diligent
technology & business consulting

SweeterCalc Documentation

diligent technology & business consulting GmbH, Schumannstraße 56, 60325 Frankfurt am Main, Germany

Updated: *27.04.2016*
Version: *1.0*

diligent technology & business consulting GmbH
Schumannstraße 56
60325 Frankfurt am Main
Telefon +49 (0) 69/ 175 3660 - 0
Telefax +49 (0) 69/ 175 3660 - 99
E-Mail: info@dtbc.eu

Sitz der Gesellschaft:
Frankfurt am Main
Amtsgericht Frankfurt am Main
HRB 86980
Umsatzsteuer-ID: DE269831168
Geschäftsführer: Sascha Krause

Bankverbindungen:
Frankfurter Sparkasse Frankfurt/M.
IBAN: DE08500502010200566024
BIC/Swift-Code: HELADEF1822

Commerzbank Frankfurt/M.
IBAN: DE34500400000683517700
BIC/Swift-Code: COBADEFFXXX

Table of contents

diligent technology & business consulting GmbH
Schumannstraße 56
60325 Frankfurt am Main
Telefon +49 (0) 69/ 175 3660 - 0
Telefax +49 (0) 69/ 175 3660 - 99
E-Mail: info@dtbc.eu

Sitz der Gesellschaft:
Frankfurt am Main
Amtsgericht Frankfurt am Main
HRB 86980
Umsatzsteuer-ID: DE269831168
Geschäftsführer: Sascha Krause

Bankverbindungen:
Frankfurter Sparkasse Frankfurt/M.
IBAN: DE08500502010200566024
BIC/Swift-Code: HELADEF1822

Commerzbank Frankfurt/M.
IBAN: DE34500400000683517700
BIC/Swift-Code: COBADEFFXXX

1. Installation

SweeterCalc comes as an installable package for SuiteCRM, so its install procedure is identical to other packages.

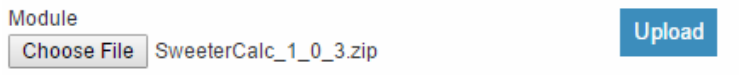
SweeterCalc is compatible with all SuiteCRM versions from 7.2.1 and every PHP versions SuiteCRM 7.2.1 and above compatible with.

The downloaded package contains the following files:

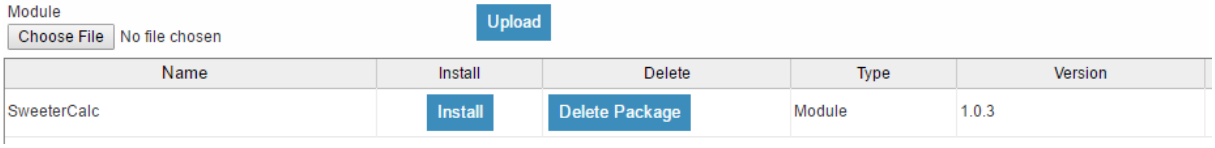
- **SweeterCalc_x_y_z.zip – installable SuiteCRM package (x, y and z is the version, e.g.: SweeterCalc_1_0_3.zip)**
- **SweeterCalc Documentation x.y.pdf – This file (x and y is the version)**

To install SweeterCalc go to the Admin page, and select the Module Loader menu entry.

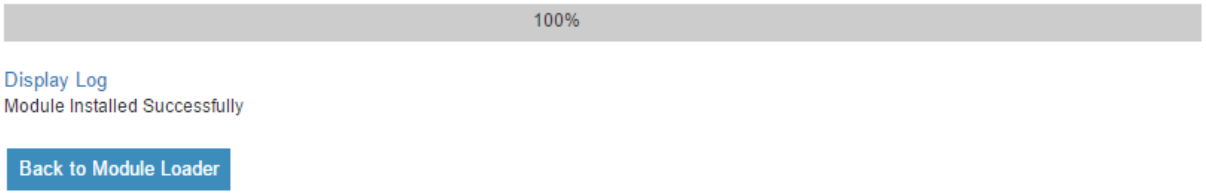
On the Module Loader page choose the SweeterCalc installable SuiteCRM package from the computer and push the Upload button.



After the upload process, the uploaded file with package information shows up in the bottom list view.



Now, push the Install button. On the next screen accept the license agreement and push the Commit button. After a while, SuiteCRM should show that the module is successfully installed.



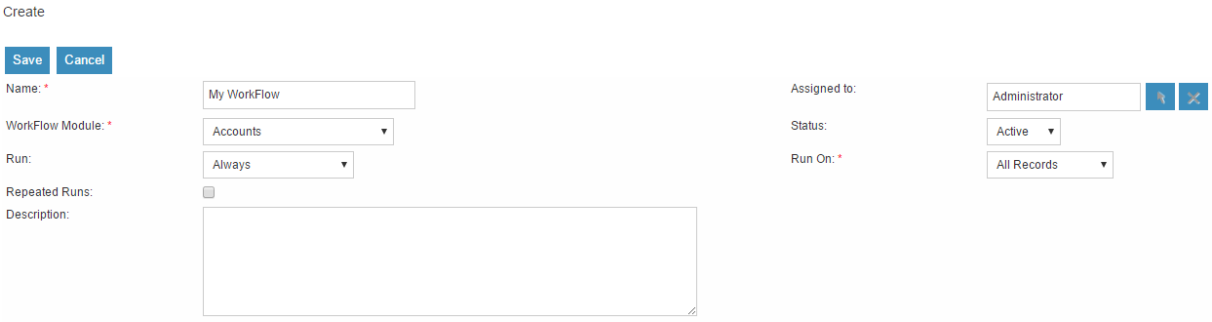
2. Creating WorkFlows

2.1 Using the WorkFlow engine

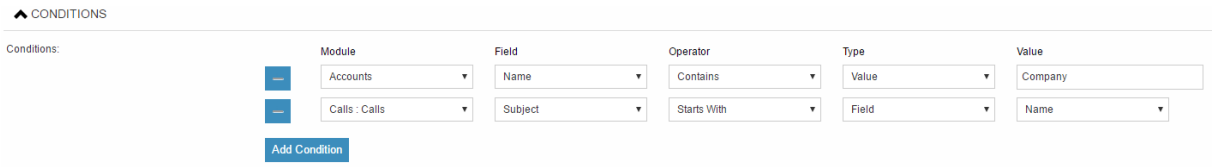
Since SweeterCalc is an extension of SuiteCRM's WorkFlow engine, the first steps of creating a SweeterCalc WorkFlow is identical to other WorkFlows.

