FEED – CDO Application Full User Guide Including Eval Syntax (DRAFT)

Step 1: Create an array (or arrays) of CDO records to utilize in the editor.

This sets an array with the name “body” with the CDO records, which are available in the array called “items”:

<eval set='body' value='items'></eval>

You can then add conditions to the array by calling a C# function such as “sort” or a “where” clause:

<eval set='body' value='items._sort(function(x) { (x.company + x.first_name) })'></eval>

<eval set='body' value='items._where(function(x) { !x.city.StartsWith("DSO") })'></eval>

You can combine functions to control the array:

<eval set='body' value='items._where(function(x) { x.city != "Offer" && x.city != "Toolkit" && !x.city.Contains("Virtual") })
._sort(function(x) { _root._todate(x.start_date) })'></eval>

You can also create sub-arrays using the first array you created:

<eval set='body_offers' value='body._where(function(x) { x.city == "Offer" })'></eval>

Step 2: Call the array to begin looping through the records

When calling the array, all code within the array’s <eval> tags will render for each item in the array.

<eval for='body_live'> Content goes here. </eval>

Step 3: Create content within the array to render HTML based on the CDO records

The tokens on the CDO mapping page represent the different fields returned from the CDO.

To display a field simply add a tags with the field's token:

<eval for='location'></eval>, <eval for='city'></eval>

You can also manipulate the format that the data displays in or create conditions based on the value of the field:

<eval for='_root._todate(start_date)' format='MM/dd/yyyy'></eval>

```
<if condition='type2 == "Blogs"'>
        <eval set='src2'
value='"http://images.response.cisco.com/EloquaImages/clients/CiscoSystems/%7b7
1b4782b-40de-40d3-8878-33a98975031d%7d_blog24.png"' ></eval>
    </if>
    <elseif condition='type2 == "Twitter"'>
      <eval set='src2'
value='"http://images.response.cisco.com/EloquaImages/clients/CiscoSystems/%7b4
6e5020a-0dc7-450b-9fb5-7ff0089a1857%7d_twitter24.png"' ></eval>
    </elseif>
```

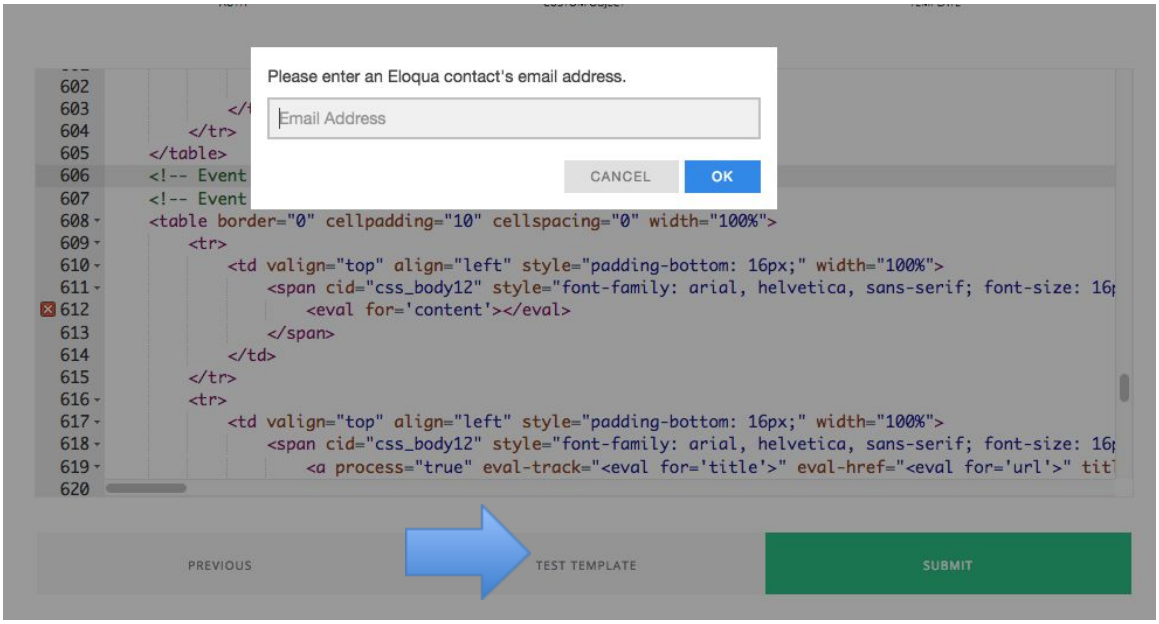
You can also use the field values in common HTML tags by adding “eval-“ to the tag:

<a eval-href="<eval for='url'>" eval-title="<eval for='title'>" />

<img eval-src="<eval for='url'>" />

Step 4: Test your content

By entering an email address of a contact that has CDOs, you can fully test the rendering of your content to that person:



Additional Libraries for C# (.Net) functions:

- String: https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx
- DateTime: https://msdn.microsoft.com/en-us/library/system.datetime(v=vs.110).aspx
- List (Array): https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx

PLEASE NOTE: Not all of these functions have been fully tested in the <eval> syntax. The SureShot team can work with you if there’s a specific function that you are struggling with.

