ELENA – Formula interpreter

1 Implementation

When implemented on all plugins, it gives the possibility of executing the formulas on the input files and on the output files generated.

2 Description of the interpreter

The basic idea arose from the need to be able to perform the calculation of the centers of gravity of the components produced for the SP3D catalog. Starting from this initial requisite, and with the aim of giving greater flexibility to all the plugins of the interface in those cases in which it is necessary to perform mathematical calculations, add or modify values in the columns of the input and output files, and execute operations on strings, an interpreter of configurable expressions has been implemented. To take advantage of the interpreter, it is necessary to set a new rule_formulas.xml configuration file (valid for all plugins).

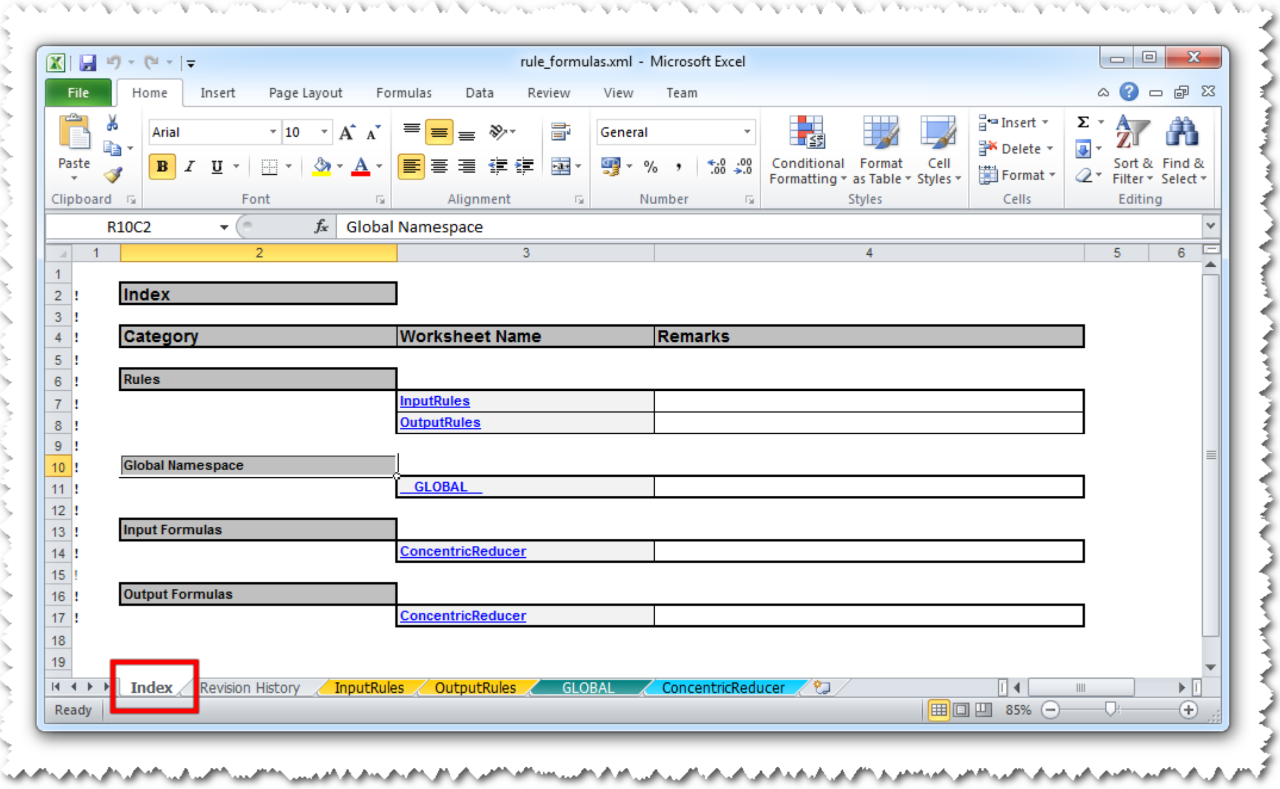
Subsequently, within the Builder Config_XX.xml file, it is necessary to define the (optional) “rule_expressions_pathname” parameter in the “xls” section and set it with the pathname of the rule_formulas.xml file.

The following paragraphs describe how to configure the rule_formulas.xml file, and provide an example of one.

3 Rule_formulas.xml

The rule_formulas.xml file has the same structure as the other configuration files; below is a description of the sheets of the configuration file:

- *Index:* lists all the sections present in the file itself, with the related hyperlinks.
- *Revision History:* is used for keeping track of the modifications made to the file over time.
- *InputRules*: contains the mapping for applying the formulas to the Input files (csv files) during the loading process.
- *OutputRules*: contains the mapping for applying the formulas to the output files produced during the processing activity
- *Formulas*: the first four sheets are fixed and must always be present in the configuration file, while a formula must be configured in a new sheet of the workbook created especially (one sheet, one formula). A formula will be executed on every input record or on every record output produced during the respective loading and generation processes.
- *__GLOBAL__*: in this (optional) sheet it is possible to configure a formula which will be executed once only in a global namespace. The variables that will be assigned in this namespace will be visible during the entire processing activity of the Builder to all the specific formulas that will be executed during the loading of the input files and the generation of the output files.



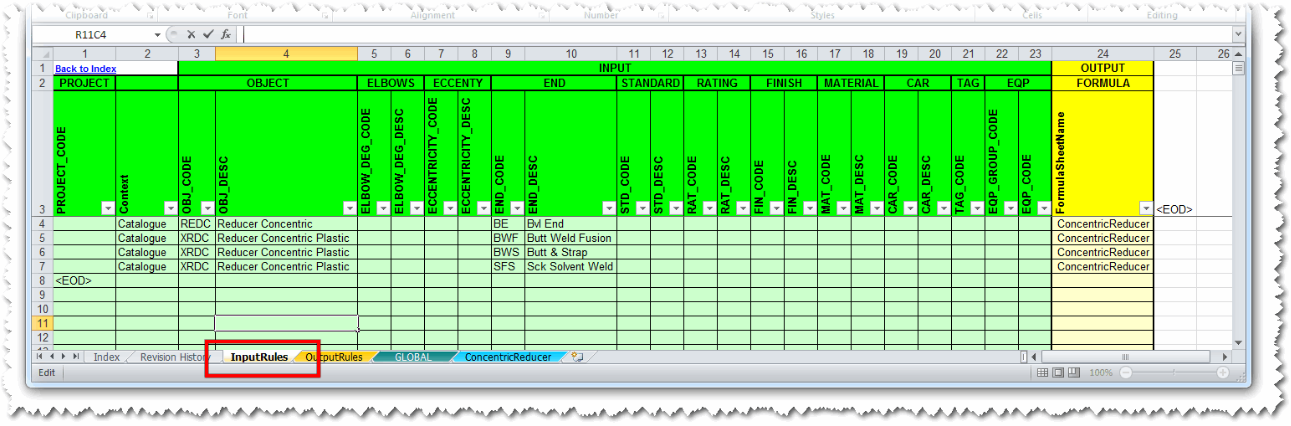
R10C2 — Global Namespace

	1	2	3	4
1				
2	!	**Index**		
3	!			
4	!	**Category**	**Worksheet Name**	**Remarks**
5	!			
6	!	**Rules**		
7	!		InputRules	
8	!		OutputRules	
9	!			
10	!	**Global Namespace**		
11	!		__GLOBAL__	
12	!			
13	!	**Input Formulas**		
14	!		ConcentricReducer	
15	!			
16	!	**Output Formulas**		
17	!		ConcentricReducer	
18				
19				

NOTE: the formulas applied to the input files can add new columns to the input records at runtime; while the formulas applied to the output files cannot add new columns but only set or modify values in columns already defined in the cata_headers.xml and spec_headers.xml configuration files.

