute1) + 10
- attribute5 : \max(attribute1)
- attribute6 : \max(attribute1) + \max(attribute1)

then the values of these attributes are:

- attribute1 = 14
- attribute2 = 4
- attribute3 = 3
- attribute4 = 13
- attribute5 = 6

- attribute6 = 9

Notice that \min(attribute1) is equal to 3 because 3 is the minimum of the attribute1 declaration values. And \max(attribute1) is 6 because 6 is the maximum (that is the value of the declaration: attribute2 + 2).

2.6 Macro's return

We can execute a macro and use the exit instruction (3.8.1) to return a value. To do that, we use the '&' character followed by the macro's name. There are more information in the section 'Macros'.

Example: attribute: 5 + &macrosname + 7

2.6.1 Set of instructions' return

We can execute a set of instructions and get a returned value via the exit instruction as we do with the **Macro's return**. The syntax to do that is:

&([INSTRUCTIONS])

When we write a macro, we separate the instructions by lines, i.e., one instruction one line. However, in this case we have to put all instructions in only one line. To do that we separate the instructions using the ';' character.

Example:

min2d20: &(new array arr; array-add %1d20 \to arr; array-add %1d20 \to arr; min arr \to minValue; exit !minValue)

Example:

```
min5d6Plus3: &( \\
                 new array arr; \\
                 for i \from 1 \to 5 \do; \\
                   array-add %1d6 \to arr; \\
                 \end; \\
                 min arr \to minValue; \\
                 exit !minValue \\
              ) \\
              + 3
```

2.7 \windowInput()

The value of this parameter change depending on the input the user write each time that it is called. The user can write this input in a window which opens each time that the parameter is called. The syntax you must use is:

\windowInput([MESSAGE TO SHOW])

Example: Suppose that we have the next attributes' declarations:

- attribute1 : attribute2 + 2
- attribute2 : 3
- attribute3 : attribute1 + \windowInput(bonus?) + 1

then when the attribute3 is called, a window with the message "bonus?" opens. In this window the user can write a declaration, for example if the user write

- 3, then the value of the windowInput parameter is 3 and the value of attribute3 is 8.
- attribute2 + 5, then the value of the windowInput parameter is 8 and the value of attribute3 is 14.
- attribute2 + 6 + 2, then the value of the windowInput parameter is 11 and the value of attribute3 is 17.

2.8 \windowOption()

The value of this parameter also change depending on the user decision each time that it is called. In this case the user can not write the input directly, instead of that the user choose between different options. The syntax you must use is:

\windowOption([MESSAGETOSHOW],[OPTION1]:[DECLARATION1],[OPTION2]:[DECLARATION2],...)

Examples:

- Suppose that the next parameter is called:

  \windowOption(flank bonus?, yes : 2, no : 0)

  then a window with the message "flank bonus" opens. The user can choose between two buttons, "yes" and "no". If the user chooses "yes" then the value of this parameter is 2. If the user chooses "no" then the value is 0.

- Suppose that the next parameter is called:

  \windowOption(which saving throw?, fortitude : fort, reflex : ref + 2, will : will)

  then a window with the message "which saving throw?" opens. The user can choose between three buttons: "fortitude", "reflex" and "will". If the user chooses

  – "fortitude" then the value of this parameter is the value of the declaration fort, i.e., the attribute fort.
  – "ref" then the value is the value of the declaration ref + 2.
  – "will" then the value is the value of the declaration will.

2.9 \windowOptionList()

This parameter is almost the same than the parameter \windowOptionList(). There is only one difference. When the user have to choose between the different options, instead of buttons, this options are shown in a list. The syntax is:

\windowOptionList([MESSAGE],[OPTION1]:[DECLARATION1],[OPTION2]:[DECLARATION2],...)

2.10 \if()()

We use this parameter when we want to add a value only in some situations. The syntax is:

\if([CONDITION])([DECLARATION])

If the [CONDITION] is satisfied when this parameter is called, then the value of this paramter is the value of [DECLARATION]. If the [CONDITION] is not satisfied, then the value is 0.

2.10.1 Condition's syntax

- **Logical AND** : &
- **Logical OR** : \OR
- **Logical Negation** : ¬
- **Relational Operators** : = < <= > >=
- **File Checker Operator** : isEnable([FILES PATTERN])

It is NOT possible to use parenthesis in a condition's syntax. It is necessary to develop the condition's expression in order to avoid the parenthesis. For exemple,

¬(A=B & C=D)

is NOT possible. However, we can use the equivalent expression

¬A=B \OR ¬C=D

where A, B, C and D are declarations.

isEnable([FILES PATTERN])

This condition is *TRUE* if the string [FILES PATTERN] is a substring of an enabled file. For example, isEnable(weapon) is *TRUE* if one or more of these files

- *weapon.sword.csheet*

- *weapon.katana.csheet*
- *axe.weapon.csheet*
- *weaponsword.csheet*
- *axeweapon.csheet*

or any other file which contains the string *weapon* is enabled.

Examples:

- strengthBonus : 5 + \if(isEnable(rage.csheet))(2)

  The value of strengthBonus is 7 if the file *rage.csheet* (or another file whose name contains the string *rage.csheet*) is enable, and 5 if there is no matched file enable.

- strengthBonus : 5 + \if(isEnable(rage.csheet) & level > 5)(2)

  The value of strengthBonus is 7 if the file *rage.csheet* (or another file whose name contains the string *rage.csheet*) is enable and the attribute level is greater than 5. However, the value is 5 if there is no matched file enable or the attribute level is less or equal to 5.

- \if(str = dex & ¬ char = int \OR sab+5 <= con) (level + 10)

  Supponse that level, str, dex, con, sab, int and char are attributes. Then, the value of this parameter is equal to the value of level + 10 if the condition str = dex & ¬ char = int \OR sab <= con is satisfied, i.e., if str is equal to dex and char is different to int; or sab+5 is less or equal to con. If the condition is not satisfied, then the value is 0.

2.11 Tables

We can use tables as a parameter. A table has an *input* and generates an *output*. Lets see an example:

Example:

0	:	2
1	:	4
2	:	7
3	:	8

This table generates the *output* 2 for the *input* 0. In the same way, it generates the *outputs* 4, 7 and 8 for the *inputs* 1, 2 and 3 respectively. For any *input* different to 0, 1, 2 and 3 the *output* is 0.