First, go to the WorkFlow module and select Create to create a new WorkFlow. Now, fill all the basic fields similar to other WorkFlows:

- **Name: the name of the WorkFlow**
- **Assigned to: the user this WorkFlow belongs to**
- **WorkFlow module: the module this WorkFlow will work on**
- **Status: this option sets the WorkFlow as active (will operate) or inactive (will not operate)**
- **Run: with this option, the user can decide when the WorkFlow should operate. It can be only on save of the entity, when the scheduler processing it or both of them.**
- **Run on: the user can select on which records the WorkFlow should operate on. This can be only on new records, only on modified records or both of them.**
- **Repeated runs: if this checkbox is not checked, then one entity could be processed only once.**
- **Description: a description for this WorkFlow**

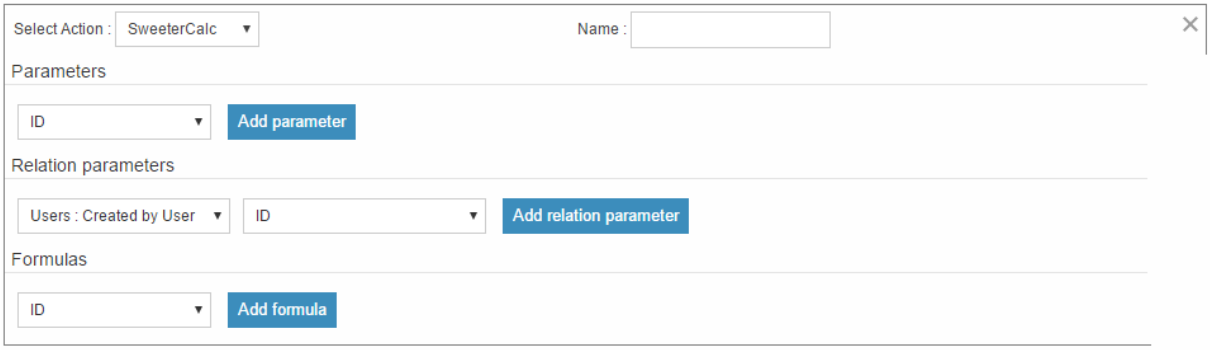


Now, add the desired conditions for the base module or for the related modules as for other WorkFlows.



2.2 Setting up a SweeterCalc action

After setting the basic data and conditions (if any), the user can proceed to the actions. Press the Add Action button to add a new action, and then select SweeterCalc from the Select Action dropdown.



After a second, the SweeterCalc user interface is loaded and looks like the picture above.

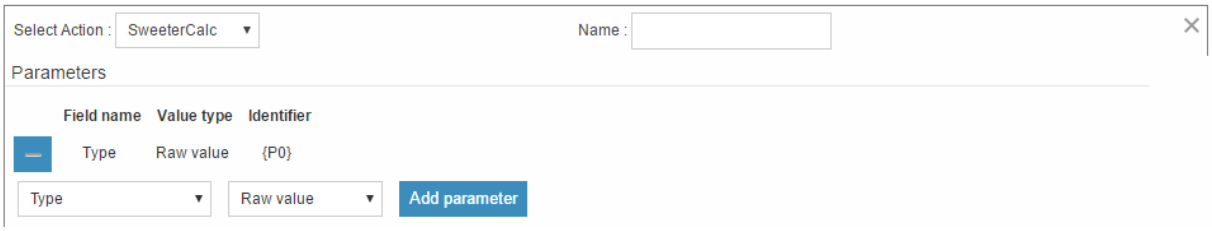
2.2.1 Adding parameters

It is possible to add parameters to the formulas by using the dropdown in the Parameters section of the SweeterCalc's user interface. The dropdown contains all of the (basic and custom) fields which belongs to the module selected in the basic fields section.

To add a parameter, select the field from the dropdown and click on the Add parameter button. After this action, a new line appears in the parameter table with the name of the field and the given identifier.

For some fields (dropdowns and multi-selects) an additional dropdown shown up where the user can select if the raw or the formatted value should be used in SweeterCalc. The raw format means the value which is stored in the database and the formatted value means the label for that database value.

To remove a parameter from the table, simply click on the minus button in the row of the parameter. Be aware, that if you remove a parameter, all of the identifiers are recalculated, so the identifiers could change for fields!



The identifier is used to reference this field when the user creates the formula. For example all appearances of the {P0} identifier will be replaced with the Account's name in the formula. All parameters are like {Px} where x is the sequential order of the parameter. The amount of the parameters is not limited.

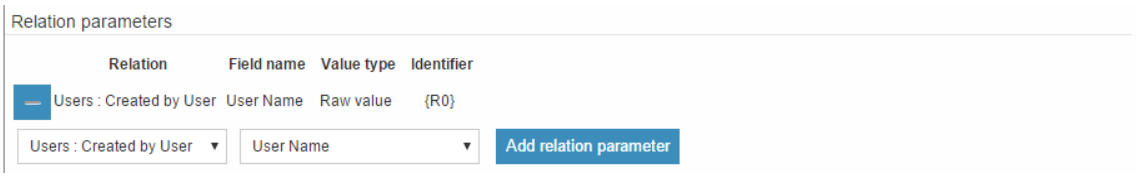
2.2.2 Adding relation parameters

Relation parameters are very similar to the regular parameters, the only difference is that the user first selects an entity which is in a one-to-one or one-to-many relationship with the actual entity.

To add a relation parameter, select the relation first, and then select the field from the connected entity and push the Add relation parameter button. After this action, a new line appears in the relation parameter table with the name of the relationship, the name of the field and the given identifier.

As for parameters for some relation parameter fields (dropdowns and multi-selects) an additional dropdown shown up where the user can select if the raw or the formatted value should be used in SweeterCalc.

To remove a relation parameter from the table, simply click on the minus button in the row of the relation parameter. Be aware, that if you remove a relation parameter, all of the identifiers are recalculated, so the identifiers could change for fields!



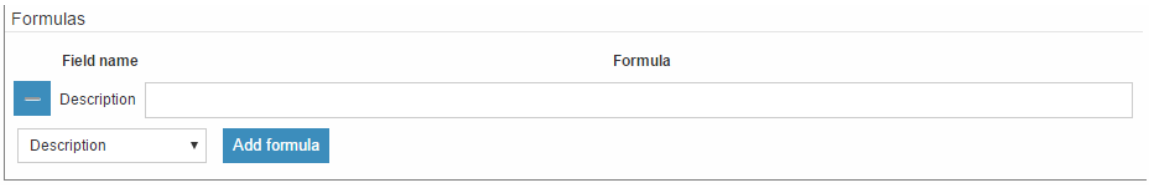
The identifier is used to reference this field when the user creates the formula. For example all appearances of the {R0} identifier will be replaced with the creator user's username in the formula. All relation parameters are like {Rx} where x is the sequential order of the relation parameter. The amount of the relation parameters is not limited.