3.1 InputRules

The *InputRules* section is illustrated below:

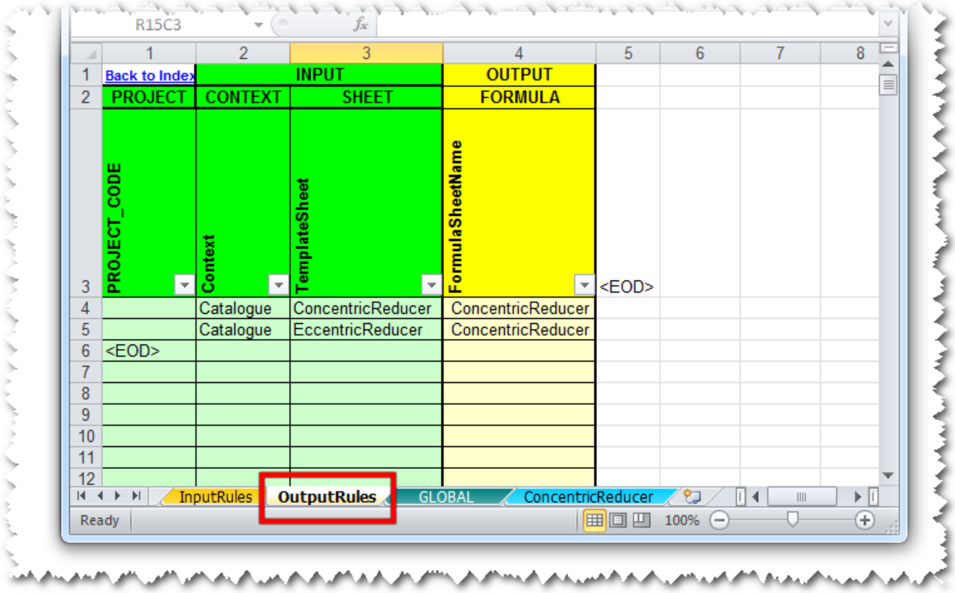


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	Back to Index		INPUT																					OUTPUT	
2	PROJECT		OBJECT		ELBOWS		ECCENTY		END		STANDARD		RATING		FINISH		MATERIAL		CAR		TAG	EQP		FORMULA	
3	PROJECT_CODE	Context	OBJ_CODE	OBJ_DESC	ELBOW_DEG_CODE	ELBOW_DEG_DESC	ECCENTRICITY_CODE	ECCENTRICITY_DESC	END_CODE	END_DESC	STD_CODE	STD_DESC	RAT_CODE	RAT_DESC	FIN_CODE	FIN_DESC	MAT_CODE	MAT_DESC	CAR_CODE	CAR_DESC	TAG_CODE	EQP_GROUP_CODE	EQP_CODE	FormulaSheetName	<EOD>
4		Catalogue	REDC	Reducer Concentric					BE	Bvl End														ConcentricReducer	
5		Catalogue	XRDC	Reducer Concentric Plastic					BWF	Butt Weld Fusion														ConcentricReducer	
6		Catalogue	XRDC	Reducer Concentric Plastic					BWS	Butt & Strap														ConcentricReducer	
7		Catalogue	XRDC	Reducer Concentric Plastic					SFS	Sck Solvent Weld														ConcentricReducer	
8	<EOD>																								

The mapping logic is similar to the other configuration files (for example the Templates section of the rule_codelist.xml file). The user maps the components on which it is intended to apply the formula on the basis of their mechanical characteristics (with the green background). In the “FormulaSheetName” column (with the yellow background) s/he should set the name of the sheet in which the formula to be applied is set.

3.2 OutputRules

The *OutputRules* section is illustrated below:



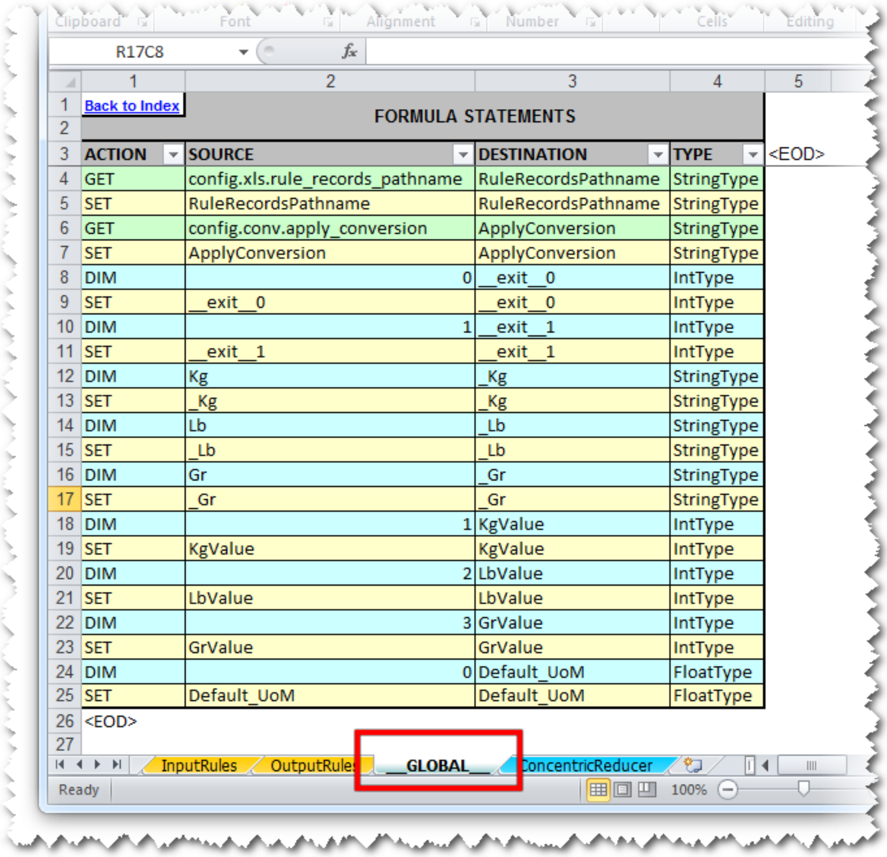
	1	2	3	4	5
1	Back to Index	INPUT		OUTPUT	
2	PROJECT	CONTEXT	SHEET	FORMULA	
3	PROJECT_CODE	Context	TemplateSheet	FormulaSheetName	<EOD>
4		Catalogue	ConcentricReducer	ConcentricReducer	
5		Catalogue	EccentricReducer	ConcentricReducer	
6	<EOD>				

The rules for applying the formulas can be defined by the processing context (*Context)* and by the name of the sheet of the output template in which the records generated by the Builder will be appended (*TemplatesSheet),* more specifically, the sheets contained within the cata_template.xml and spec_template.xml files.