A table is composed by different lines. Each line has the next syntax:

[CONDITION] : [OUTPUT]

When we call a table, we have to specify an *input*. The *output* of the table is given by the first line that has a fulfilled condition according to the *input*. If there are no fulfilled condition in the table, then the *output* is 0.

2.11.1 Table's conditions

In the first table example we used only one type of condition. This was

if the *input* is **equal** to X, then the *output* is Y

However, there are more possibilities. Lets see the next example:

Example:

<=0	:	2
>0	:	4

The *output* of this table is `2` if the *input* is equal or less than `0`, and `4` if the *input* is greater than `0`.

We can also use the & character as a logical AND.

Example:

<=0			:	−1
0			:	0
>0	&	<5	:	2
>=5	&	<10	:	4
>=10	&	<15	:	6
>=15	&	<20	:	8
=20			:	10

Notice that to use the operator **equal** we can use the = character but it is not necessary. Then, the last table is equivalent to the next one:

<=0			:	−1
0			:	0
>0	&	<5	:	2
>=5	&	<10	:	4
>=10	&	<15	:	6
>=15	&	<20	:	8
20			:	10

2.11.2 How to write a table in your sheet

There are two possibilities to create a table.

Create a .ctable file

We can create a plain text file with the extension `.ctable`. Inside this file we have to write the table. If the file has this name:

`tablename.ctable`

then `tablename` is the name of the table. We have to use this name to call the table.

Table declaration inside a .csheet file

We can declare a table inside a .csheet file. This is the syntax:

```
\TABLE: [TABLE'S NAME]
[TABLE'S LINES]
\ENDTABLE
```

Example:

```
\TABLE: savesSlow
   1 & 2        : 0
  >=3  & <=5    : 1
  >=6  & <=8    : 2
  >=9  & <=11   : 3
  >=12 & <=14   : 4
  >=15 & <=17   : 5
  >=18 & <=20   : 6
\ENDTABLE
```

2.11.3 How to call a table as a parameter

In order to call a table as a parameter, you must use this syntax:

```
\table( [INPUT DECLARATION] , [TABLE'S NAME] )
```

Example: Suppose that we have the next *.csheet* file:

```
level : 6

fort  : \table(level, savesFast) + 2
ref   : \table(level, savesSlow)
will  : \table( level + 3 , savesSlow )

\TABLE: savesSlow
   1 & 2        : 0
  >=3  & <=5    : 1
  >=6  & <=8    : 2
  >=9  & <=11   : 3
  >=12 & <=14   : 4
  >=15 & <=17   : 5
  >=18 & <=20   : 6
\ENDTABLE

\TABLE: savesFast
   1          : 2
   2  & 3     : 3
   4  & 5     : 4
   6  & 7     : 5
   8  & 9     : 6
   10 & 11    : 7
  12 & 13 : 8
  14 & 15 : 9
  16 & 17 : 10
  18 & 19 : 11
  20      : 12
\ENDTABLE
```

then, using only this *.csheet* file, the attributes' values are:

- fort = 7
- ref = 2
- will = 3

2.12 Parameter's options

2.12.1 Create a variable with parameter's value

We can create a variable in the current macro with the value of a parameter. The syntax to do that is:

[PARAMETER] {[VARIABLE'S NAME]}

Example:

```
attack: %1d20{die} + meleeBonus
```

then when we call the attack attribute it is going to be created a variable with the die and value equal to the result of the d20 rolled.

Example:

```
swordAttack: %1d20{die} + meleeBonus
swordDamage: %1d8 + 5
swordCritDamage: %2d8 + 10
\MACRO: sword_Attack
  ?swordAttack
  if !die >= 19 \then
    ?swordCritDamage
    exit
  \end
  ?swordDamage
\ENDMACRO
```

2.12.2 Attribute's tables

We can make attributes's tables. To do that first we can write labels in the parameters which define this attribute.

Paramenter's labels

To assign a label to a parameter we use this syntax:

[PARAMETER] [[LABEL]]

Example:

```
attack: %1d20[die] + 20[melee bonus] + 2*4*3[mega bonus]
```

then the attack's table is:

attack	
die	1d20 result
melee bonus	20
mega bonus	24
TOTAL	the total

Example:

```
damage: %2d6[dice] + 5[strengh] + \windowOption(extra bonus?)[extra bonus]
```

then the attack's table is:

damage	
dice	2d6 result
strengh	5
extra bonus	user's iput
TOTAL	the total

Attribute labels

When we make an attribute table, we can show a name for the attribute different to the name we use in the sheet. To do that we must include a line with this syntax in a *.csheet* file:

@[ATTRIBUTE'S NAME]:[LABEL]

Example:

```
swordAtt: %1d20[die] + 10[melee bonus]
@swordAtt:Attack with short sword
```

the swordAtt table is:

Attack with short sword	
die	1d20 result
melee bonus	10
TOTAL	the total

Using ? to make tables recursively

When we make a attribute's table, we can also write the tables of some attribute parameters. To do that we use the ? at the begining of the attribute parameter.

Example:

```
lvl: 3
strBonus: 5
magicBonus: 3
meleeBonus: strBonus[strengh] + lvl[level]
attack: %1d20[die] + ?meleeBonus[melee bonus] + magicBonus[magic bonus]
```

when we call the attack's table, the system is going to write:

meleeBonus	
strengh	5
level	3
TOTAL	8

attack	
die	1d20 result
melee bonus	8
magic bonus	3
TOTAL	the total

Calling an attribute's table

To call and write in the display an attribute's table you can use the CodeSheet interface making double-click in the attribute's name or you can use the table macro instruction (3.2.8).

2.12.3 Labels and variable creation. Both options in the same parameter

If you want to use both parameter's options, you have to specify first the label and at the end the variable creation. The syntax would be:

[PARAMETER] [[LABEL]] {[VARIABLE'S NAME]}

If you invert the order, the options are not going to work.

Examples:

```
attack: %1d20[die]{die} + 20[melee bonus] + 2*4*3[mega bonus]
damage: %2d6[dice]{dice} + 5[strengh] + \windowOption(extra bonus?)[extra bonus]
```

2.13 Multiplications and divisions

When we make a declaration we can do multiplications and divisions. The characters you must use are the usuals * and /.