2.2.3 Creating formula for a field

In the Formulas part of the user interface the user can add formulas for fields of the actual entity.

To add a formula, select a field from the dropdown first and then push the Add formula button. After this action, a new line appears in the formula table with the name of the field and with the place for the formula.

To remove a formula from the table, simply click on the minus button in the row of the formula.



The formula is a textbox where the user can write the formulas. The module evaluates the formula on the given time (on save, on scheduler run or both) and fills the selected field with the evaluated value.

The formula can contain any text (with full UTF-8 support), but only the function parts (functions with parameters between '{' and '}') are evaluated.

For example and with the parameters added in the previous sections, if we fill the formula like: Account {P0} created by user name {R0}, then the description field will have the following value after save: Account My Account created by user name MyUser (implying the account's name is My Account and the creator user's username is MyUser).

SweeterCalc has many built-in functions which allows the user to build complex formulas to achieve various goals. These functions are described in the next section.

3. Usable functions

As it is mentioned above, all of the functions are wrapped between '{' and '}' signs, and they look like {functionName(parameter1; parameter2; …)}. The count of the parameters are different for the different functions. The module evaluates the functions and changes them with their result in the formula.

The functions can be embedded into each other (using a result of a function as a parameter for another function) like in this example:

{power({subtract({divide({add({multiply(10; 2)}; 12)}; 8)}; 1)}; 2)}

This function is the formalized look of the following mathematical expression:

$$((((10 * 2) + 12) / 8) - 1)^2$$

The functions are divided to six groups. These groups are described in the next section of the document.

3.1 Logical functions

Logical functions are returning true or false in the form of 1 and 0 so checkboxes typed fields can be filled with these functions. They can be also used as the logical condition for the ifThenElse function.

3.1.1 equal

Signature	{equal(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** equals with **parameter2**
Returns	1 if the two parameters are equal or 0 if not
Example call	{equal(1; 2)} returns 0

3.1.2 notEqual

Signature	{notEqual(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** not equals with **parameter2**
Returns	0 if the two parameters are equal or 1 if not
Example call	{notEqual(1; 2)} returns 1

3.1.3 greaterThan

Signature	{greaterThan(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** greater than **parameter2**
Returns	1 if **parameter1** greater than **parameter2**, 0 if not
Example call	{greaterThan(3; 3)} returns 0

3.1.4 greaterThanOrEqual

Signature	{greaterThanOrEqual(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** greater than or equal **parameter2**
Returns	1 if **parameter1** greater than or equal **parameter2**, 0 if not
Example call	{greaterThanOrEqual(3; 3)} returns 1

3.1.5 lessThan

Signature	{lessThan(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** less than **parameter2**
Returns	1 if **parameter1** less than **parameter2**, 0 if not
Example call	{lessThan(3; 3)} returns 0

3.1.6 lessThanOrEqual

Signature	{lessThanOrEqual(parameter1; parameter2)}
Parameters	parameter1: can be any value of any type
	parameter2: can be any value of any type
Description	Determines if **parameter1** less than or equal **parameter2**
Returns	1 if **parameter1** less than or equal **parameter2**, 0 if not
Example call	{lessThanOrEqual(3; 3)} returns 1

3.1.7 empty

Signature	{empty(parameter)}
Parameters	parameter: text value
Description	Determines if **parameter** is empty
Returns	1 if **parameter** is empty, 0 if not
Example call	{empty(any text)} returns 0

3.1.8 notEmpty

Signature	{notEmpty(parameter)}
Parameters	parameter: text value
Description	Determines if **parameter** is not empty
Returns	1 if **parameter** is not empty, 0 if empty
Example call	{notEmpty(any text)} returns 1

3.1.9 not

Signature	{not(parameter)}
Parameters	parameter: logical value
Description	Negates the logical value of the **parameter**
Returns	1 if **parameter** is 0, 0 if **parameter** is 1
Example call	{not(0)} returns 1

3.1.10 and

Signature	{and(parameter1; parameter2)}
Parameters	parameter1: logical value
	parameter2: logical value
Description	Applies the AND logical operator to two logical values
Returns	1 if **parameter1** and **parameter2** is 1, 0 if any parameters are 0
Example call	{and(1; 0)} returns 0

3.1.11 or

Signature	{or(parameter1; parameter2)}
Parameters	parameter1: logical value
	parameter2: logical value
Description	Applies the OR logical operator to two logical values
Returns	1 if **parameter1** or **parameter2** is 1, 0 if both parameters are 0
Example call	{or(1; 0)} returns 1

3.2 Text functions

Text functions are used to manipulate text in various ways. All the functions listed here are fully supports UTF-8 texts, so special characters should not raise any problems.

3.2.1 substring

Signature	{substring(text; start; length)}
Parameters	text: text value
	start: decimal value
	length [optional parameter]: decimal value
Description	Cuts the substring of a text field from **start**. If the **length** optional parameter is not set, then it cuts all characters until the end of the string, otherwise cuts the provided **length**. Indexing of a text's characters starting from 0.
Returns	Substring of the given text
Example call	{substring(This is my text; 5)} returns is my text
	{substring(This is my text; 5; 5)} returns is my

3.2.2 length

Signature	{length(parameter)}
Parameters	parameter: text value
Description	Count the characters in a text.
Returns	The count of the characters in a text.
Example call	{length(sample text)} returns 11

3.2.3 replace

Signature	{replace(search; replace; subject)}
Parameters	search: text value
	replace: text value
	subject: text value
Description	Replace all occurrences of **search** to **replace** in the text **subject**.
Returns	**subject** with replaced values.
Example call	{replace(apple; orange; This is an apple tree)} returns This is an orange tree

3.2.4 position

Signature	{position(subject; search)}
Parameters	subject: text value
	search: text value
Description	Find position of first occurrence of **search** in a **subject**
Returns	Numeric position of **search** in **subject** or -1 if **search** not present in **subject**
Example call	{position(Where is my text?; text)} returns 12

3.2.5 lowercase

Signature	{lowercase(parameter)}
Parameters	parameter: text value
Description	Make text lowercase
Returns	The lowercased text.
Example call	{lowercase(ThIs iS a sAmPlE tExT)} returns this is a sample text

3.2.6 uppercase

Signature	{uppercase(parameter)}
Parameters	parameter: text value
Description	Make text uppercase
Returns	The uppercased text.
Example call	{uppercase(ThIs iS a sAmPlE tExT)} returns THIS IS A SAMPLE TEXT

3.3 Mathematical functions

Mathematical functions are used to manipulate numbers in various ways. Several mathematical operators are implemented as functions in SweeterCalc.