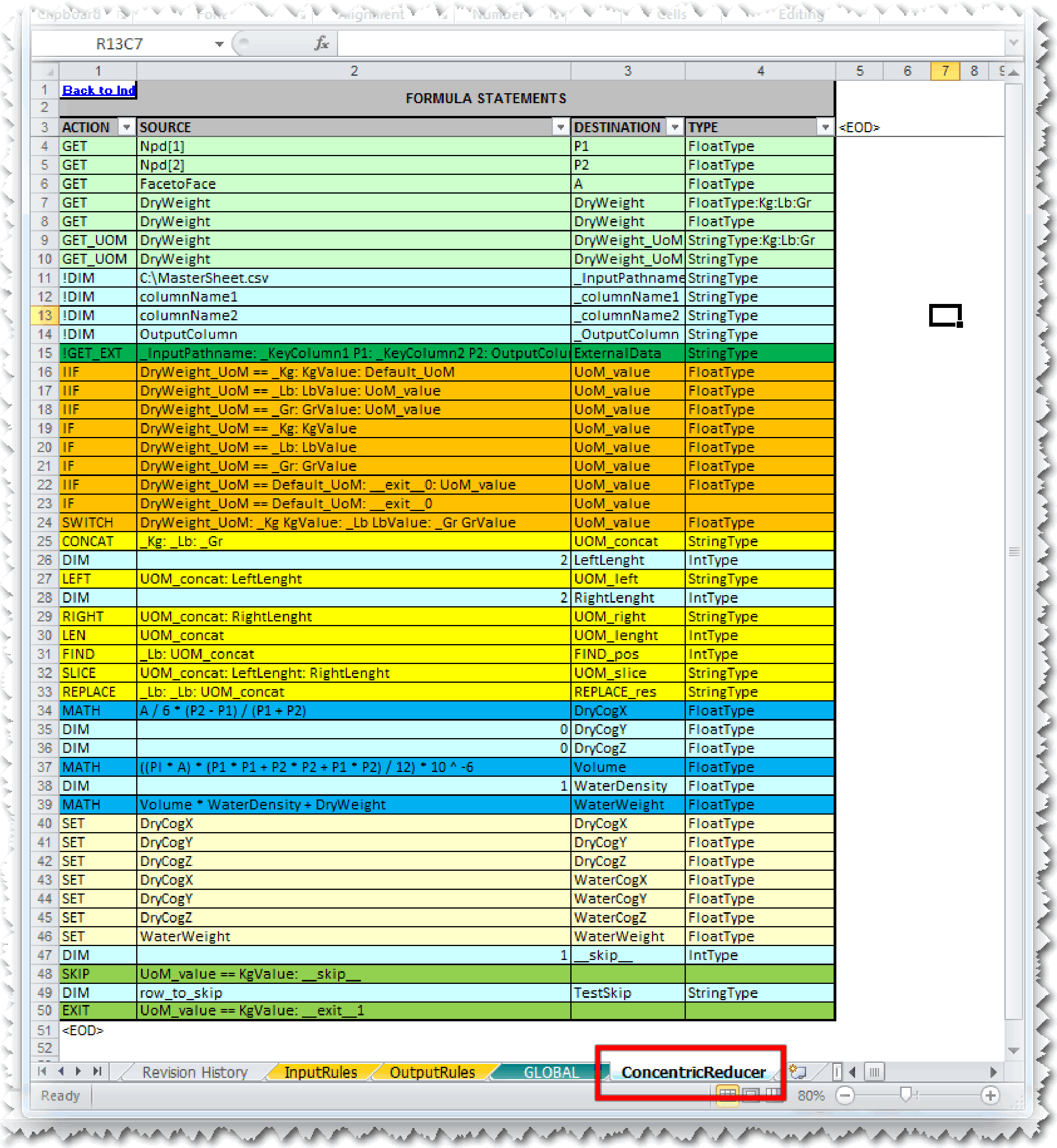
NOTE: the necessity of specifying the template context-sheet pairing concerns only the *Catalogue* context. The catalog (cata_template.xml file) contains the sheets relating to all the components. In the case of the sheets of the spec_template.xml file, they have a biunivocal correspondence with the related processing context; hence it is sufficient to set only one of the two parameters.

3.3 __GLOBAL__, InputRules and OutputRules sections

Below are illustrations of an example of a *__GLOBAL__* section and an example of an *Input/OutputRules* section, merely for the purpose of showing the various configuration possibilities:



	1	2	3	4	5
1	Back to Index	FORMULA STATEMENTS			
2					
3	ACTION	SOURCE	DESTINATION	TYPE	<EOD>
4	GET	config.xls.rule_records_pathname	RuleRecordsPathname	StringType	
5	SET	RuleRecordsPathname	RuleRecordsPathname	StringType	
6	GET	config.conv.apply_conversion	ApplyConversion	StringType	
7	SET	ApplyConversion	ApplyConversion	StringType	
8	DIM	0	__exit__0	IntType	
9	SET	__exit__0	__exit__0	IntType	
10	DIM	1	__exit__1	IntType	
11	SET	__exit__1	__exit__1	IntType	
12	DIM	Kg	_Kg	StringType	
13	SET	_Kg	_Kg	StringType	
14	DIM	Lb	_Lb	StringType	
15	SET	_Lb	_Lb	StringType	
16	DIM	Gr	_Gr	StringType	
17	SET	_Gr	_Gr	StringType	
18	DIM	1	KgValue	IntType	
19	SET	KgValue	KgValue	IntType	
20	DIM	2	LbValue	IntType	
21	SET	LbValue	LbValue	IntType	
22	DIM	3	GrValue	IntType	
23	SET	GrValue	GrValue	IntType	
24	DIM	0	Default_UoM	FloatType	
25	SET	Default_UoM	Default_UoM	FloatType	
26	<EOD>				
27					