Example: Suppose that we have the next declarations:

- attribute1 : 5 + 5
- attribute2 : attribute1/3
- attribute3 : 2*3/10

then the attributes' values are:

- attribute1 = 10
- attribute2 = 3.333333
- attribute2 = 0.6

We can also truncate or round the result of a multiplication or division using the next syntax:

- Round Product:

  [FIRST FACTOR]*r:[SECOND FACTOR]

- Truncate Product:

  [FIRST FACTOR]*t:[SECOND FACTOR]

- Round Quotient:

  [NUMERATOR]/r:[DENOMINATOR]

- Truncate Quotient:

  [NUMERATOR]/t:[DENOMINATOR]

Examples:

- 2/3
  result = 0.666666
- 2/r:3
  result = 1
- 2/t:3
  result = 0
- 2*0.333333
  result = 0.6666667
- 2*r:0.333333
  result = 1
- 2*t:0.333333
  result = 0
- (attribute1 + attribute2)/t:3
- (attribute1 + attribute2)/t:attribute3
- (attribute1 + attribute2)*r:(attribute3 + attribute4)

2.13.1 Parenthesis

We recommend to use parenthesis when the operations' order is not clear. For example

$$8/4/2$$

could be

$$1 = (8/4)/2$$

or

$$4 = 8/(4/2).$$

Chapter 3

Macros

In this section we are going to revise the different instructions you can use in your macros. First, we are going to show the syntax to create a macro in a *.csheet* file:

\MACRO: [MACRO'S NAME]
[MACRO'S INSTRUCTIONS]
\ENDMACRO

You must write each instruction in one line (or more, some instructions need more than one line ,i.e., to print lines). It is NOT possible to write two or more instructions in one line.

3.1 Variables

There are three variable types:

- Numerical variables
- Text variables
- Numerical arrays
- Text arrays

3.1.1 Array Positions

Sometimes we want to use the value of an array's position. It is important to stress that the positions start with the 0 position.

Example: In this array:

arr = [7, 5, 6, 3]

7 is the 0 position, 5 is the 1st position, 6 is the 2nd position and 3 is the 3rd position.

3.1.2 Numerical variables

To assign a value to a numerical variable, we use this syntax:

!\[VARIABLE'S NAME\] = \[DECLARATION\]

Examples:

- !var = 5
- !var = attribute + !var2 + 5
- !attack = attack + !attackBonus

In the last example, we assign to the attack **variable** the value of the sum of the attack **attribute** and the attackBonus **variable**.

3.1.3 Numerical arrays

Create arrays

To create an array you have to use this syntax:

new array \[ARRAY'S NAME\]

Example: new array attacksDiceRolls

Add value

To add an element to you array you have to use this syntax:

array-add \[DECLARATION\] \to \[ARRAY'S NAME\]

Example: array-add %1d20 + 12 \to attacks

Delete array element

We can also delete an array element. To do that we use this syntax:

array-del \[POSITION'S DECLARATION\] \of \[ARRAY'S NAME\]

Examples:

- array-del 3 \of attacks
  Delete the 3rd position of the attacks array.
- array-del 3 + 2 \of attacks
  Delete the 5th position of the attacks array.
- array-del !positionToDelelete \of attacks
  Delete the attacks array's element according to the position given by the position-ToDelete variable.

Sort an array

We can sort an array using the next syntax:

sort [ARRAY'S NAME] \to [SORTED ARRAY'S NAME]

After this instruction a new array with the sorted values of the [ARRAY'S NAME] array is created. The name of this new array is [SORTED ARRAY'S NAME].

Example: Suppose that we run this instructions

```
new array arr
arr-add 7 \to arr
arr-add 5 \to arr
arr-add 6 \to arr
arr-add 3 \to arr
sort arr \to arrSorted
```

then the values of the arr array are:

arr = [7, 5, 6, 3]

and the values of the arrSorted are:

arrSorted = [3, 5, 6, 7]

Assign to a variable the sum, the minimum or the maximum value of a numerical array

The syntax to do that is:

sum [ARRAY'S NAME] \to [VARIABLE'S NAME]

min [ARRAY'S NAME] \to [VARIABLE'S NAME]

max [ARRAY'S NAME] \to [VARIABLE'S NAME]

Example: Suppose that we run this instructions

```
new array arr
arr-add 7 \to arr
arr-add 5 \to arr
arr-add 6 \to arr
arr-add 3 \to arr
sum arr \to arrSum
min arr \to arrMin
max arr \to arrMax
```

then the values of the arr array are:

arr = [7, 5, 6, 3]

and the variables' values are:

- arrSum = 21
- arrMin = 3
- arrMax = 7

In the case of min and max instruction we can assign the position in the array instead of the value. To do that we use this syntax:

min-pos [ARRAY'S NAME] \to [VARIABLE'S NAME]

max-pos [ARRAY'S NAME] \to [VARIABLE'S NAME]

Example: Suppose that we run this instructions

```
new array arr
arr-add 7 \to arr
arr-add 5 \to arr
arr-add 6 \to arr
arr-add 3 \to arr
min-pos arr \to arrMinPos
max-pos arr \to arrMaxPos
```

then the values of the arr array are:

arr = [7, 5, 6, 3]

and the variables' values are:

- arrMinPos = 3
- arrMaxPos = 0

Use array element as a variable

We can use array elements as we use variables. You have to use the next syntax:

!([POSITION'S DECLARATION])[ARRAY'S NAME]

Examples:

- !(2)attacks = 4
  Change the value of the 2nd position of the attacks array. This position MUST exists.
- !(!position)attacks = attribute + !var*2
- !(attribute + !var + 5)attacks = 34

- !var1 = !(3)var2 + !(!pos+1)var3

The position's values should be integers, i.e., numbers without decimals. If the system detect a decimal number, this number will be rounded.

