In This Article

Online Engagement

How to: Use Variables in Targeted Email

(8 min read)

You can use **Variables** in Targeted Email content to make it easier to customize your content to fit your circumstances and **to create emails with values that are not available as one of our typical merge fields**. This gives you greater flexibility than using one of our default merge fields or reusables.

Read more about using Reusables with the Classic editor
Read more about using merge fields for conditional content

A simple example might be where you want to create a message that can be used by several of your affiliates but you need the organization name to change wherever it appears in the messages. You can create a custom variable for the organization name and use that as a placeholder in your content whenever the organization name needs to appear. If your affiliate wants to use your email, they will only need to set the value for the variable in one spot. Wherever the variable exists in the message will then display the value they entered.

Understanding variables

There are two kinds of variables available:

- **Single-value variables** can only contain one value such as a number, word, or block of text
- **Array variables** allow you to specify a list of values

You will need to use slightly different commands and formatting to work with each kind of variable, but there are a few aspects they have in common.

Variable names:

- are case sensitive
- should not have spaces
- can only use alphanumeric characters
- cannot start with a number

Variable values can include:

- can have !@#$%^&*()_+-={}|[]\:";<>?,/
- emojis
- spaces

While you cannot use variables in the domain and path portion of a URL, you can use them in the query string portion (the section after the "?").

Example:

https://secure.everyaction.com/6FieqyWE_kepf0B2-2YLgA2?contactdata={{ContactData}}&am=

Working with single-value variables

If you want to **create a variable with only one value**, you can create it using the **set()** command.

The syntax for **set()** is similar to what you use with merge fields and conditional content. Start and end the command using two curly brackets. Inside the parenthesis, add the name and the value for your variable. Each must be wrapped in single or double quotes and separated by a comma.

Example:

{{set('org_name','People For Good')}}

To ensure your variables are included when you use Generate Plain Text, **we suggest that you add all your set() commands directly after the <body> tag in your email**. While variables *can* be set in the **<head>** section of your email, you would need to add them again when you generate your Plain Text version. You can hide your **set()** commands from appearing in the WYSIWYG editor by using a **<noscript>** tag.

Example:

<noscript>

{{set('my_variable_name','my_variable_value')}}

{{set('org_name','People For Good')}}

{{set('ask_multiplier','1.2')}}

</noscript>

Once you've created your variable, **it can be used as many times as needed in your email by using the get()** command.

Example:

Welcome from {{get('org_name')}}!

Working with array variables

An **array contains a list of values**, like, 1,2,3,4 or John, Mike, Mas, Luc, Clifton. You can use the **set()** command to create an array variable but the formatting will differ slightly from what you use for a single-value variable.

Example:

```
{{set('my_array',{'1','2','3','4'})}}
```

Notice that:

- The list starts with a single {
- Each value in the list is wrapped in single quotes
- Each value is separated from the next with a comma

Because there are multiple values associated with an array variable, you cannot use a simple **get()** command to see the value. Instead, **you will need to create a loop** that will allow you to display each value.

Start your loop with **{{each}}** and use **{{end}}** to close the loop. Then, use one of these additional commands to display values based on your array:

- **{{loop_var}}** shows the content of the current value in your list
- **{{loop_index}}** gives you the current value's position in the list
- **{{#get('my_array')}}** calculates the total number of objects in the array

Here is a simple example of an array variable being set and then a loop which displays each value along with its place in the list:

```
{{set('ask_amounts',{'10','20','30','40','50'})}}
{{each get('ask_amounts')}}                    //open the loop for the ask_amounts array
<p>Ask amount #{{loop_index}}: {{loop_var}}</p> //output text, index number, content for present value
{{end}}                                        //close the loop
```

Output:

```
Ask amount #1: 10
Ask amount #2: 20
Ask amount #3: 30
Ask amount #4: 40
Ask amount #5: 50
```

Creating nested loops

You can nest an {{EACH}} statement within another {{EACH}} statement but **if you aren't careful, this can cause the {{loop_var}} to display the value used in the second {{EACH}} call**, rather than what you intend. This is because there is only one {{loop_var}} value at a time and it resets to the new value whenever you execute an {{EACH get()}} statement.

For example, you can see this issue with this nested loop:

```
{{set('array1',{'A','B','C'})}}       //set the array variables
{{set('array2',{'1','2','3'})}}
{{each get('array1')}}                //open the loop for array1
<p>my value: {{loop_var}}             //print the text and current content of loop_var
my list of values:
{{each get('array2')}}                //open loop for array2
{{loop_var}},                         // print current content of loop_var & ','
{{end}}                               //close loop for array2
| my value again:
{{loop_var}}</p>                      //print current content of loop_var
{{end}}                               //close loop for array1
```

You might expect the result of this to be:

```
my value: A my list of values: 1, 2, 3, | my value again: A etc.
```

But instead, you will see:

```
my value: A my list of values: 1, 2, 3 | my value again: 3
```

In this case of nested loops, the first {{loop_var}} prints as expected but after running through our second loop (array2), the final {{loop_var}} printed still has the last value from the second loop.

To get around this issue, you can set() another variable within your loop. The following example will give you the desired result.