FORMULA STATEMENTS

	ACTION	SOURCE	DESTINATION	TYPE	
3					<EOD>
4	GET	Npd[1]	P1	FloatType	
5	GET	Npd[2]	P2	FloatType	
6	GET	FacetoFace	A	FloatType	
7	GET	DryWeight	DryWeight	FloatType:Kg:Lb:Gr	
8	GET	DryWeight	DryWeight	FloatType	
9	GET_UOM	DryWeight	DryWeight_UoM	StringType:Kg:Lb:Gr	
10	GET_UOM	DryWeight	DryWeight_UoM	StringType	
11	!DIM	C:\MasterSheet.csv	_InputPathname	StringType	
12	!DIM	columnName1	_columnName1	StringType	
13	!DIM	columnName2	_columnName2	StringType	
14	!DIM	OutputColumn	_OutputColumn	StringType	
15	!GET_EXT	_InputPathname: _KeyColumn1 P1: _KeyColumn2 P2: OutputColu	ExternalData	StringType	
16	IIF	DryWeight_UoM == _Kg: KgValue: Default_UoM	UoM_value	FloatType	
17	IIF	DryWeight_UoM == _Lb: LbValue: UoM_value	UoM_value	FloatType	
18	IIF	DryWeight_UoM == _Gr: GrValue: UoM_value	UoM_value	FloatType	
19	IF	DryWeight_UoM == _Kg: KgValue	UoM_value	FloatType	
20	IF	DryWeight_UoM == _Lb: LbValue	UoM_value	FloatType	
21	IF	DryWeight_UoM == _Gr: GrValue	UoM_value	FloatType	
22	IIF	DryWeight_UoM == Default_UoM: __exit__0: UoM_value	UoM_value	FloatType	
23	IF	DryWeight_UoM == Default_UoM: __exit__0	UoM_value		
24	SWITCH	DryWeight_UoM: _Kg KgValue: _Lb LbValue: _Gr GrValue	UoM_value	FloatType	
25	CONCAT	_Kg: _Lb: _Gr	UOM_concat	StringType	
26	DIM	2	LeftLenght	IntType	
27	LEFT	UOM_concat: LeftLenght	UOM_left	StringType	
28	DIM	2	RightLenght	IntType	
29	RIGHT	UOM_concat: RightLenght	UOM_right	StringType	
30	LEN	UOM_concat	UOM_lenght	IntType	
31	FIND	_Lb: UOM_concat	FIND_pos	IntType	
32	SLICE	UOM_concat: LeftLenght: RightLenght	UOM_slice	StringType	
33	REPLACE	_Lb: _Lb: UOM_concat	REPLACE_res	StringType	
34	MATH	A / 6 * (P2 - P1) / (P1 + P2)	DryCogX	FloatType	
35	DIM	0	DryCogY	FloatType	
36	DIM	0	DryCogZ	FloatType	
37	MATH	((PI * A) * (P1 * P1 + P2 * P2 + P1 * P2) / 12) * 10 ^ -6	Volume	FloatType	
38	DIM	1	WaterDensity	FloatType	
39	MATH	Volume * WaterDensity + DryWeight	WaterWeight	FloatType	
40	SET	DryCogX	DryCogX	FloatType	
41	SET	DryCogY	DryCogY	FloatType	
42	SET	DryCogZ	DryCogZ	FloatType	
43	SET	DryCogX	WaterCogX	FloatType	
44	SET	DryCogY	WaterCogY	FloatType	
45	SET	DryCogZ	WaterCogZ	FloatType	
46	SET	WaterWeight	WaterWeight	FloatType	
47	DIM	1	__skip__	IntType	
48	SKIP	UoM_value == KgValue: __skip__			
49	DIM	row_to_skip	TestSkip	StringType	
50	EXIT	UoM_value == KgValue: __exit__1			
51	<EOD>				

As can be seen in the illustrations, both the section of a global nature and the specific section have the same columns. One of the rows in a formula is named "statement" and consists of four columns, whose meaning is as follows:

- **ACTION**: contains the type of operation to be performed (we will call these actions *Instructions*)

- **SOURCE**: the source column is compiled on the basis of the instruction to be executed and may contain a mathematical expression, the parameters of an operation on strings, an initial value to be assigned to a variable that will be used subsequently, or a logic expression.
- **DESTINATION:** contains the destination variable in which to enter the result of the statement.
- **TYPE:** contains the type of the destination variable.

3.3.1 Instructions

The instruction present within the ACTION column can be gathered into five different *groups,* which are illustrated below:

1) RECOVERY-ASSIGNMENT INSTRUCTIONS:

i) DIM – implements the assignment of a value to a variable, which will be defined in the *namespace of the formula.*

Here below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
DIM	0	Default_UoM	FloatType

In the above example the numerical value zero is assigned to the Default_UoM variable.

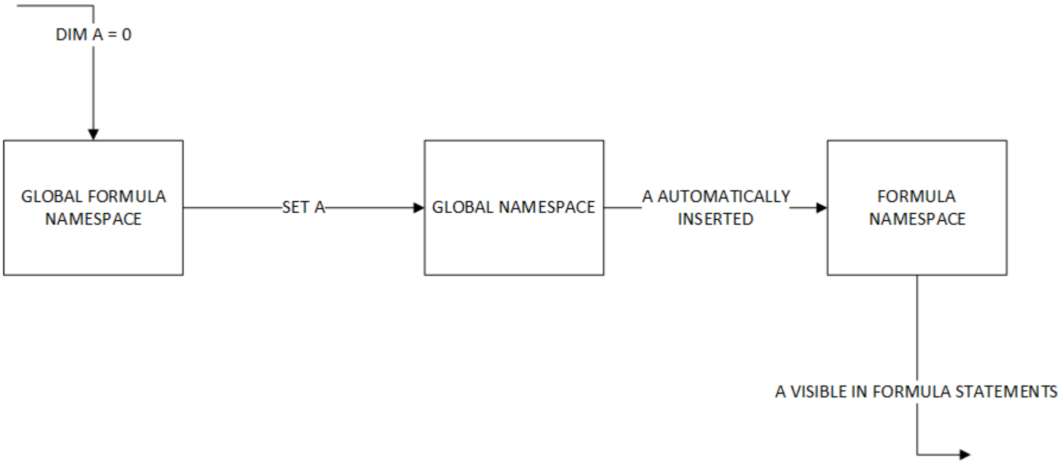
ii) SET – sets the content of a destination variable in a result namespace. If the SET instruction is used in the __GLOBAL__ sheet, it sets the content of the variable in the global namespace, while if it is used in any other formula sheet, it sets the content of a variable of the Local namespace in the input or output record on the basis of where the formula is applied. The SET instruction always overwrites the content of the variable (in the case of a __GLOBAL__ sheet), or the content of the record field (in all the other cases).