Example: Suppose that we have the next array:

arr = [7, 5, 6, 3]

then

- !(0.1)arr is equal to 7
- !(0.9)arr is equal to 5
- !(1.499)arr is equal to 5
- !(2.5)arr is equal to 3

3.1.4 Text variables

To assign text to a text variable, we use this syntax:

@[TEXT VARIABLE'S NAME]=[TEXT]

Examples:

- @textVar=some text

3.1.5 Text arrays

Create text arrays

To create a text array you have to use this syntax:

new text-array [TEXT ARRAY'S NAME]

Example: new text-array attackNames

Add text element

To add an element to your text-array you have to use this syntax:

text-array-add [TEXT]\to [TEXT ARRAY'S NAME]

Example: Suppose that we want to create a text array named attackNames with three elements: "Right hand sword", "Left hand axe" and "Bite". Then we can use these instructions:

```
new text-array attackNames
text-array-add Right hand sword\to attackNames
text-array-add Left hand axe\to attackNames
text-array-add Bite\to attackNames
```

Delete text array element

We can also delete a text array element. To do that we use this syntax:

text-array-del [POSITION'S DECLARATION] \of [TEXT ARRAY'S NAME]

Examples:

- text-array-del 3 \of attackNames
  Delete the 3rd position of the attackNames text array.
- array-del 3 + 2 \of attackNames
  Delete the 5th position of the attackNames text array.
- array-del !positionToDelelete \of attackNames
  Delete the attackNames text array's element according to the position given by the positionToDelete variable.

Change an existing text array element

We can change the text stored in a text array element. To do that we use this syntax:

@([POSITION'S DECLARATION])[TEXT ARRAY'S NAME]=[TEXT]

Example: Suppose that we have this instructions:

```
new text-array strings
text-array-add this is the first string\to strings
text-array-add this is the second string\to strings
text-array-add this is the third string\to strings
```

then these are the elements of the strings array:

strings = ["this is the first string", "this is the second string", "this is the third string"]

Now, if we add these instructions:

```
!var=1
@(0)strings=this is the 1st str
@(3-2)strings=this is the 2nd str
@(!var+1)strings=this is the 3rd str
```

then these are the elements of the strings array:

strings = ["this is the 1st str", "this is the 2nd str", "this is the 3rd str"]

3.1.6 Assign to a variable the length of an array

If we want to know the number of elements that an array have, we can save it in a variable usign the next syntax:

length of [ARRAY'S NAME] \to [VARIABLE'S NAME]

Example: length of attacks \to attacksLen
After this instruction, the value of the variable attacksLen is equal to the number of elements of the attacks array.

If there are a numerical array and a text array with the same name, then the saved value is the length of the NUMERICAL array.

3.1.7 copy

We can make a copy of a variable (numerical variable, numerical array, text variable or text array). To do that we use this syntax:

copy [VARIABLE'S NAME] \in [NEW VARIABLE'S NAME]

Example: Suppose that we have this instructions

```
new array arr1
array-add 1 \to arr1
array-add 2 \to arr1
array-add 3 \to arr1
copy arr1 \in arr2
```

then we have created these two arrays:

arr1 = [1 ,2, 3]

arr2 = [1 ,2, 3]

If we have different types of variables with the same name, then all matched variables are copied.

Example: Suppose that we have this instructions

```
!var = 0
@var = some text
new array var
array-add 1 \to var
array-add 2 \to var
array-add 3 \to var
new text-array var
text-array-add textone \to var
text-array-add texttwo \to var
text-array-add textthree \to var
copy var \to var2
```

then we have created these variables:

- Numerical variables

var = 0

$$\mathsf{var2} = 0$$

- Text variables

$$\mathsf{var} = \text{“some text”}$$

$$\mathsf{var2} = \text{“some text”}$$

- Numerical arrays

$$\mathsf{var} = [1, 2, 3]$$

$$\mathsf{var2} = [1, 2, 3]$$

- Text arrays

$$\mathsf{var} = [\text{“textone”}, \text{“texttwo”}, \text{“textthree”}]$$

$$\mathsf{var2} = [\text{“textone”}, \text{“texttwo”}, \text{“textthree”}]$$

3.2 Print

We have some instructions to print in the CodeSheet's display.

3.2.1 HTML

The CodeSheet display interprets the HTML language. For example, if we print the next text:

```
<p><strong>text</strong></p>
<p><em>text</em></p>
<p><span style="text-decoration: underline;">text</span></p>
<p><span style="color: #ff0000;">text</span></p>
<p><strong><span style="color: #ff0000;">text</span></strong></p>
<p><em><span style="color: #ff0000;">text</span></em></p>
<p><span style="color: #0000ff;">text</span></p>
```

then the CodeSheet display is going to show something like:

text
text
text
text
text
text
text

The style for body and tables is preset.

You MUST NOT use <body> because the display could go crazy. When you send a message the system write your message between <body> and </body> in the HTML code:

<body> [YOUR MESSAGE] </body>

3.2.2 print buffer

When we use the "print" instruction, we send the text stored in the **print buffer** and then the display show a message according to this text. To store text in the **print buffer** we use the instruction "to print".

3.2.3 to print

To store text in the print buffer we use this syntax:

to print:[TEXT TO BUFFER]

Example: Suppose that we execute this instructions:

```
to print:first line ,
to print: more text in the first line <br>
to print: second line <br>
to print: third line <br>
```

then we have stored this text in the **print buffer**:

```
first line , more text in the first line <br> second line <br> third line <br>
```

3.2.4 print

With the print instruction we send to the display the text stored in the **print buffer**.

Example: Suppose that we execute this instructions:

```
to print:first line ,
to print: more text in the first line <br>
to print:second line <br>
to print:third line <br>
print
```

then the display is going to show something like:

first line, more text in the first line
second line
third line

3.2.5 to print lines

This instruction is similar to "to print" instruction. With this instruction you can send text to the **print buffer**. Howver, you can send more than one line. The syntax is:

to print lines
[LINES TO BUFFER]

\end

Example: Suppose that we execute this instructions:

```
to print lines
first line,
more text in the first line <br>
second line <br>
third line <br>
\end
print
```

then the display is going to show something like:

first line, more text in the first line
second line
third line

It is important to stress that you have to use the HTML syntax to make a line break.

Example: Suppose that we execute this instructions:

```
to print lines
First
Second
Third
\end
print
```

then the display is going to show something like:

FirstSecondThird

However, we can use the "to print with line breaks" instruction to automatically make a line break at the end of each line.