3.3.1 add

Signature	{add(parameter1; parameter2)}
Parameters	parameter1: number value
	parameter2: number value
Description	Adds **parameter1** and **parameter2**
Returns	The sum of **parameter1** and **parameter2**
Example call	{add(3.12; 4.83)} returns 7.95

3.3.2 subtract

Signature	{subtract(parameter1; parameter2)}
Parameters	parameter1: number value
	parameter2: number value
Description	Subtracts **parameter2** from **parameter1**
Returns	The distinction of **parameter2** and **parameter1**
Example call	{subtract(8; 3)} returns 5

3.3.3 multiply

Signature	{multiply(parameter1; parameter2)}
Parameters	parameter1: number value
	parameter2: number value
Description	Multiplies **parameter1** and **parameter2**
Returns	The product of **parameter1** and **parameter2**
Example call	{multiply(2; 4)} returns 8

3.3.4 divide

Signature	{divide(parameter1; parameter2)}
Parameters	parameter1: number value
	parameter2: number value
Description	Divides **parameter2** with **parameter1**
Returns	The division of **parameter2** and **parameter1**
Example call	{divide(8; 2)} returns 4

3.3.5 power

Signature	{power(parameter1; parameter2)}
Parameters	parameter1: number value
	parameter2: number value
Description	Raises **parameter1** to the power of **parameter2**
Returns	**parameter1** raised to the power of **parameter2**
Example call	{power(2; 7)} returns 128

3.3.6 squareRoot

Signature	{squareRoot(parameter)}
Parameters	parameter: number value
Description	Calculates the square root of **parameter**
Returns	The square root of **parameter**
Example call	{squareRoot(4)} returns 2

3.3.7 absolute

Signature	{absolute(parameter)}
Parameters	parameter: number value
Description	Calculates the absolute value of **parameter**
Returns	The absolute value of **parameter**
Example call	{absolute(-4)} returns 4

3.4 Date functions

There are several date functions implemented in SweeterCalc, so the user can manipulate dates in many ways.

Most of the functions uses a format parameter, which is used to set the result of the functions formatted as the user wants to. The options for these formats are equivalent with the PHP format parameters:

Format character	Description	Example returned values
For day		
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l	A full textual representation of the day of the week	Sunday through Saturday
N	ISO-8601 numeric representation of the day of the week	1 (for Monday) through 7 (for Sunday)
S	English ordinal suffix for the day of the month, 2 characters	st, nd, rd or th. Works well with j
w	Numeric representation of the day of the week	0 (for Sunday) through 6 (for Saturday)
z	The day of the year (starting from 0)	0 through 365
For week		
W	ISO-8601 week number of year, weeks starting on Monday	42 (the 42nd week in the year)
For month		
F	A full textual representation of a month, such as January or March	January through December
m	Numeric representation of a month, with leading zeros	01 through 12
M	A short textual representation of a month, three letters	Jan through Dec
n	Numeric representation of a month, without leading zeros	1 through 12
t	Number of days in the given month	28 through 31
For year		
L	Whether it's a leap year	1 if it is a leap year, 0 otherwise

o	ISO-8601 year number. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, that year is used instead	1999 or 2003
Y	A full numeric representation of a year, 4 digits	1999 or 2003
y	A two digit representation of a year	99 or 03
For time		
a	Lowercase Ante meridiem and Post meridiem	am or pm
A	Uppercase Ante meridiem and Post meridiem	AM or PM
B	Swatch Internet time	000 through 999
g	12-hour format of an hour without leading zeros	1 through 12
G	24-hour format of an hour without leading zeros	0 through 23
h	12-hour format of an hour with leading zeros	01 through 12
H	24-hour format of an hour with leading zeros	00 through 23
i	Minutes with leading zeros	00 to 59
s	Seconds, with leading zeros	00 through 59
For timezone		
e	Timezone identifier	UTC, GMT, Atlantic/Azores
I	Whether or not the date is in daylight saving time	1 if Daylight Saving Time, 0 otherwise
O	Difference to Greenwich time (GMT) in hours	+0200
P	Difference to Greenwich time (GMT) with colon between hours and minutes	+02:00
T	Timezone abbreviation	EST, MDT
Z	Timezone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	-43200 through 50400
For full date/time		
c	ISO 8601 date	2004-02-12T15:19:21+00:00
r	RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
U	Seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)	

To set a date field with SweeterCalc you must use the Y-m-d format, to set a datetime field, you must use the Y-m-d H:i:s format.

All date functions can parse user given timestamps only in predefined formats. These predefined formats are equivalent with the formats the PHP DateTime object can parse:

Description	Format	Examples
Localized Notations		
American month and day	mm "/" dd	"5/12", "10/27"
American month, day and year	mm "/" dd "/" y	"12/22/78", "1/17/2006"
Four digit year, month and day with slashes	YY "/" mm "/" dd	"2008/6/30", "1978/12/22"
Four digit year and month (GNU)	YY "-" mm	"2008-6", "2008-06"
Year, month and day with dashes	y "-" mm "-" dd	"2008-6-30", "78-12-22"
Day, month and four digit year, with dots, tabs or dashes	dd [.\t-] mm [.-] YY	"30-6-2008", "22.12.1978"
Day, month and two digit year, with dots or tabs	dd [.\t] mm "." yy	"30.6.08", "22\t12.78"
Day, textual month and year	dd ([\t.-])* m ([\t.-])* y	"30-June 2008", "22DEC78", "14 III 1879"
Textual month and four digit year (Day reset to 1)	m ([\t.-])*	"June 2008", "DEC1978", "March 1879"
Four digit year and textual month (Day reset to 1)	YY ([\t.-])* m	"2008 June", "1978-XII", "1879.MArCH"
Textual month, day and year	m ([.\t-])* dd [,.stndrh\t]+ y	"July 1st, 2008", "May.9,78"
Textual month and day	m ([.\t-])* dd [,.stndrh\t]*	"July 1st,", "Apr 17", "May.9"
Day and textual month	d ([.\t-])* m	"1 July", "17 Apr", "9.May"
Month abbreviation, day and year	M "-" DD "-" y	"May-09-78", "Apr-17-1790"
Year, month abbreviation and day	y "-" M "-" DD	"78-Dec-22"
Year (and just the year)	YY	"1978", "2008"
Textual month (and just the month)	m	"March", "jun", "DEC"
ISO8601 Notations		
Eight digit year, month and day	YY MM DD	"15810726", "19780417"
Four digit year, month and day with slashes	YY "/" MM "/" DD	"2008/06/30", "1978/12/22"
Two digit year, month and day with dashes	yy "-" MM "-" DD	"08-06-30", "78-12-22"
Four digit year with optional sign, month and day	[+-]? YY "-" MM "-" DD	"-0002-07-26", "+1978-04-17", "1814-05-17"