Below is an example of a statement:

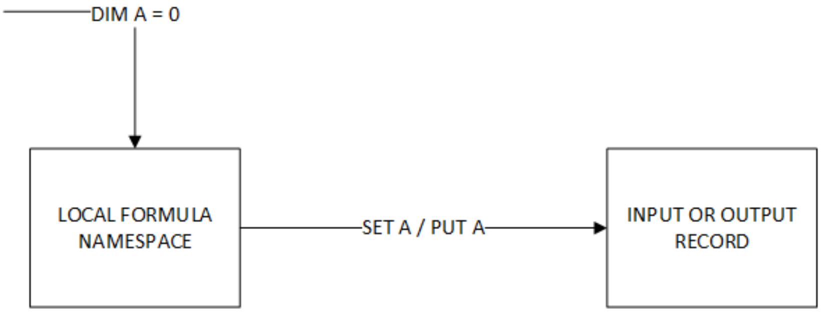
ACTION	SOURCE	DESTINATION	TYPE
SET	Default_UoM	Default_UoM	FloatType

In the above example the content of the Default_UoM variable is set in a Float-type variable of the same name in the result namespace (which can be an input record, an output record, or the global namespace in the case of the __GLOBAL__ sheet).

For a better understanding of the operating logic of the SET actions, see the following flow chart for the *global namespace* case:



Below is the equivalent diagram for the *formula namespace*:



iii) DIMNS – implements the assignment of a value to a variable and define it in the *namespace of the formula,* with the same name and type. Allows the user to perform the DIM and SET operation at the same time.

Here below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
DIMNS	0	Default_UoM	FloatType

In the above example the numerical value zero is assigned to the Default_UoM variable.

iv) PUT – the same operation as the SET instruction, except that if the destination variable is already assigned, an error message will be signaled in the file log and the value will not be set.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
PUT	DryWeight	DryWeight	FloatType

In the above example the content of the DryWeight variable is set in a Float-type variable of the same name in the result namespace (which can be an input record, an output record, or the global namespace in the case of the __GLOBAL__ sheet).

v) GET – allows the user to recover a value from an input or output record (according to where the formula is applied) and set it in the execution *namespace* of the formula. In the case of the

__*GLOBAL*__ section, the GET instruction can load only the parameters of the Config_XX.xml file of the Builder.

Within the TYPE column, the user must specify the type of the destination variable. In the case of CAD systems (for example, SP3D) which concatenate the units of measurement to the numerical value, the GET instruction automatically separates the numerical value from the unit of measurement (implicit form). As an option it is possible to indicate the expected unit or units of measurement separated by ":" (explicit form). In both the implicit and explicit forms, GET loads the value of the parameter without a unit of measurement and converts it into the type set in the TYPE column. In the case of the implicit form, in which the units of measurement are specified, ELENA will signal an exception if it identifies a unit of measurement that is not among those declared. In the event that the units of measurement are not specified (implicit form), ELENA will accept any unit of measurement value and will load only the numerical value without providing exceptions.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
GET	DryWeight	DryWeight	FloatType:Kg:Lb:Gr

In the above example the content of the DryWeight variable is set in a Float-type variable of the same name in the namespace of the formula.

In the event that the unit of measurement is other than "Kg", "Lb" or "Gr", or if the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

vi) GET_UOM – the functioning is similar to that of GET with the difference that instead of loading the numerical value of the parameter, the unit of measurement is loaded. Once again, as in the previous case, it is possible to choose the explicit form specifying a range of possible units of measurement, or the implicit form without specifying the units of measurement.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
GET_UOM	DryWeight	DryWeight_UoM	StringType:Kg:Lb:Gr

In the above example, the unit of measurement of the DryWeight variable is set in a variable named DryWeight_UoM in the namespace of the formula.

In the event that the unit of measurement is other than "Kg", "Lb" or "Gr", or if the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file.

vii) GET_EXT – allows the user to load a value from an external pre-compiled file. Can be used for importing values at runtime defining the parameters that make up the access key and the value to be returned. The external file structure supports the variable attributes mapping.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
GET_EXT	InputPathname: KeyColumnName1 P1: KeyColumnName2 P2: OutputColumnName	ExternalData	StringType

In the above example, access is made to the file pathname set in the *InputPathname* variable. The search is made in the columns set in the variables *KeyColumnName1* e *KeyColumnName2* and returns the value in the variable OutputColumnName. The value returned is set in the String-type ExternalData variable in the namespace of the formula.

In the event that the search yields a negative result, or if the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

2) FLOW CONTROL OPERATORS:

i) IF – if the evaluated expression is true, a value is assigned in a variable. It is necessary to bear in mind that if the condition is false, the variable set in the DESTINATION column will be set with the None value and this could give rise to an exception in the subsequent statements in the event that reference is made to the variable without first checking its value. The syntax of the instruction is as follows:

“expression to be evaluated” : “value to be assigned if the condition is true”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
IF	DryWeight_UoM == _Kg: KgValue	UoM_value	FloatType

In the above example the expression set in the SOURCE column is evaluated and in the event that the result is True, the value contained in the KgValue variable will be assigned to the Float-type UoM_value variable (defined in the DESTINATION column) in the namespace of the formula. In the event that the result of the expression is False, the UoM_value variable will be set on Null in the namespace of the formula.

In the event that the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

ii) IIF – if the evaluated expression is true, a value is assigned; if the condition is false, an alternative value is assigned. The structure of the instruction is as follows:

“expression to be evaluated” : “value to be assigned if the condition is true” : “value to be assigned if the condition is false”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
IIF	DryWeight_UoM == _Kg: KgValue: Default_UoM	UoM_value	FloatType

In the above example the expression set in the SOURCE column is evaluated and in the event that the result is True, the value contained in the KgValue variable will be assigned to the Float-type UoM_value variable (defined in the DESTINATION column) in the namespace of the formula. In the event that the result of the expression is False, the value contained in the Default_Uom variable will be assigned to the Float-type UoM_value variable in the namespace of the formula.

In the event that the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

iii) SWITCH – the switch instruction enables the user to "manage" several cases of the value of a variable that has previously been initialized. This is useful when managing a large number of possibilities of values of a variable without using the IF condition. In the specific case, given a "switch" variable, the possible values of the variable are evaluated, when the value evaluated corresponds to that of the "case" variable, the destination variable is assigned with the corresponding value of the "case". The syntax of the instruction is as follows:

"switch variable" : "case variable1" "case value1" : "case variable2" "case value2" : ... : "case variableN" "case valueN": "default value if none of the previous cases is verified"

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
SWITCH	DryWeight_UoM: _Kg KgValue: _Lb LbValue: _Gr GrValue: Otherwise_value	UoM_value	FloatType

In the above example the value contained in the DryWeight_UoM variable (switch variable) is evaluated, when the value of the DryWeight_UoM variable corresponds with the value of one of the _Kg, _Lb, or _Gr variables (case variables), the value of the respective KgValue, LbValue, or GrValue variable is assigned to the result UoM_value variable.

In the event that none of the values of the _Kg, _Lb, or _Gr variables corresponds with the value of the DryWeight_UoM variable, the default value Otherwise_value is assigned.