3.2.6 to print with line breaks

This instruction is similar to "to print lines" instruction. The difference is that with this instruction, the system add "<br>" (the HTML syntax of a line break) at the end of each line. The syntax is:

to print with line breaks
[LINES TO BUFFER (WITH AN AUTOMATIC LINE BREAK AT THE END OF EACH LINE)]
\end

Example: Suppose that we execute this instructions:

```
to print with line breaks
First
Second
Third
```

```
\end
print
```

then the display is going to show something like:

```
First
Second
Third
```

3.2.7 to print files matching with

With this instruction you can send to the **print buffer** a list of the files which match according to a pattern. The syntax to do that is:

to print files matching with [PATTERN]

Example: Suppose that we have this files in a CodeSheet:

- *sword.weapon.csheet*
- *longsword.weapon.csheet*
- *axe.weapon.csheet*
- *stats.csheet*

then when we run to print files matching with weapon, then the display is going to show something like:

- sword.weapon.csheet
- longsword.weapon.csheet
- axe.weapon.csheet

3.2.8 Print attribute's table

To print an attribute's table (2.12.2) you have to use this syntax:

?[ATTRIBUTE'S NAME]

Example:

```
swordAtt: %1d20[die] + 15[melee bonus]
@swordAtt:Attack with long sword
swordDmg: %1d8[die]  + 5 [strengh]
@swordDmg:Damage with long sword
\MACRO: swordAttack
  ?swordAtt
  ?swordDmg
  print
\ENDMACRO
```

Print attribute's table and store the attribute's value in a variable

If you want to store the attribute's value when you print it using the attribute's table instruction, you have to use this syntax:

?[ATTRIBUTE'S NAME] \to [VARIABLE'S NAME]

Example:

```
\MACRO: fullAttack
  ?swordAtt \to swordAttack
  ?swordDmg \to swordDamage
  ?axeAtt \to axeAttack
  ?axeDmg \to axeDamage
  ?biteAtt \to biteAttack
  ?biteDmg \to biteDamage
  to print: <p>SUMMARIZE</p>
  to print: <p>sword: attack->$$!swordAttack$$ / damage->$$!swordDamage$$ </p>
  to print: <p>axe:   attack->$$!axeAttack$$ / damage->$$!axeDamage$$ </p>
  to print: <p>bite:  attack->$$!biteAttack$$ / damage->$$!biteDamage$$ </p>
  print
\ENDMACRO
```

3.2.9 Write/get numerical variables and declarations' values

If you want to print/get a declaration value, you must use this syntax:

$$[DECLARATION]$$

Example: Suppose that you have a file with these lines:

```
attribute: 5
\MACRO: printAttributePlus2
  to print:The attribute's value plus 2 is $$attribute + 2$$. <br> Bye!
  print
\ENDMACRO
```

then when you run the printAttributePlus2 macro, your display is going to show this message:

```
The attribute's value plus 2 is 7
Bye!
```

It is important to stress that the declaration's value is rounded to the closest integer.

Example: Suppose that you have a file with these lines:

```
attribute: 5
\MACRO: printAttributePlusPoint6
  !pointFive = 5/10
  to print:The value is $$attribute + !pointFive + 1/10$$. <br> Bye!
  print
\ENDMACRO
```

then when you run the printAttributePlusPoint6 macro, your display is going to show this message:

```
The value is 6
Bye!
```

However, we can show the decimals using this syntax:

$$[NUMBER OF DECIMALS TO SHOW]$[DECLARATION]$$

Example: Suppose that you have a file with these lines:

```
attribute: 5
\MACRO: printDeclarations
  new array pointFiveArray
  array-add 1/2 \to pointFiveArray
  to print with line breaks
Attribute plus 0.6 with 1 decimal = $$1$attribute + !(0)pointFiveArray + 1/10$$
2/3 with no decimals = $$2/3$$
2/3 with 1 decimal   = $$1$2/3$$
2/3 with 2 decimals  = $$2$2/3$$
2/3 with 3 decimals  = $$3$2/3$$
Bye!
  \end
  print
\ENDMACRO
```

then when you run the printDeclarations macro, your display is going to show this message:

```
Attribute plus 0.6 with 1 decimal = 5,6
2/3 with no decimals = 1
2/3 with 1 decimal = 0,7
2/3 with 2 decimals = 0,67
2/3 with 3 decimals = 0,667
Bye!
```

You can use this syntax in more contexts. You can use this syntax almost always when the system reads text.

Example: Suppose that we have this instruction

```
@var=this is twenty $$5*3+2+3$$, and this is fifty $$25+25$$.
```

then the text stored in the var text variable is:

var = "this is twenty 20, and this is fifty 50."

3.2.10 Write/get text variables

To write/get the text stored in a text variable you must use this syntax:

@@[TEXT VARIABLE'S NAME]@@

Example: Suppose that you have a file with these lines:

```
\MACRO: printTextVariables
  @attackName = bite
  new text-array weaponNames
  text-array-add sword\to weaponNames
  text-array-add axe\to weaponNames
  text-array-add spear\to weaponNames
  to print:The attack's name is @@attackName@@. <br>
  to print:Some weapons: @@(0)weaponNames@@, @@(1)weaponNames@@ and @@(2)weaponNames@@
  print
\ENDMACRO
```

then when you run the printTextVariables macro, your display is going to show this message:

```
The attack's name is bite.
Some weapons: sword, axe and spear
```

You can use this syntax in more contexts. You can use this syntax almost always when the system reads text.

Example: Suppose that you have a file with these lines:

```
attribute:5
\MACRO: printTextVariable
  @bye=Good bye!
  @var=the attribute's value is $$attribute$$.
  @var=@@var@@ And the attribute's value plus 2 is $$attribute+2$$.
  @var=@@var@@ <br>
  @var=@@var@@@@bye@@
  to print:@@var@@
  print
\ENDMACRO
```

then when you run the printTextVariable macro, your display is going to show this message:

```
the attribute's value is 5. And the attribute's value plus 2 is 7.
Good bye!
```

3.2.11 Write/get text from an user's input

We can get text from the user as we do from declarations via $$ ... $$ or from text variables via @@ ... @@. To do that we use this syntax:

@?[MESSAGE TO SHOW]@@

Then a window with the message [MESSAGE TO SHOW] opens and the user can write text.

