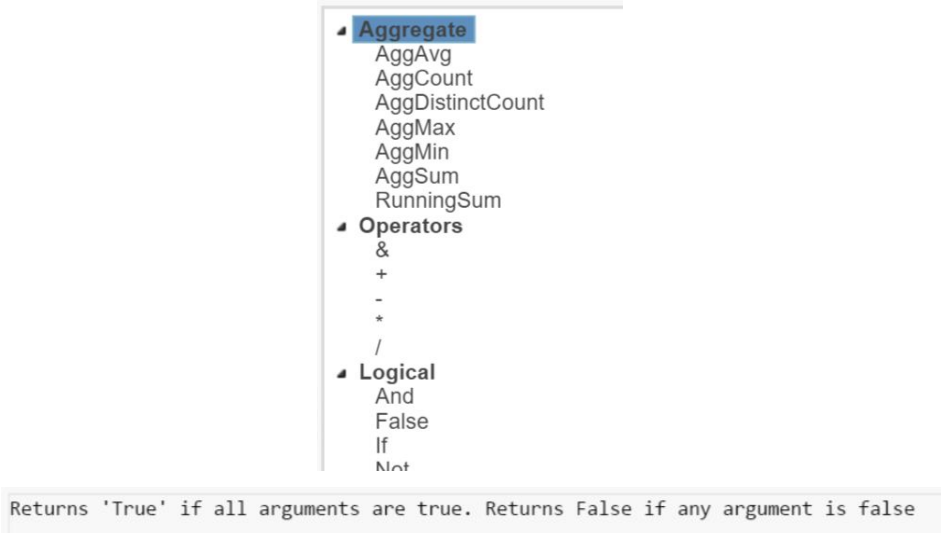
Introduction

When you want to display something more complex than a simple data field in a report cell, you use the formula builder:

Formulas in Report Writer are similar in concept to those in Excel. If you are not familiar with how to use formulas in Excel, there are many introductory guides on the internet, such as this one.

The formula builder provides a list of all available functions. Clicking on one will give you some help text to describe what it does and how to use it.



Aggregate Functions

You would use an aggregate function in a group footer or report footer, to compute some value based on each value in multiple data rows. For example, the detail section of your report might include the Invoice Subtotal, and you may want to get a grand total in a group or report footer.

AggSum

AggSum returns the sum of a column (or of the results of another formula). For example:

Footer: Invoices.Invoice Date	7	**=aggsum({Invoices.Total})**

AggCount

AggCount is similar to AggSum, except it returns the number of items in a column rather than their sum.

AggDistinctCount

AggDistinctCount is similar to AggCount, except it returns the number of unique values in a column. For example, AggCount({Locations.City}) will return the number of locations being summarized, but AggDistinctCount({Locations.City}) will return the number of unique cities in that list of locations. In other words, it computes the count after duplicates are eliminated.

AggMin / AggMax

Similar to AggSum, except these return the minimum or maximum value in a column.

RunningSum

RunningSum is different from the other aggregate functions. You would more typically use this in a detail section. It returns the sum of all values up to this point. For example, if you are creating a customer statement (a list of unpaid invoices), you may want a column that shows, on each row, the total amount due up to that point. RunningSum takes an optional second parameter: the field to total by. Whenever the value of this field changes, the RunningSum starts over.

Example

This report shows all open invoices per Bill-To. In the group footer, we compute a distinct count of cities, a count of invoices, the minimum (oldest) invoice date, and a total balance. In the last column of the detail section, we compute a running sum of the balance.

Statements

Bill-To Code	Name	City	Invoice Number	Invoice Date	SubTotal	Balance	Running Balance
Bill-Tos.Bill-To Code	Bill-Tos.Name	Bill-Tos.City	Invoices.Invoice Number	Invoices.Invoice Date	Invoices.SubTotal	Open Items.Balance	=RunningSum({Open Items.Balance}, {Bill-Tos.Bill-To Code})
		=AggDistinctCount({Bill-Tos.City})	=aggcount({Invoices.Invoice Number})	=aggmin({Invoices.Invoice Date})			=aggsum({Open Items.Balance})

Here is the result:

100111	Tillatoba Diner	Framingham	1189611	12/23/2012	101	101	101
100111	Tillatoba Diner	Framingham	1189785	12/30/2012	147	147	248
		1	2	12/23/2012			248

Operators

Operators are easy! They just combine strings or do math.

&

Use & to concatenate (combine) strings. For example:

```
{Locations.First Name} & " " & {Locations.Last Name}
```

+, -, *, %

Use these to do math on numbers: addition, subtraction, multiplication and division. For example, to compute the effective tax rate of an invoice and show it as a percentage:

```
({Invoices.Tax} / {Invoices.SubTotal}) * 100
```

A $50 invoice with $4 tax has a rate of:

```
(4.00 / 50.00) * 100 = 8
```

Logical Functions

Logical functions let you make decisions inside formulas. The are often combined with each other, which can make them hard to read. Sometimes it's helpful to write them out in Notepad so you can space them out and get the parentheses just right.

If

If lets you test a condition and return different things depending on result of that test. For example, suppose you want to show a Location's name. If it's a commercial property with a company name, you want to show that; otherwise you want to show the first and last name.

```
If({Locations.Company} = "", {Locations.Last Name} & ", " & {Locations.First
Name}, {Locations.Company})
```

Let's break that down.

```
If(<Condition>, <Value if Condition is True>, <Value if Condition is False>)
```

Our condition is, "is the Location Company field empty?" If it is, then the If function will return the second parameter, which is the concatenation of the last name, a comma, and the first name. If the

condition is false -- i.e. Location Company is not empty -- the If function will return the third parameter, which is the Location Company name.