If the statement refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

NOTE: in the expressions associated with operators of this type, it is not possible to use constant values in explicit mode. All constant values must be assigned in a variable by means of the DIM instruction before being used.

3) OPERATOR FOR MATHEMATICAL EXPRESSIONS:

i) MATH – this instruction enables the user to implement mathematical expressions. The interpreter accepts simple and intuitive grammar. The operators have normal precedence, f (x, y, z) (calls a function), ^ (raising to a power), *, / and % (respectively: multiplication, division, and the remainder), +, -, and || (addition, subtraction, and concatenation of strings) and are joined from left to right. The list of the possible operators, functions and constants is illustrated in the "Standard mathematical syntax" paragraph of this document.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
MATH	A / 6 * (P2 - P1) / (P1 + P2)	WaterWeight	FloatType

In the above example the mathematical expression set in the source column is evaluated and the result is set in the WaterWeight destination variable.

In the event that the mathematical expression refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

NOTE: differently from the flow control operators, for the MATH action it is also possible to use constant numerical values, without previously assigning the value to a variable.

4) OPERATORS FOR FUNCTIONS ON STRINGS

i) CONCAT – performs the operation of concatenating two or more strings. The syntax of the instruction is as follows:

“string1 to be concatenated” : “string2 to be concatenated” : ... : “stringN to be concatenated”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
CONCAT	_Kg: _Lb: _Gr	UOM_concat	StringType

In the above example, the strings contained in the Kg, _Lb, _Gr variables are concatenated in a single string; the string resulting from the concatenation is set in the UOM_concat destination variable.

In the event that the instruction refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

ii) LEFT – returns a string containing a specified number of characters starting from the left of the string. The syntax of the instruction is as follows:

“string on which to perform the operation” : “number of characters to be returned starting from the left of the string”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
LEFT	UOM_concat: LeftLength	UOM_left	StringType

In the above example, a portion of a string is extrapolated which contains a number of characters specified in the LeftLength variable starting from the left of the string contained in the UOM_concat variable, the resulting portion of string is set in the String-type Uom_left destination variable.

In the event that the instruction refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

iii) RIGHT – returns a string containing a specified number of characters starting from the right of the string. The syntax of the instruction is as follows:

“string on which to perform the operation” : “number of characters to be returned starting from the right of the string”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
RIGHT	UOM_concat: RightLength	UOM_right	StringType

In the above example, a portion of a string is extrapolated which contains a number of characters specified in the RightLength variable starting from the right of the string contained in the UOM_concat variable; the resulting portion of string is set in the String-type Uom_right destination variable.

In the event that the instruction refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

iv) LEN – calculates the length of the string

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
LEN	UOM_concat	UOM_length	IntType

In the above example, the length of the string contained in the UOM_concat variable is calculated; the result is set in the Int-type Uom_length destination variable.

In the event that the instruction refers to variables that are not defined or the variable is not String-type, an error will be generated in the log file and the execution of the formula will be interrupted.

v) FIND – performs a search for a sub-string within a string and returns the position in which it finds the first occurrence of the sub-string. The syntax of the instruction is as follows:

“sub-string to be searched for” : “string in which to perform the search”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
FIND	_Lb: UOM_concat	FIND_pos	IntType

In the above example, the string contained in the _Lb variable is searched for in the string contained in the UOM_concat variable, the position in which it finds the first occurrence of the _Lb string is set in the Int-type FIND_pos destination variable.

In the event that the instruction refers to variables that are not defined or if the variables are not String-type, an error will be generated in the log file and the execution of the formula will be interrupted.

vi) SLICE – allows the user to extrapolate a portion of string. The portion to be extracted is identified by means of a start index and an end index, which will be used for determining the resulting sub-string. The syntax of the instruction is as follows:

“string on which to perform the operation” : “start index” : “end index”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
SLICE	UOM_concat: LeftLength: RightLength	UOM_slice	StringType

In the above example, a portion of a string is extrapolated starting from the initial position contained in the LeftLength variable until the final position contained in the RightLength variable, the resulting portion of string is set in the String-type Uom_left destination variable.

In the event that the instruction refers to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

vii) REPLACE – this function allows the user to perform replacements of sub-strings within a string. The syntax of the instruction is as follows:

“sub-string to be replaced” : “new sub-string” : “string on which to perform the replacement”

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
REPLACE	_Lb: _Kg: UOM_concat	REPLACE_res	StringType

In the above example, in the string contained in the UOM_concat variable all the occurrences of the string contained in the _Lb variable are replaced with the string contained in the _Kg variable; the resulting string is set in the String-type REPLACE_res destination variable.

In the event that the instruction refers to variables that are not string-type or to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

viii) UPPER – the function allows the user to bring to uppercase all the characters of the string.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
UPPER	UOM_concat	UPPER_res	StringType

In the above example, in the string contained in the UOM_concat variable, all the characters in the string are brought to uppercase; the resulting string is set in the String-type UPPER_res destination variable.

In the event that the instruction refers to variables that are not string-type or to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

ix) LOWER – the function allows the user to bring to lowercase all the characters of the string.

Below is an example of a statement:

ACTION	SOURCE	DESTINATION	TYPE
LOWER	UOM_concat	LOWER _res	StringType

In the above example, in the string contained in the UOM_concat variable, all the characters in the string are brought to lowercase; the resulting string is set in the String-type LOWER_res destination variable.

In the event that the instruction refers to variables that are not string-type or to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

NOTE: in the expressions associated with operators of this type, it is not possible to use constant values. These may only be used after having assigned a constant value to a variable.

5) OPERATORS FOR INSTRUCTIONS FOR A CONDITIONAL JUMP

i) SKIP – allows the user to perform a conditional jump of a certain number of statements ahead if the result of a given condition is true. The syntax of the instruction is as follows:

“condition to be evaluated” : “number of statements to be skipped”

Below is an example of a statement:

ACTION	**SOURCE**	**DESTINATION**	**TYPE**
SKIP	UoM_value == KgValue: __skip__		

In the above example, if the expression evaluated in the SOURCE column results as true, it will skip the number of rows contained in the __skip__ variable, while if the condition is false, the interpreter will continue with the execution of the following statement.

In the event that the instruction refers to variables that are not Int-type or to variables that are not defined, or if it contains syntax errors, an error will be generated in the log file and the execution of the formula will be interrupted.

ii) EXIT – allows the user to interrupt the execution of the formula if a given condition is true. This instruction also makes it possible to perform a dump of the content of the namespaces before the interruption, depending on the value set in the exit variable. The syntax of the instruction is as follows:

“condition to be evaluated” : “exit variable”

If the exit variable is set on 1, ELENA interrupts the execution and carries out a dump of the content of the namespaces into the log file. In particular, the information printed is the following:

- Source namespace (input or output record being processed before it is modified by the statements of the formula)
- Formula namespace (variables contained in the namespace of the formula)
- Result namespace (input or output record after having applied the formula)
- Statement row (contained in the last statement executed at the moment of the EXIT)

Below is an example of a statement:

ACTION	**SOURCE**	**DESTINATION**	**TYPE**
EXIT	UoM_value == KgValue: __exit__1		

In the above example, if the expression evaluated in the SOURCE column results as true, the execution of the formula will be interrupted. If the content of the __exit__1 exit variable is “1”, the dump of the namespaces in the log file will be performed, if __exit__1 is “0”, the execution will only be interrupted.