For all functions without timestamp parameter, we assume that the current date/time is 2016.04.29. 15:08:03

3.4.1 date

Signature	{date(format; timestamp)}
Parameters	format: format text
	timestamp: date/time value
Description	Creates a date in the given format
Returns	**timestamp** in the given **format**
Example call	{date(ymd; 2016-02-11)} returns 160211

3.4.2 now

Signature	{now(format)}
Parameters	format: format text
Description	Creates the actual date/time in the given format
Returns	Current date/time in the given **format**
Example call	{now(Y-m-d H:i:s)} returns 2016-04-29 15:08:03

3.4.3 yesterday

Signature	{yesterday(format)}
Parameters	format: format text
Description	Creates yesterday's date/time in the given format
Returns	Yesterday's date/time in the given **format**
Example call	{yesterday(Y-m-d H:i:s)} returns 2016-04-28 15:08:03

3.4.4 tomorrow

Signature	{tomorrow(format)}
Parameters	format: format text
Description	Creates tomorrow's date/time in the given format
Returns	Tomorrow's date/time in the given **format**
Example call	{tomorrow(Y-m-d H:i:s)} returns 2016-04-30 15:08:03

3.4.5 datediff

Signature	{datediff(timestamp1; timestamp2; unit)}
Parameters	timestamp1: date/time value
	timestamp2: date/time value
	unit: years/months/days/hours/minutes/seconds; default: days
Description	Subtracts **timestamp2** from **timestamp1**
Returns	The difference between the two dates returned in **unit**
Example call	{datediff(2016-02-01; 2016-04-22; days)} returns 81

3.4.6 addYears

Signature	{addYears(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** years to **timestamp**
Returns	Incremented date in **format**
Example call	{addYears(Ymd; 2016-04-22; 1)} returns 20170422

3.4.7 addMonths

Signature	{addMonths(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** months to **timestamp**
Returns	Incremented date in **format**
Example call	{addMonths(Ymd; 2016-04-22; 1)} returns 20160522

3.4.8 addDays

Signature	{addDays(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** days to **timestamp**
Returns	Incremented date in **format**
Example call	{addDays(Ymd; 2016-04-22; 1)} returns 20160423

3.4.9 addHours

Signature	{addHours(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** hours to **timestamp**
Returns	Incremented date in **format**
Example call	{addHours(Ymd H:i:s; 2016-04-22 23:30; 5)} returns 20160423 04:30:00

3.4.10 addMinutes

Signature	{addMinutes(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** minutes to **timestamp**
Returns	Incremented date in **format**
Example call	{addMinutes(Ymd H:i:s; 2016-04-22 22:58; 5)} returns 20160422 23:03:00

3.4.11 addSeconds

Signature	{addSeconds(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Adds **amount** seconds to **timestamp**
Returns	Incremented date in **format**
Example call	{addSeconds(Ymd H:i:s; 2016-04-22 22:58; 5)} returns 20160422 22:58:05

3.4.12 subtractYears

Signature	{subtractYears(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** years from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractYears(Ymd; 2016-04-22; 5)} returns 20110422

3.4.13 subtractMonths

Signature	{subtractMonths(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** months from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractMonths(Ymd; 2016-04-22; 5)} returns 20151122

3.4.14 subtractDays

Signature	{subtractDays(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** days from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractDays(Ymd; 2016-04-22; 5)} returns 20160417

3.4.15 subtractHours

Signature	{subtractHours(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** hours from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractHours(Ymd H:i:s; 2016-04-22 12:37; 5)} returns 20160422 07:37:00

3.4.16 subtractMinutes

Signature	{subtractMinutes(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** minutes from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractMinutes(Ymd H:i:s; 2016-04-22 12:37; 5)} returns 20160422 12:32:00

3.4.17 subtractSeconds

Signature	{subtractSeconds(format; timestamp; amount)}
Parameters	format: format text
	timestamp: date/time value
	amount: decimal number
Description	Subtracts **amount** minutes from **timestamp**
Returns	Decremented date in **format**
Example call	{subtractSeconds(Ymd H:i:s; 2016-04-22 12:37; 5)} returns 20160422 12:36:55

3.5 Control functions

There is only one control function implemented in SweeterCalc so far, but this function ensures that the user can write very complex formulas with conditions.

Since the functions can be embedded in each other, the user can write junctions with many branches.

3.5.1 ifThenElse

Signature	{ifThenElse(condition; trueBranch; falseBranch)}
Parameters	condition: logical value
	trueBranch: any expression
	falseBranch: any expression
Description	Selects one of the two branches depending on **condition**
Returns	**trueBranch** if **condition** is true, **falseBranch** otherwise
Example call	{ifThenElse({equal(1; 1)}; 1 equals 1; 1 not equals 1)} returns 1 equals 1

3.6 Counters

There are several counters implemented in SweeterCalc which can be used in various scenarios.

The counters sorted into two groups:

1. **Global counters: Counters which are incremented every time an affected formula is evaluated**
2. **Daily counters: Counters which resets every day. (Starting from 1)**

In this chapter we assume that the counters current value is 4, so the incremented value will be 5 with the given format.

3.6.1 GlobalCounter

Signature	{GlobalCounter(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** with length **numberLength**
Returns	Counter with length **numberLength**
Example call	{GlobalCounter(myName; 4)} returns 0005

3.6.2 GlobalCounterPerUser

Signature	{GlobalCounterPerUser(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the user who creates the entity with length **numberLength**
Returns	Counter with length **numberLength**
Example call	{GlobalCounterPerUser(myName; 3)} returns 005

3.6.3 GlobalCounterPerModule

Signature	{GlobalCounterPerModule(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the module of the entity with length **numberLength**
Returns	Counter with length **numberLength**
Example call	{GlobalCounterPerModule(myName; 2)} returns 05

3.6.4 GlobalCounterPerUserPerModule

Signature	{GlobalCounterPerUserPerModule(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the user who creates the entity and for the module of the entity with length **numberLength**
Returns	Counter with length **numberLength**
Example call	{GlobalCounterPerUserPerModule(myName; 1)} returns 5

3.6.5 DailyCounter

Signature	{DailyCounter(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** with length **numberLength**
Returns	Counter with length **numberLength**, or if the counter is not incremented this day then 1 with length **numberLength**
Example call	{DailyCounter(myName; 1)} returns 5