Example:

```
{{set('array1',{'A','B','C'})}}
{{set('array2',{'1','2','3'})}}
{{each get('array1')}}
<p>my value: {{loop_var}}                          //print the current content of loop_var
{{set('my_loop_value',loop_var)}}                  //save that value to a new variable we can use later
my list of values: {{each get('array2')}}{{loop_var}},{{end}}
| my value again: {{get('my_loop_value')}}</p>    //print the value we saved above
{{end}}
```

You should now see:

my value: A my list of values: 1, 2, 3 | my value again: A

Conditional display of items in an array

You may have an **array with a list of items that you only want to display if certain conditions are met.**

A simple example might be if you want to create a comma-separated list from a list of values stored in your array. In this example, we'll assume we have a list of animals and we'll create a variable for our list:

```
{{set('animals',{'cat','dog','bird','mouse'})}}
```

We might want to display the items as a comma-separated list in our final content. We know that each item should be followed by a comma and a space unless it is the last value in the list. But if the number of items in our array changes each time we send our email, **we will need to use conditional statements that will display properly for a list of any length.**

Before we can create our conditional loop, we need to know how many values are in our array. We can create a variable to store the number of items in our array using **#get('my_array')**.

```
{{set('animals_length',#get('animals'))}}
```

We can now **create our loop using an {{if equals}} statement to check if we've reached the last value of our array.** If a value is not the last value, we will display the value followed by a comma and a space. If we have reached the last value, we will only return the value.

```
<p>Our list of animals:                          //print this text
{{each get('animals')}}                          //open the loop
{{loop_var}}                                     //print the current item
{{if equals(get('animals_length'),loop_index)}}  //check to see if we are on the last item
{{else}},                                        //if we are not on the last item, add ',' & a space
{{end}}                                          //end the if conditional
{{end}}<p>                                       //end the loop
```

When the email is sent your supporters will see, "Our list of animals: cat, dog, bird, mouse".

If we want to change the list into proper grammar, we can add an *and* and a period at the end.

```
<p>Our list of animals:
{{each get('animals')}}
{{loop_var}}                                                  //print the current item
{{if equals(get('animals_length'),loop_index)}}.              //if this is last item, print '.'
{{elseif equals((get('animals_length') - 1), (loop_index))}}, and   //if this is penultimate item, print ', and'
{{else}},                                                     //if this is any other item, print ', '
{{end}}                                                       //end the if conditional
{{end}}<p>
```

Now when your email is sent, supporters will see, "Our list of animals: cat, dog, bird, and mouse."

Example: Conditional Ask Amounts

Although we provide you with an automated way to create conditional asks in your Targeted Email, you can use variables to create a customized solution to fit your needs.

For example, let's say you'd like to create an email that has:

- five links to the contribution form which will display different ask amounts based on the **Most Recent Contribution** and multiplied by a value
- a query string on the URL that includes the same amount displayed in the linked text that will pre-populate the form with the correct amount
- a query string value on each link with a list of all amounts used in the five links that can be used to customize the buttons for each supporter

The output in your email should look something like this:

Give $10 right now!

Give $15 right now!

Give $20 right now!

Give $25 right now!

Give $30 right now!

You can use conditional content and macros like **round()** and **round_currency()** and so on to calculate amounts, rounded amounts, the max amount, and minimum amount parameters.

Read more about available macros

You can use the following code to add this to your email:

Directly after the <body> tag, set your array of multipliers. For each multiplier in the array, a link will be displayed in your email. Add or remove array values to increase or decrease the number of *ask links*.

```
<body><noscript>

{{set('multipliers',{'1','1.5','2','2.5','3'})}} <!-- For each value in the array, a link will be created where the ask amount is multiplied by an array value. Add or remove values to change the number of links presented to your supporters. -->

{{set('no_hpc_default','20')}} <!-- In our links this value will be used if the supporter does not have a giving history. -->

{{set('count_of_multipliers',#get('multipliers'))}} <!-- To control the ask values for the buttons on a form, we need to provide a comma-separated list of values for amtOpts in each links query string. This value will be used to make sure we do not have a trailing comma at the end of our list. -->

</noscript>
```

Where you want the block of links to show in your email, add the following code (remember to set the link address to your form's URL):

```
<noscript>{{each get('multipliers')}}</noscript>

<p style="text-align: center;"><a href="https://secure.everyaction.com/xPlpnusI3UCGS3-gsRe1gg2?contactdata={{ContactData}}&amp;am={{round_min_max(zero_if_null(loop_var * (HighestPreviousContributionAmount or get('no_hpc_default'))),1,3.00,2500)}}{{set('amount',round_currency(zero_if_null(loop_var * (HighestPreviousContributionAmount or get('no_hpc_default'))),1,3.00,2500))}}&amp;amtOpts={{each get('multipliers')}}{{round_min_max(zero_if_null(loop_var * (HighestPreviousContributionAmount or get('no_hpc_default'))),1,3.00,2500)}}{{if equals(get('count_of_multipliers,loop_index'))}}{{else}},{{end}}{{end}}">Please give {{get('amount')}} right now!</a></p>

<noscript>{{end}}</noscript>
```

NGP | Online Engagement | Targeted Email