> **NOTE**: in the expressions associated with operators of this type, it is not possible to use constant values in explicit mode. All constant values must be assigned in a variable by means of the DIM instruction before being used.

NOTE: the directive to be set in the output columns (in the cata_headers.xml and spec_headers.xml files) to be compiled by means of the formulas is “Default” (only in the absence of a different directive), with the corresponding default value set when the cell is empty. As an alternative, it is also possible to set a Default value to be used subsequently within the formulas.

4 Standard mathematical syntax

Below is a list of the possible operators, functions and constants that can be used within the MATH instruction:

- **sin**: returns the value of the sine of a variable expressed in radians or in degrees,
- **cos**: returns the value of the cosine of a variable expressed in radians or in degrees,
- **tan**: returns the value of the tangent of a variable expressed in radians or in degrees,
- **asin**: returns the value of the inverse sine expressed in radians or in degrees of a variable,
- **acos**: returns the value of the inverse cosine expressed in radians or in degrees of a variable,
- **atan**: returns the value of the inverse tangent expressed in radians or in degrees of a variable,
- **sqrt**: returns the square root of a variable,
- **log**: returns the logarithm of a variable,
- **abs**: returns the absolute value of a variable,
- **ceil**: returns the upper value of a decimal integer,
- **floor**: returns the lower value of a decimal integer,
- **round**: returns a rounding off with a defined number of decimals after the point given a decimal integer,
- **- (or neg):** returns the negative value of a number,
- **exp:** returns the exponential of a number,
- **+ (or add):** performs the addition of two numbers,
- **- (or sub):** performs the subtraction of two numbers,
- *** (or mul):** performs the multiplication of two numbers,
- **/ (or div):** performs the division of two numbers,
- **% (or mod):** performs the remainder after division of a number,
- **^ (or pow):** returns the power of a number,
- **, (or append):** performs the union of two strings or characters,
- **|| (or concat):** performs the concatenation of two strings,
- **random:** is a generator of random variables,
- **min:** returns the minimum of two numbers,
- **max**: returns the maximum of two numbers,
- **fac**: performs the factorial of a number,
- **pyt**: returns the square root of the sum of the square of two numbers,
- **atan2**: returns the inverse tangent expressed in radians of a fractional number.

The constant values allowed must, however, be expressed as follows:

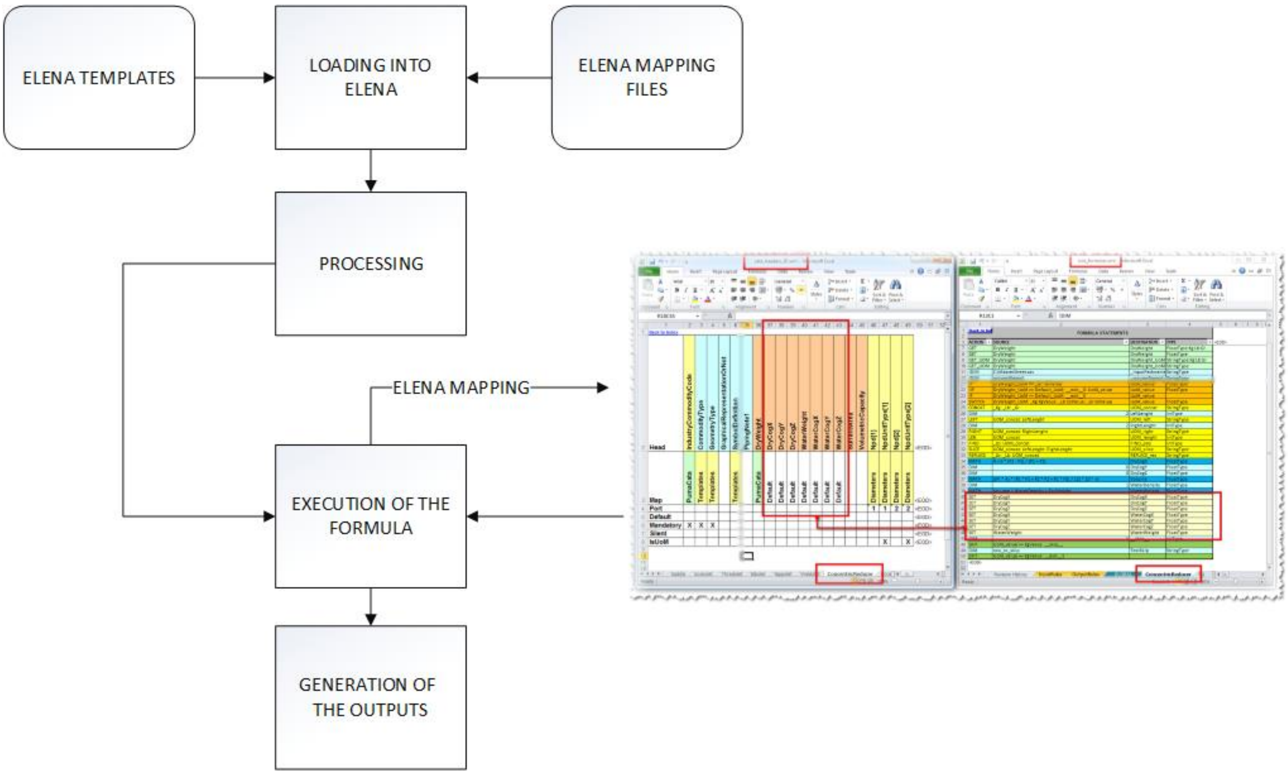
- **e** = 2.718281828459045 for the value of the Euler number
- **PI** = 3.141592653589793 for the value of Pi