3.6.6 DailyCounterPerUser

Signature	{DailyCounterPerUser(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the user who creates the entity with length **numberLength**
Returns	Counter with length **numberLength**, or if the counter is not incremented this day for this user then 1 with length **numberLength**
Example call	{DailyCounter(myName; 1)} returns 5

3.6.7 DailyCounterPerModule

Signature	{DailyCounterPerModule(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the module of the entity with length **numberLength**
Returns	Counter with length **numberLength**, or if the counter is not incremented this day for this module then 1 with length **numberLength**
Example call	{DailyCounterPerModule(myName; 1)} returns 5

3.6.8 DailyCounterPerUserPerModule

Signature	{DailyCounterPerUserPerModule(name; numberLength)}
Parameters	name: any text
	numberLength: decimal number
Description	Increments and returns the counter for **name** for the user who creates the entity and for the module of the entity with length **numberLength**
Returns	Counter with length **numberLength**, or if the counter is not incremented this day for the user who creates the entity and for this module then 1 with length **numberLength**
Example call	{DailyCounterPerUserPerModule(myName; 1)} returns 5

4. Examples

In this section we provide examples for some use cases.

4.1 Calculate monthly fee for an opportunity

4.1.1 Use case

The user would like to calculate a monthly fee of an opportunity to a custom field by dividing the amount of the opportunity by the duration.

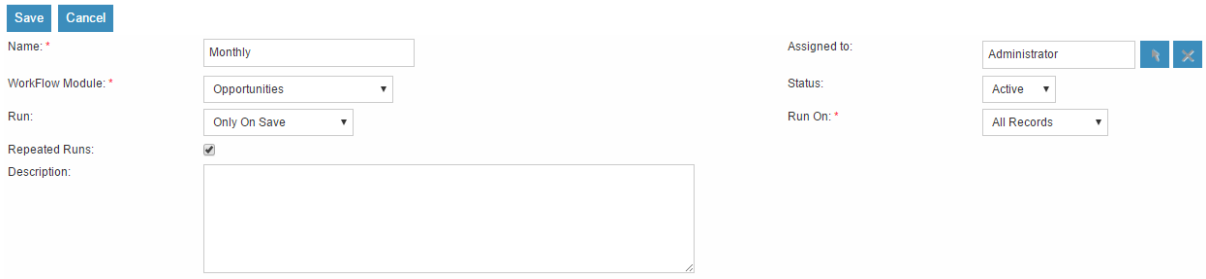
4.1.2 Setup

Our opportunities module has a dropdown field called Duration with values: (database value in brackets) 6 months [6], 1 year [12], 2 years [24]. There is also a currency field called Monthly.

4.1.3 The WorkFlow

Go to WorkFlow module and create a new WorkFlow. Set the base options like the following:

Name: as you wish	**WorkFlow Module**: Opportunities
Status: Active	**Run**: Only on save
Run on: All records	**Repeated runs**: checked



We do not create any conditions, since we would like the WorkFlow to run on all opportunities.

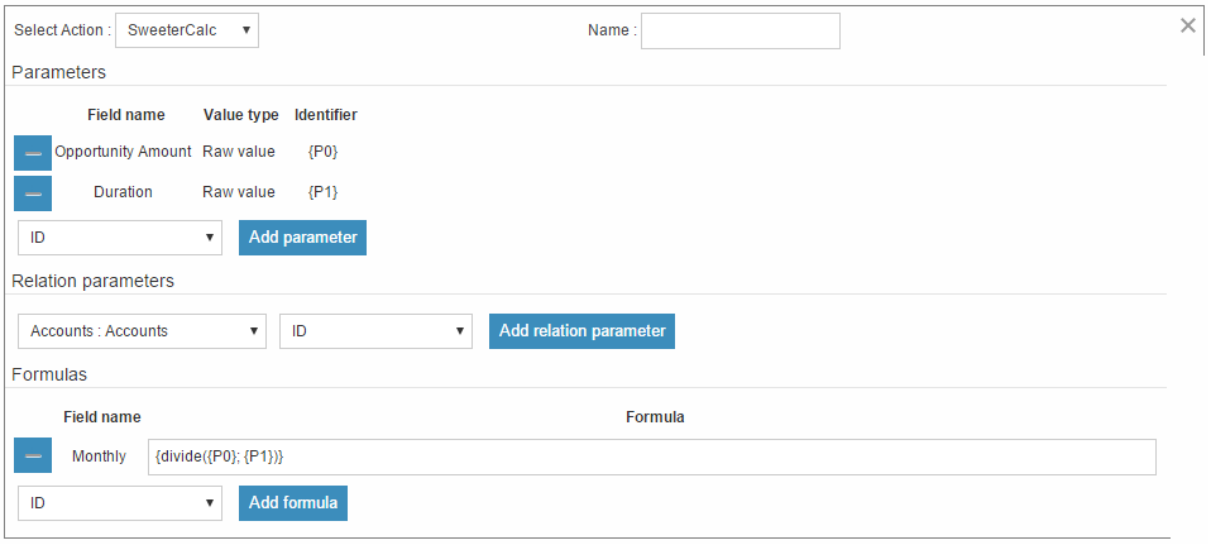
Now, add an action and select SweeterCalc from the dropdown.

Then, add two fields from Opportunities as parameters. First, select Opportunity amount (amount) and add it as a parameter (it will be {P0}) then select Duration and the raw value option from the data type dropdown and add it as parameter two (it will be {P1}). There is no need to add any relational parameters for this formula.

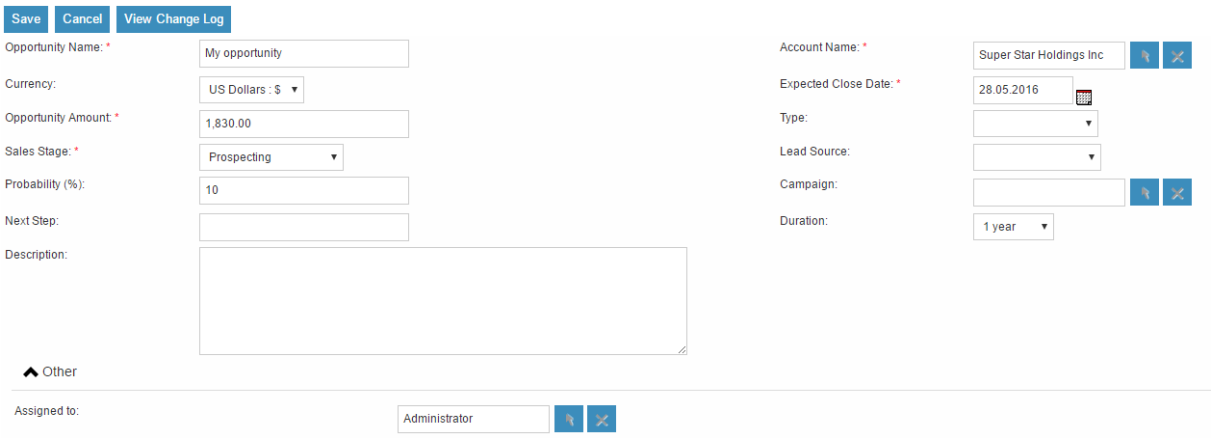
Now, add a formula for the monthly field and fill the textbox with the following formula:

{divide({P0}; {P1})}

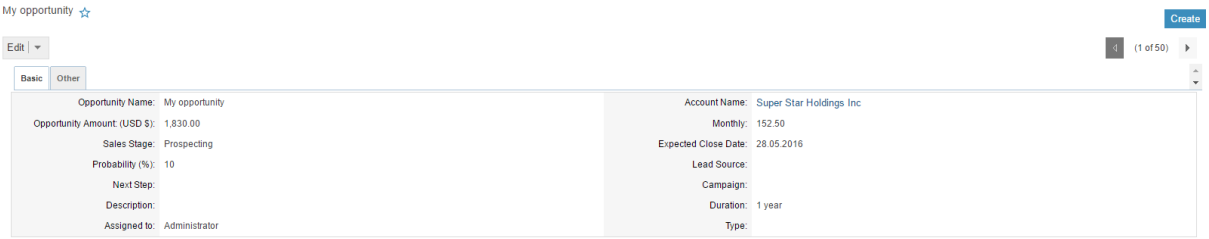
So the whole action should look like this:



Save the WorkFlow and create a new Opportunity:



As you can see, we did not even add the monthly field to the EditView, because we don't want to force the user to make calculations. Save the Opportunity and check the results on the DetailView:



4.2 Set discount based on related contact's type

4.2.1 Use case

Imagine that the company the user works for gives a discount for certain contact types (like students, teachers, etc.). When a contract is saved and a contact is selected, the discount's value needs to be calculated automatically which is determined by the attached contact's type.

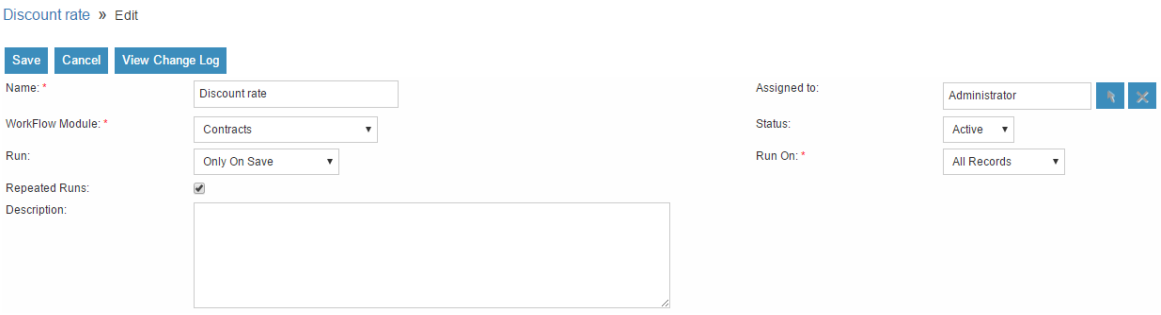
4.2.2 Setup

Our contact module has a dropdown field called Type with values: (database value in brackets) Normal [1], Student [2], Teacher [3]. There is also a Number typed field on contracts called Discount rate.

4.2.3 The WorkFlow

Go to WorkFlow module and create a new WorkFlow. Set the base options like the following:

Name: as you wish	**WorkFlow Module**: Contracts
Status: Active	**Run**: Only on save
Run on: All records	**Repeated runs**: checked



We do not create any conditions, since we would like the WorkFlow to run on all contracts.

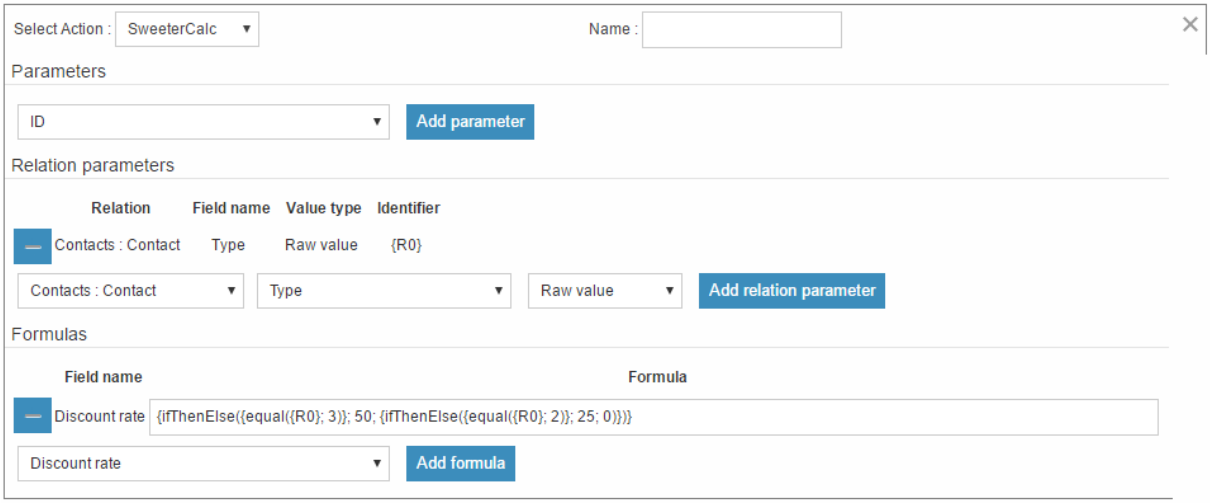
Now, add an action and select SweeterCalc from the dropdown.

Then add a relational parameter: select Contacts : Contact from the relation dropdown, then select Type, then select Raw value and finally push the Add relation parameter button. Our field will have the identifier {R0}. There is no need to add any other parameters for this formula.