APPENDIX: Additional examples using EVAL syntax

Text Formatting

Note about formatting: Trying to perform specific data formatting on a data type that does not match will result in an error. The "Test" button will allow you to check if your content is rendering correctly before you launch a campaign.

Date Formatting

Eval Examples:

- <eval for='date_created' format='MM/dd/yyyy'></eval>
    - output: 01/15/2017
- <eval for='{cdo_date_field}' format='dddd, MMMM d, yyyy'></eval>
    - output: Monday, April 17, 2017

The following table indicates examples of values that can be included in a date format request. The source date must be in one the following formats:

- 2008-09-15T09:30:41.7752486-07:00
- 2008-09-15T09:30:41.7752486Z
- 2008-09-15T09:30:41.7752486
- Mon, 15 Sep 2008 09:30:41 GMT
- Mon, 15 Sep 2008 09:30:41
- 15 Sep 2008 09:30:41
- 2008-09-15 09:30:41
- 2008-09-15 09:30:41.7752486
- 2008-09-15 09:30:41 AM
- 09-15-2008 09:30:41
- 09-15-2008 09:30:41.7752486
- 09-15-2008 09:30:41 AM

Format specifier	Description	Examples
"d"	The day of the month, from 1 through 31.	2009-06-01T13:45:30 -> 1 2009-06-15T13:45:30 -> 15

"dd"	The day of the month, from 01 through 31.	2009-06-01T13:45:30 -> 01 2009-06-15T13:45:30 -> 15
"ddd"	The abbreviated name of the day of the week.	2009-06-15T13:45:30 -> Mon
"dddd"	The full name of the day of the week.	2009-06-15T13:45:30 -> Monday

"f"	The tenths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 6 2009-06-15T13:45:30.05 -> 0
"ff"	The hundredths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 61 2009-06-15T13:45:30.0050000 -> 00
"fff"	The milliseconds in a date and time value.	6/15/2009 13:45:30.617 -> 617 6/15/2009 13:45:30.0005 -> 000
"ffff"	The ten thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175000 -> 6175 2009-06-15T13:45:30.0000500 -> 0000
"fffff"	The hundred thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175400 -> 61754 6/15/2009 13:45:30.000005 -> 00000
"ffffff"	The millionths of a second in a date and time value.	2009-06-15T13:45:30.6175420 -> 617542 2009-06-15T13:45:30.0000005 -> 000000

"fffffff"	The ten millionths of a second in a date and time value.	2009-06-15T13:45:30.6175425 -> 6175425 2009-06-15T13:45:30.0001150 -> 0001150
"F"	If non-zero, the tenths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 6 2009-06-15T13:45:30.0500000 -> (no output)

"FF"	If non-zero, the hundredths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 61 2009-06-15T13:45:30.0050000 -> (no output)
"FFF"	If non-zero, the milliseconds in a date and time value.	2009-06-15T13:45:30.6170000 -> 617 2009-06-15T13:45:30.0005000 -> (no output)
"FFFF"	If non-zero, the ten thousandths of a second in a date and time value.	2009-06-15T13:45:30.5275000 -> 5275 2009-06-15T13:45:30.0000500 -> (no output)
"FFFFF"	If non-zero, the hundred thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175400 -> 61754 2009-06-15T13:45:30.0000050 -> (no output)
"FFFFFF"	If non-zero, the millionths of a second in a date and time value.	2009-06-15T13:45:30.6175420 -> 617542 2009-06-15T13:45:30.0000005 -> (no output)
"FFFFFFF"	If non-zero, the ten millionths of a second in a date and time value.	2009-06-15T13:45:30.6175425 -> 6175425 2009-06-15T13:45:30.0001150 -> 000115
"g", "gg"	The period or era.	2009-06-15T13:45:30.6170000 -> A.D.