5 Example of a working case

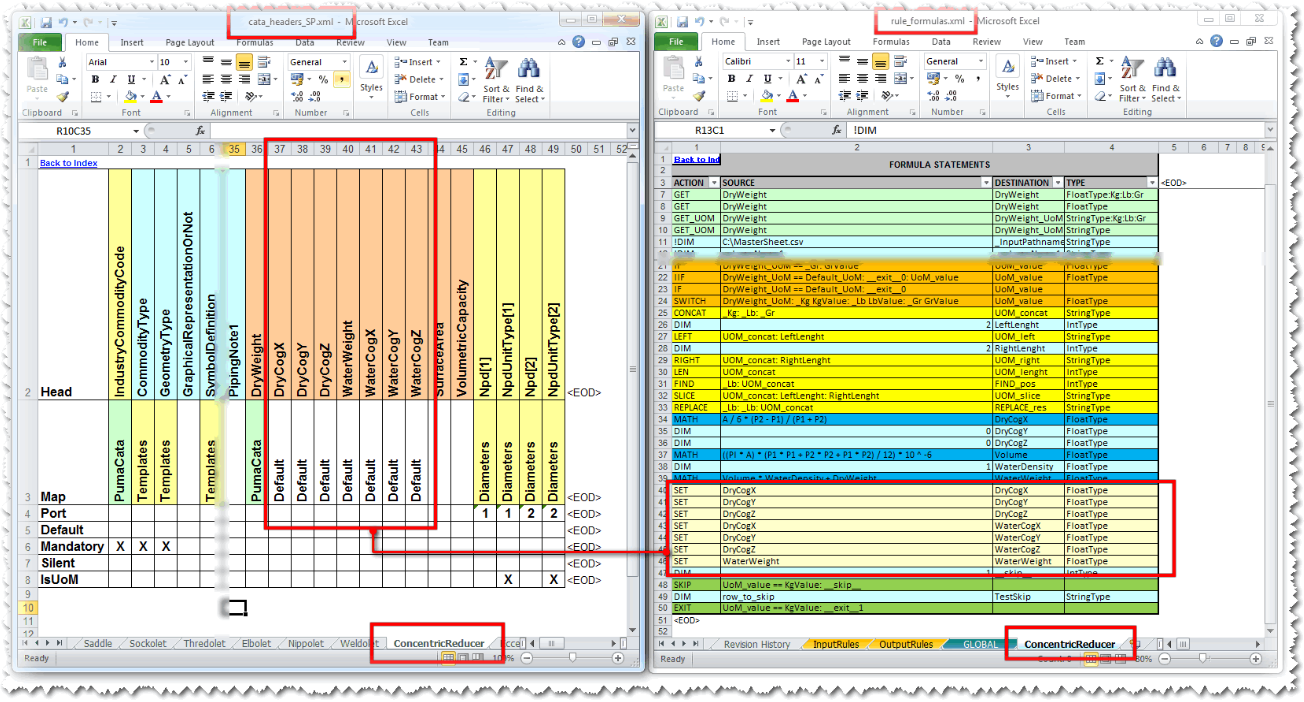
This paragraph details as an example the case of the set of additional values that are necessary in SP3D for calculating the center of gravity:

- **DryCogX**: x coordinate of the center of gravity in the case of an empty component
- **DryCogY**: y coordinate of the center of gravity in the case of an empty component
- **DryCogZ**: z coordinate of the center of gravity in the case of an empty component
- **Weight:** weight of the empty component
- **WaterCogX**: x coordinate of the center of gravity in the case of an object in use, but filled with water instead of the fluid it should contain
- **WaterCogY:** y coordinate of the center of gravity in the case of an object in use, but filled with water instead of the fluid it should contain
- **WaterCogZ:** z coordinate of the center of gravity in the case of an object in use, but filled with water instead of the fluid it should contain
- **WaterWeight:** weight of the component in use, but filled with water instead of the fluid it should contain

Below is a diagram explaining the processing flow.



For greater clarity, below is a detail of the sample mapping for the ConcentricReducer object.



	1	2	3	4	5	6	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	Back to Index																					
2	Head	IndustryCommodityCode	CommodityType	GeometryType	GraphicalRepresentationOrNot	SymbolDefinition	PipingNote1	DryWeight	DryCogX	DryCogY	DryCogZ	WaterWeight	WaterCogX	WaterCogY	WaterCogZ	SurfaceArea	VolumetricCapacity	Npd[1]	NpdUnitType[1]	Npd[2]	NpdUnitType[2]	<EOD>
3	Map	PumaCata	Templates	Templates		Templates		PumaCata	Default	Default	Default	Default	Default	Default	Default			Diameters	Diameters	Diameters	Diameters	<EOD>
4	Port																	1	1	2	2	<EOD>
5	Default																					<EOD>
6	Mandatory	X	X	X																		<EOD>
7	Silent																					<EOD>
8	IsUoM																		X		X	<EOD>

FORMULA STATEMENTS

	1	2	3	4	5
1	Back to Ind				
3	ACTION	SOURCE	DESTINATION	TYPE	<EOD>
7	GET	DryWeight	DryWeight	FloatType:Kg:Lb:Gr	
8	GET	DryWeight	DryWeight	FloatType	
9	GET_UOM	DryWeight	DryWeight_UoM	StringType:Kg:Lb:Gr	
10	GET_UOM	DryWeight	DryWeight_UoM	StringType	
11	!DIM	C:\MasterSheet.csv	_InputPathname	StringType	
21	IF	DryWeight_UoM == _Gr: GrValue	UoM_value	FloatType	
22	IIF	DryWeight_UoM == Default_UoM: __exit__0: UoM_value	UoM_value	FloatType	
23	IF	DryWeight_UoM == Default_UoM: __exit__0	UoM_value		
24	SWITCH	DryWeight_UoM: _Kg KgValue: _Lb LbValue: _Gr GrValue	UoM_value	FloatType	
25	CONCAT	_Kg: _Lb: _Gr	UOM_concat	StringType	
26	DIM	2	LeftLenght	IntType	
27	LEFT	UOM_concat: LeftLenght	UOM_left	StringType	
28	DIM	2	RightLenght	IntType	
29	RIGHT	UOM_concat: RightLenght	UOM_right	StringType	
30	LEN	UOM_concat	UOM_lenght	IntType	
31	FIND	_Lb: UOM_concat	FIND_pos	IntType	
32	SLICE	UOM_concat: LeftLenght: RightLenght	UOM_slice	StringType	
33	REPLACE	_Lb: _Lb: UOM_concat	REPLACE_res	StringType	
34	MATH	A / 6 * (P2 - P1) / (P1 + P2)	DryCogX	FloatType	
35	DIM	0	DryCogY	FloatType	
36	DIM	0	DryCogZ	FloatType	
37	MATH	((PI * A) * (P1 * P1 + P2 * P2 + P1 * P2) / 12) * 10 ^ -6	Volume	FloatType	
38	DIM	1	WaterDensity	FloatType	
39	MATH	Volume * WaterDensity + DryWeight	WaterWeight	FloatType	
40	SET	DryCogX	DryCogX	FloatType	
41	SET	DryCogY	DryCogY	FloatType	
42	SET	DryCogZ	DryCogZ	FloatType	
43	SET	DryCogX	WaterCogX	FloatType	
44	SET	DryCogY	WaterCogY	FloatType	
45	SET	DryCogZ	WaterCogZ	FloatType	
46	SET	WaterWeight	WaterWeight	FloatType	
47	DIM	1	__skip__	IntType	
48	SKIP	UoM_value == KgValue: __skip__			
49	DIM	row_to_skip	TestSkip	StringType	
50	EXIT	UoM_value == KgValue: __exit__1			
51	<EOD>				