Let's look at a more complex example. Suppose we want the same result as above, but if the Location First Name is empty, we don't want to include the comma after the Last Name.

```
If({Locations.Company} = "", {Locations.Last Name} & If({Locations.First Name} =
"", "", ", ") & {Locations.First Name}, {Locations.Company})
```

To achieve this, we've nested a second If inside the first one: if the first name is empty, don't print anything; if it's not empty, print the comma. You should read the entire formula from the inside out: figure out the result of the inner If first, then plug that into the outer If.

Here's the same formula but spaced out so it's easier to visualize.

```
If(
	{Locations.Company} = "",		← outer condition
	{Locations.Last Name} & If(		← outer value if true
		{Locations.First Name} = "",	← inner condition
		"",				← inner value if true
		", "				← inner value if false
	) & {Locations.First Name},
	{Locations.Company}			← outer value if false
)
```

And, Or

The And and Or functions let you test multiple conditions at once. You'd typically use them as part of the first parameter (the condition) to an If statement.

The And function lets you write formulas like, "if ALL of these things are true, then do this, otherwise do that". The Or function lets you write formulas like, "if ANY of these things are true, then do this, otherwise do that".

For example (and with extra spacing for readability):

```
If(
	And(
		{Locations.Branch} = "East Branch",
		{Locations.City} = "Middletown"
	),
	<Value if Both are True>,
	<Value if Either is False>
)
```

Another:

```
If(
	Or(
		{Service Orders.Service Code} = “RES PC”,
		{Service Orders.Service Code} = “COM PC”
	),
	<Value if Either is True>,
	<Value if Both are False>
)
```

Switch

Switch is a strange but handy way to string lots of Ifs together. The help text describes it like this:

> Takes at least two arguments. The first argument is the evaluation value, the second argument is the default value. Each subsequent pair of arguments represent a condition and a return value. If the condition argument is equal to the evaluation value, the return value argument is returned. Otherwise the next pair of arguments are evaluated. If no condition arguments match, the default value is returned.

Suppose you want to display a certain description depending on the service code of a Service Order. First, in English:

> If the service code is “PC”, print “Pest”. If the service code is “TC”, print “Termite”. If the service code is “MOSQ”, print “Mosquito. Otherwise, print “Other”.

Now with Ifs. Notice that we have to get the number and placement of parentheses just right!

```
If({Service Orders.Service Code} = “PC”, “Pest”, If({Service
Orders.Service Code} = “TC”, “Termite”, If({Service Orders.Service
Code} = “MOSQ”, “Mosquito”, “Other”)))
```

And finally with Switch:

```
Switch({Service Orders.Service Code}, “Other”, “PC”, “Pest”, “TC”,
“Termite”, “MOSQ”, “Mosquito”)
```

Date Functions

Date functions let you perform math on, extract parts of, or format dates and times.

DateAdd

DateAdd lets you add or subtract a certain amount of time from a date. For example, you can subtract 2 hours from a Work Time or add 30 days to an Invoice Date. It takes 3 parameters: the unit to add, the number of units to add, and the date/time to add them to.

```
DateAdd("h", -2, {Service Orders.Work Time})

DateAdd("d", 30, {Invoices.Invoice Date})
```

The possible values for the "unit" parameter are:

```
yyyy   years
q      quarters
ww     weeks
m      months
d      days
h      hours
n      minutes
s      seconds
```

DateDiff

DateDiff subtracts two dates and finds the number of units between them. For example, you can compute how many days there are between a service order's Order Date and Work Date or how many minutes between its Work Time and Posted Work Time. It takes 3 parameters: the unit to compute, the first date, and the second date.

```
DateDiff("d", {Service Orders.Order Date}, {Service Orders.Work Date})

DateDiff("n", {Service Orders.Work Time}, {Service Orders.Posted Work Time})
```

The "unit" values are the same as in DateAdd.

If the second date is earlier than the first, it will return a negative number.

Day, Month, Year

These functions extract a certain part of a date:

```
Day("2/8/2016") = 8

Month("2/8/2016") = 2

Year("2/8/2016") = 2016
```

You can use this to test if the Work Date is in February, for example.

Hour, Minute, Second

These functions extract a certain part of a time:

```
Hour("5:15:27 PM") = 17

Minute("5:15:27 PM") = 15

Second("5:15:27 PM") = 27
```

Note that "Hour" uses 24-hour time, so midnight is zero and "PM" hours are from 12 to 23.

GlobalDateFormat, GlobalDateTimeFormat, TimeFormat1

These functions format a date or time field into a string. You can use cell formatting to do this if the date/time is the only value in the field, but what if you want to concatenate a date with other text?

"Completed at 5:19 pm, Invoiced on 2/8/2016 and Due on 3/9/2016"

```
="Completed at " & TimeFormat1({Invoices.Work Date}) & ", Invoiced on
" & GlobalDateFormat({Invoices.Invoice Date}) & " and Due on " &
GlobalDateFormat(DateAdd("d", 30, {Invoices.Invoice Date}))
```

Now, Today

These functions return the current date ("Today") or date and time ("Now"). These are handy when you're computing how far something is from today, or when you just want to print the date and time the report was executed in the report footer.

String Functions

To come...

Other Functions

To come...