Example:

```
\MACRO: printInputText
  @s=@?Give me some text@@
  to print:The input text is:@@s@@
  print
\ENDMACRO
```

Example:

```
\MACRO: printAttributeTable
  @s=@?Give me the attribute to show@@
  ?@s
  print
\ENDMACRO
```

3.3 execute

We can execute a macro from another macro. To do that we use this syntax:

execute [MACRO'S NAME]

Example: Suppose that we have this lines in a *.csheet* file:

```
\MACRO: printAttacks
  execute attack1text
  execute attack2text
  execute attack3text
  print
\ENDMACRO

\MACRO: attack1text
  to print:This is the attack 1 <br>
\ENDMACRO

\MACRO: attack2text
  to print:This is the attack 2 <br>
\ENDMACRO

\MACRO: attack3text
  to print:This is the attack 3 <br>
\ENDMACRO
```

then when we execute the printAttacks macro, the display is going to show something like:

```
This is the attack 1
This is the attack 2
This is the attack 3
```

3.3.1 Where the variables live

When we execute a macro (*child macro*) from another macro (*mother macro*), we CAN NOT use or modify the variables of the *mother macro* using instructions in the *child macro*.

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
```

```
  !var = 10
  execute childMacro
  to print:the variable's value is $$!var$$.
  print
\ENDMACRO

\MACRO: childMacro
  !var = 5
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

```
the variable's value is 10.
```

This is useful because we can execute a macro without being worried that there may be variables with the same name. However, sometimes we want to use the variables of the *mother macro* from the *child macro* or we want to use the variables of the *child macro* from the *mother macro* when the *child* finish. To do that we use the import and *export* instructions.

3.3.2 import

We can import a variable from the macro which has executed the current macro (*mother macro*) to the current macro (*child macro*). To do that we use this syntax:

import [VARIABLE'S NAME]

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  !var = 10
  execute childMacro
  print
\ENDMACRO

\MACRO: childMacro
  import var
  to print:the variable's value is $$!var$$.
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

When we import a variable with the import instruction and there are variables of different types with the same name, i.e., a text variable and a numerical variable with the same name, then all variables are imported. You can see the example of the copy instruction which follows the same behavior.

```
the variable's value is 10.
```

3.3.3 export

We can export a variable from the current macro (*child macro*) to the macro which has executed the current macro (*mother macro*). To do that we use this syntax:

export [VARIABLE'S NAME]

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  !var = 10
  execute childMacro
  to print:the variable's value is $$!var$$.
  print
\ENDMACRO

\MACRO: childMacro
  !var = 5
  export var
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

```
the variable's value is 5.
```

When we export a variable with the export instruction and there are variables of different types with the same name, i.e., a text variable and a numerical variable with the same name, then all variables are exported. You can see the example of the copy instruction which follows the same behavior.

3.3.4 execute ... \with

We can add some instructions to execute at the begining of a macro (*child macro*) when we execute it from another macro (*mother macro*). To do that we use this syntax:

```
execute [MACRO'S NAME] \with
[INSTRUCTIONS]
\end
```

This is useful, for example, when we want to import variables from the *mother macro*.

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  !var = 10
  execute childMacro \with
    import var
    !valueToPrint = !var
  \end
\ENDMACRO
```

```
\MACRO: childMacro
  to print:the value is $$!valueToPrint$$.
  print
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

the value is 10.

3.3.5 set default value of

We can set a default value of a variable. When we do that, this instruction creates this variable with the specified value IF THE VARIABLE DOES NOT EXISTS. If the variable exists then the system ignore this instruction. The syntax of this instruction is:

set default value of [VARIABLE'S NAME]: [DECLARATION]

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  !var = 10
  execute childMacro \with
    import var
    !valueToPrint = !var
  \end
\ENDMACRO

\MACRO: childMacro
  !aux = 50
  set default value of valueToPrint: !aux*2
  to print:the value is $$!valueToPrint$$.
  print
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

the value is 10.

and if we execute the childMacro macro, the display is going to show something like:

the value is 100.

3.3.6 set default text of

We can set a default text of a text variable. When we do that, this instruction creates this text variable with the specified text IF THE TEXT VARIABLE DOES NOT EXISTS. If the text variable exists then the system ignore this instruction. The syntax of this instruction is:

set default text of [TEXT VARIABLE'S NAME]:[TEXT]

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  @var =mother text
  execute childMacro \with
    import var
    @textToPrint =@@var@@
  \end
\ENDMACRO

\MACRO: childMacro
  set default text of textToPrint:child text
  to print:the text is "@@textToPrint@@".
  print
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

```
the text is "mother text".
```

and if we execute the childMacro macro, the display is going to show something like:

```
the text is "child text".
```

3.3.7 execute ... \with dependent variables

If you execute a macro (*child macro*) with this instruction, then all variables of the *mother macro* are going to be automatically imported and all variables that you create in the *child macro* are going to be automatically exported. In fact, the instructions of the *child macro* are going to be run in the *mother macro.* The syntax of this instruction is:

execute [MACRO'S NAME] \with dependent variables

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: motherMacro
  !var = 10
  execute childMacro \with dependent variables
  to print:the value is $$!var$$.
  print
  \end
\ENDMACRO

\MACRO: childMacro
  to print:the value is $$!var$$.
  !var = 5
\ENDMACRO
```

then when we execute the motherMacro macro, the display is going to show something like:

```
the value is 10.
the value is 5.
```

3.4 if ... \then

You can use a conditional expression in your macros. The syntax is:

if [CONDITION] \then
[INSTRUCTIONS]
\end

To write the condition you have to use the same syntax that we use in the if parameter (2.10.1).

Example: Suppose that we have these lines in a *.csheet* file:

```
attribute: 5
\MACRO: ifMacro
  !var = 2
  if attribute + !var > 6 \then
    to print:attribute plus 2 is more than 6.
  \end
  if attribute + !var <= 6 \then
    to print:attribute plus 2 is less or equal to 6.
  \end
  print
\ENDMACRO
```

then when we execute the ifMacro macro, the display is going to show something like:

```
attribute plus 2 is more than 6.
```

3.4.1 if ... else ...

You can use a conditional expression with an else expression in your macros. The syntax is:

if [CONDITION] \then
[INSTRUCTIONS]
\end else
[INSTRUCTIONS]
\end

Example: Suppose that we have these lines in a *.csheet* file:

```
attribute: 4
\MACRO: ifElseMacro
  !var = 2
  if attribute + !var > 6 \then
    to print:attribute plus 2 is more than 6.
  \end else
```