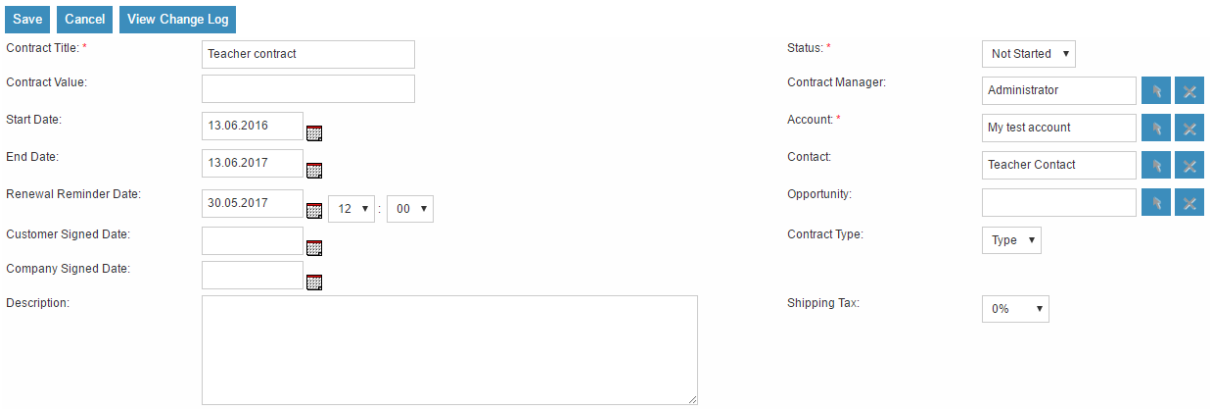
Now, add a formula for the discount rate field and fill the textbox with the following formula:

{ifThenElse({equal({R0}; 3)}; 50; {ifThenElse({equal({R0}; 2)}; 25; 0)})}

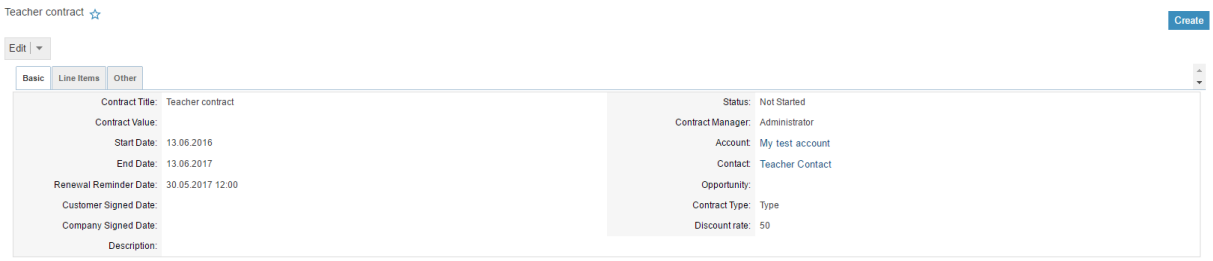
So the whole action should look like this:



Save the WorkFlow and create a new contact with Teacher Type. Now, create a contract and select the contact with Teacher Type in the Contact relate field:



As you can see, we did not even add the discount rate field to the EditView, because we don’t want to force the user to decide this. Save the Contract and check the results on the DetailView:



diligent technology & business consulting GmbH
Schumannstraße 56
60325 Frankfurt am Main
Telefon +49 (0) 69/ 175 3660 - 0
Telefax +49 (0) 69/ 175 3660 - 99
E-Mail: info@dtbc.eu

Sitz der Gesellschaft:
Frankfurt am Main
Amtsgericht Frankfurt am Main
HRB 86980
Umsatzsteuer-ID: DE269831168
Geschäftsführer: Sascha Krause

Bankverbindungen:
Frankfurter Sparkasse Frankfurt/M.
IBAN: DE08500502010200566024
BIC/Swift-Code: HELADEF1822

Commerzbank Frankfurt/M.
IBAN: DE34500400000683517700
BIC/Swift-Code: COBADEFFXXX

5. Troubleshooting

5.1 Common cases

5.1.1 Part of the formula is returned instead of the desired result

In this case the most probable error is that the part of the formula is not well formed. Please check the problematic part of your formula with the techniques described in the **Troubleshooting options** part of the document.

5.1.2 The whole formula is returned instead of the desired result

In this case the most probable error is that the part of the formula is not well formed. Please check the problematic part of your formula with the techniques described in the **Troubleshooting options** part of the document.

5.1.3 Nothing returned instead of the desired result

In this case two things could happened:

- **The calculator may ran into a PHP exception. In this case, you can see on the DetailView of the WorkFlow that the WorkFlow has been failed. Please check the suitecrm.log for more information.**
- **Typical case is when date functions are called with a non-parsable date format.**
- **You tried to set a date or datetime field with a not well formatted data.**
- **Date fields must be filled with dates in the Y-m-d format**
- **Datetime fields must be filled with dates in the Y-m-d H:i:s format**

5.1.4 Formula calculated, but the result is wrong

In this case, two things could have happened:

- **You tried to set a typed field with a wrongly typed data. For example if you try to set a number field with text, the result will be 0, because the database can't insert text into a column with number type. Do the calculation for a text field to see the result of a calculated formula in a non-typed field.**
- **The formula is well formed syntactically, but semantically wrong. For example the substring method is indexing the characters from 0 (which could be strange for non-programmers) and if you except to get the first character and wrote 1 to the start index, you will receive the second character. Please check your function calls with the help of the Usable functions.**

5.2 Troubleshooting options

5.2.1 Checking the syntax of a formula

The following hints can help to identify the problem with a given formula:

- **All functions wrapped with { and } characters, the function's name is followed by (character and after the parameter(s) there is a) character.**
- **The separator for function parameters is the ; character.**
- **The function you use is called with exact amount of parameters. You can check the parameter count for all functions in the Usable functions in this document.**
- **Parameters and relational parameters formatted like {P0}, {P1} … and like {R0}, {R1} … without any white-space characters.**

5.2.2 Enabling debug mode

There is an option to turn on SweeterCalc's verbose mode. To enable this debug functionality you must insert two lines into your config_override.php file:

```
$sugar_config['SweeterCalc']['DebugEnabled'] = 1;
$sugar_config['SweeterCalc']['DebugFileName'] = 'SweeterSyncDebug.log';
```

Of course, you can set the DebugFileName option's filename to any filename you desire.

SweeterCalc will write the process of the calculation in a human readable format into the given file which can help one to identify problems with the calculation.

Since this functionality can generate very big files for complex formulas on each calculation, we advise to turn verbose mode on only if necessary.

5.2.3 Contact us

If all techniques mentioned above fails, you can write an e-mail us to the support@dtbc.eu address and we will get back to you as soon as possible.

It can ease our work if you attach the following to your e-mail:

- **The formula which is not working well.**
- **The used parameters and relational parameters.**
- **The CRM type of the field you try to compute the formula into. (text, date, decimal, relate, etc.)**
- **The expected and the actual result of the formula.**
- **The result file of one verbose mode run of the formula. You can check how to create it in the Enabling debug mode section.**