```
    to print:attribute plus 2 is less or equal to 6.
  \end
  print
\ENDMACRO
```

then when we execute the ifElseMacro macro, the display is going to show something like:

attribute plus 2 is less or equal to 6.

3.5 Loops

3.5.1 while ... \do

The syntax to use a while structure in your macros is:

while [CONDITION] \do
[INSTRUCTIONS]
\end

Example: Suppose that we have these lines in a *.csheet* file:

```
\MACRO: whileMacro
  !var = 0
  !nDice = 0
  while !var < 200 \do
   !var = !var + %1d6
   !nDice = !nDice + 1
  \end
  to print: we have needed $$!nDice$$ dice of 6 sides to sum 200.
  print
\ENDMACRO
```

then when we execute the whileMacro macro, the display is going to show something like:

we have needed 57 dice of 6 sides to sum 200.

where 57 can change depending on the dice results.

3.5.2 for ... \from ... \to ... \do

We can use a for instruction to iterate over integers. The syntax to do that is:

for [VARIABLE'S NAME] \from [FIRST INTEGER DECLARATION] \to [LAST INTEGER DEC.] \do
[INSTRUCTIONS]
\end

Example: Suppose that we have these lines in a *.csheet* file:

```
level : 5
\MACRO: forMacro
  new array diceResults
  for i \from 1 \to level + 1 \do
    !dice = %1d6
    to print:Die number $$!i$$ is equal to $$!dice$$. <br>
    array-add !dice \to diceResults
  \end
  sum  diceResults \to aux
  to print:SUM = $$!aux$$ <br>
  min  diceResults \to aux
  to print:MIN = $$!aux$$ <br>
  max  diceResults \to aux
  to print:MAX = $$!aux$$ <br>
  sort diceResults \to diceResultsSorted
  to print:sorted results:
  length of diceResultsSorted \to n
  for i \from 0 \to !n-1 \do
    to print: $$!(!i)diceResultsSorted$$
  \end
  print
\ENDMACRO
```

then when we execute the forMacro macro, the display is going to show something like:

```
Die number 1 is equal to 5.
Die number 2 is equal to 2.
Die number 3 is equal to 6.
Die number 4 is equal to 5.
Die number 5 is equal to 3.
Die number 6 is equal to 4.
SUM = 25
MIN = 2
MAX = 6
sorted results: 2 3 4 5 5 6
```

where the numbers can change depending on the dice results.

3.5.3 for ... \in ... \do

We can use a for instruction to iterate over the elements of an array. The syntax to do that is:

```
for [VARIABLE'S NAME] \in [ARRAY'S NAME] \do
[INSTRUCTIONS]
\end
```

Example: Suppose that we have these lines in a *.csheet* file:

```
level : 5
```

```
\MACRO: forMacro
  new array diceResults
  for i \from 1 \to level + 1 \do
    !dice = %1d6
    to print:Die number $$!i$$ is equal to $$!dice$$. <br>
    array-add !dice \to diceResults
  \end
  sort diceResults \to diceResultsSorted
  to print:sorted results:
  for dieResult \in diceResultsSorted \do
    to print: $$!dieResult$$
  \end
  print
\ENDMACRO
```

then when we execute the forMacro macro, the display is going to show something like:

```
Die number 1 is equal to 1.
Die number 2 is equal to 4.
Die number 3 is equal to 2.
Die number 4 is equal to 1.
Die number 5 is equal to 1.
Die number 6 is equal to 3.
sorted results: 1 1 1 2 3 4
```

where the numbers can change depending on the dice results.

We can also iterate text arrays.

Example: Suppose that we have these lines in a *.csheet* file:

```
level : 5
\MACRO: forMacro
  new text-array diceString
  for i \from 1 \to level + 1 \do
    !dice = %1d6
    text-array-add the die number $$!i$$ is equal to $$!dice$$\to diceString
  \end
  for str \in diceString \do
    to print: @@str@@ <br>
  \end
  print
\ENDMACRO
```

then when we execute the forMacro macro, the display is going to show something like:

```
the die number 1 is equal to 1
the die number 2 is equal to 4
the die number 3 is equal to 3
the die number 4 is equal to 1
the die number 5 is equal to 3
the die number 6 is equal to 6
```

where the numbers can change depending on the dice results.

However, if we have a numerical array and a text array with the same name, then this for instruction is going to iterate the NUMERICAL array.

3.6 Enable and disable files

We can enable or disable files according to a *pattern* using the enable and disable instructions.

3.6.1 enable

The syntax to enable files is:

enable [PATTERN]

Where the [PATTERN] can not have spaces (blanks). For this reason we recommend not to use spaces in the *.csheet* file names.

Example: Suppose that we have this files in a CodeSheet:

- *sword.weapon.csheet*
- *longsword.weapon.csheet*
- *axe.weapon.csheet*
- *stats.csheet*

then when we run:

- enable weapon
  the files:
  - *sword.weapon.csheet*
  - *longsword.weapon.csheet*
  - *axe.weapon.csheet*

  are going to be enabled.
- enable sword.weapon.csheet
  the files:
  - *sword.weapon.csheet*
  - *longsword.weapon.csheet*

  are going to be enabled.
- enable longsword.weapon.csheet
  the file:

  - *longsword.weapon.csheet*

  is going to be enabled.

- enable .csheet
  all files are going to be enabled.

Example:

```
\MACRO: enableWeapon
  @name=@?Give the name of the weapon that you want to enable@@.weapon
  enable @name
\ENDMACRO
```

3.6.2 disable

The syntax to disable files is:

disable [PATTERN]

Where the [PATTERN] can not have spaces (blanks). For this reason we recommend not to use spaces in the *.csheet* file names.

You can see the example of enable instruction (3.6.1) to see how *patterns* work.

3.7 Counters

A counter is a special element of the CodeSheet software that you can use to modify more than one numerical variable with only one instruction.

You can do two things with a Counter:

- Link a variable
- Modify linked variables adding a declaration's value to all of them

To link a variable you have to use this syntax:

*[COUNTER'S NAME]:[VARIABLE'S NAME]

To modify linked variables you have to use this syntax:

*[COUNTER'S NAME]=[DECLARATION]

You can import and export counters as you can do with variables, using the import and export instructions.

Example:

```
\MACRO: counterExample
  !var1=5
  !var2=10
  !var3=15
  *counterA: var1
  *counterA: var2
  *counterB: var3
  *counterA=2
  *counterA=1
  *counterB=-10
  to print:var1=$$!var1$$<br>
  to print:var2=$$!var2$$<br>
  to print:var3=$$!var3$$<br>
  print
\ENDMACRO
```

then when we execute the counterExample macro, the display is going to show something like:

```
var1=8
var2=13
var3=5
```

Example:

```
\MACRO: shieldOfFaith
  to print: <p>*enabling spell.shieldOfFaith.csheet file ...</p>
  enable spell.shieldOfFaith.csheet
  to print: <p>*linking durationShieldOfFaith variable to roundCounter ...</p>
  import roundCounter
  # shield of faith lasts as many rounds as your level
  !durationShieldOfFaith=level
  *roundCounter:durationShieldOfFaith
  export roundCounter
  export durationShieldOfFaith
  print
\ENDMACRO

\MACRO: mageArmor
  to print: <p>*enabling spell.mageArmor.csheet file ...</p>
  enable spell.mageArmor.csheet
  to print: <p>*linking durationMageArmor variable to roundCounter ...</p>
  import roundCounter
  # mage armor lasts as many hours as your level (1h=60min and 1min=10rounds)
  !durationMageArmor=level*60*10
  *roundCounter:durationMageArmor
  export roundCounter
  export durationMageArmor
  print
\ENDMACRO

## It MUST be executed "with depenedent variables"
\MACRO: spendOneRound
  to print: spending one round ...
  *roundCounter=-1
  print
\ENDMACRO

## It MUST be executed "with depenedent variables"
\MACRO: spendRounds
  !aux=\windowInput(How many rounds do you want to spend?)
  *roundCounter=-!aux
  to print: spending $$!aux$$ rounds ...
  print
\ENDMACRO
```

3.7.1 counter-del

You can remove or disassociate a variable from a counter. To do that we use this syntax:

counter-del [VARIABLES'S NAME] \of [COUNTER'S NAME]

Example:

```
\MACRO: endMageArmor
  to print: disabling spell.mageArmor.csheet file ...<br>
  disable spell.mageArmor.csheet
  to print: removing durationShieldOfFaith of roundCounter ...<br>
  import roundCounter
  counter-del durationMageArmor \of roundCounter
  export roundCounter
  print
\ENDMACRO
```

3.8 exit

You can use this instruction to finish a macro immediately. The syntax is:

exit

Example:

```
\MACRO: makingAttacks
  while -1 < 1 \do
    ?attack
    print
    !aux = \windowOption(do you want to do another attack?, yes:1, no:0 )
    if !aux = 0 \then
      exit
    \end
  \end
\ENDMACRO
```

3.8.1 exit and returning a value

You can return a value when you use the exit instruction in order to use a macro as a parameter (2.6). The syntax is:

exit [DECLARATION]

Example:

```
attack: &highestTwoDice + 20

\MACRO: highestTwoDice
  new array arr
  array-add %1d20 \to arr
  array-add %1d20 \to arr
  max arr \to return
  exit !return
\ENDMACRO
```

3.9 set folder

You can organize your macros in folders. To do that we use this syntax:

set folder: [FOLDER'S NAME]

Example:

```
\MACRO: axeAttack
  set folder: attacks
  ?axeAttack
  print
  !aux = \windowOption(what happens?, hit:1, critic:2, no hit:0)
  if !aux = 1 \then
    ?axeDamage
    print
  \end
  if !aux = 2 \then
    ?axeCritAttack
    print
    !aux = \windowOption(is it a critic?, yes:1, no:0)
    if !aux = 1 \then
      ?axeCritDamage
      print
    \end else
      ?axeDamage
      print
    \end
  \end
\ENDMACRO
```

3.10 Multiple declarations of the same macro

You can declare the same macro as many times as you want. When you execute a macro all declarations are going to be executed (if the corresponding file is enabled).

Example: Suppose that we have these two files:

---

attacks.csheet

```
\MACRO: fullAttack
  ?rightSwordAttack
  ?rightSwordDamage
  ?leftSwordAttack
  ?leftSwordDamage
\ENDMACRO
```

---

haste.csheet

```
\MACRO: fullAttack
  ?rightSwordAttack
  ?rightSwordDamage
\ENDMACRO
```

---

then when we execute the fullAttack macro,

- if only *attacks.csheet* is enabled, then these instructions are going to be executed:

  ```
  ?rightSwordAttack
  ?rightSwordDamage
  ?leftSwordAttack
  ?leftSwordDamage
  ```

- if both files are enabled, then these instructions are going to be executed:

  ```
  ?rightSwordAttack
  ?rightSwordDamage
  ?leftSwordAttack
  ?leftSwordDamage
  ?rightSwordAttack
  ?rightSwordDamage
  ```

However, to know the order in which these instructions are going to be executed we have to use the set order instruction.

3.10.1 set order

We use this instruction to set the order in which different declarations of the same macro are going to be executed. The syntax is:

set order: [DECLARATION]

It is important to stress that the order is set when the code sheet opens. Therefore, you should not use variables to set the order. In fact, we recommend to use only numerical values (although you can use attributes or other parameters if you want).

The values should be integers. If you do not use integers, then the values are going to be rounded to the closest integer.

The execution order will be according to the values you set from lowest to highest.

Example: Suppose that we have these macro declarations:

```
\MACRO: fullAttack
  set order: 2
  to print: normal attacks
  ?rightSwordAttack
  ?rightSwordDamage
  ?leftSwordAttack
  ?leftSwordDamage
  print
\ENDMACRO

\MACRO: fullAttack
  set order: 1
  to print: haste attack
  ?rightSwordAttack
  ?rightSwordDamage
  print
\ENDMACRO
```

then when we execute the fullAttack macro, the instructions are going to be executed in these order

```
  to print: haste attack
  ?rightSwordAttack
  ?rightSwordDamage
  print
  to print: normal attacks
  ?rightSwordAttack
  ?rightSwordDamage
  ?leftSwordAttack
  ?leftSwordDamage
  print
```

EXPLICAR QUE ESTA COMPILADO CON JAVA 7 Y SE HA PROBADO EN WIN7 Y EN LINUX.