"h"	The hour, using a 12-hour clock from 1 to 12.	2009-06-15T01:45:30 -> 1 2009-06-15T13:45:30 -> 1
"hh"	The hour, using a 12-hour clock from 01 to 12.	2009-06-15T01:45:30 -> 01 2009-06-15T13:45:30 -> 01
"H"	The hour, using a 24-hour clock from 0 to 23.	2009-06-15T01:45:30 -> 1 2009-06-15T13:45:30 -> 13
"HH"	The hour, using a 24-hour clock from 00 to 23.	2009-06-15T01:45:30 -> 01 2009-06-15T13:45:30 -> 13
"m"	The minute, from 0 through 59.	2009-06-15T01:09:30 -> 9 2009-06-15T13:29:30 -> 29
"mm"	The minute, from 00 through 59.	2009-06-15T01:09:30 -> 09 2009-06-15T01:45:30 -> 45

"M"	The month, from 1 through 12.	2009-06-15T13:45:30 -> 6
"MM"	The month, from 01 through 12.	2009-06-15T13:45:30 -> 06
"MMM	The abbreviated name of the month.	2009-06-15T13:45:30 -> Jun
"MMMM"	The full name of the month.	2009-06-15T13:45:30 -> June
"s"	The second, from 0 through 59.	2009-06-15T13:45:09 -> 9

"ss"	The second, from 00 through 59.	2009-06-15T13:45:09 -> 09
"t"	The first character of the AM/PM designator.	2009-06-15T13:45:30 -> P
"tt"	The AM/PM designator.	2009-06-15T13:45:30 -> PM
"y"	The year, from 0 to 99.	0001-01-01T00:00:00 -> 1 0900-01-01T00:00:00 -> 0 1900-01-01T00:00:00 -> 0 2009-06-15T13:45:30 -> 9 2019-06-15T13:45:30 -> 19
"yy"	The year, from 00 to 99.	0001-01-01T00:00:00 -> 01 0900-01-01T00:00:00 -> 00 1900-01-01T00:00:00 -> 00 2019-06-15T13:45:30 -> 19
"yyy"	The year, with a minimum of three digits.	0001-01-01T00:00:00 -> 001 0900-01-01T00:00:00 -> 900 1900-01-01T00:00:00 -> 1900 2009-06-15T13:45:30 -> 2009
"yyyy"	The year as a four-digit number.	0001-01-01T00:00:00 -> 0001 0900-01-01T00:00:00 -> 0900 1900-01-01T00:00:00 -> 1900 2009-06-15T13:45:30 -> 2009

"yyyyy"	The year as a five-digit number.	0001-01-01T00:00:00 -> 00001 2009-06-15T13:45:30 -> 02009
"z"	Hours offset from UTC, with no leading zeros.	2009-06-15T13:45:30-07:00 -> -7
"zz"	Hours offset from UTC, with a leading zero for a single-digit value.	2009-06-15T13:45:30-07:00 -> -07
"zzz"	Hours and minutes offset from UTC.	2009-06-15T13:45:30-07:00 -> -07:00
":"	The time separator.	2009-06-15T13:45:30 -> :
"/"	The date separator.	2009-06-15T13:45:30 -> /
"*string*" '*string*'	Literal string delimiter.	2009-06-15T13:45:30 ("arr:" h:m t) -> arr: 1:45 P 2009-06-15T13:45:30 ('arr:' h:m t) -> arr: 1:45 P
%	Defines the following character as a custom format specifier.	2009-06-15T13:45:30 (%h) -> 1
\	The escape character.	2009-06-15T13:45:30 (h \h) -> 1 h
Any other character	The character is copied to the result string unchanged.	2009-06-15T01:45:30 (arr hh:mm t) -> arr 01:45 A

Numeric Formatting

Eval Examples:

- input: 15.6788
- <eval for='{cdo_number_field}' format='####'></eval>
    - output: 16
- <eval for='{cdo_number_field}' format='#.##'></eval>
    - output: 15.68

The following table indicates examples of values that can be included in a numeric format request.

Format specifier	Name	Description	Examples
"0"	Zero placeholder	Replaces the zero with the corresponding digit if one is present; otherwise, zero appears in the result string.	1234.5678 ("00000") -> 01235 0.45678 ("0.00") -> 0.46
"#"	Digit placeholder	Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string. Note that no digit appears in the result string if the corresponding digit in the input string is a non-significant 0. For example, 0003 ("####") -> 3.	1234.5678 ("#####") -> 1235 0.45678 ("#.##") -> .46
"."	Decimal point	Determines the location of the decimal separator in the result string.	0.45678 ("0.00") -> 0.46

","	Group separator and number scaling	Serves as both a group separator and a number scaling specifier. As a group separator, it inserts a localized group separator character between each group. As a number scaling specifier, it divides a number by 1000 for each comma specified.	Group separator specifier: 2147483647 ("##,#") -> 2,147,483,647 Scaling specifier: 2147483647 ("#,#,,") -> 2,147
"%"	Percentage placeholder	Multiplies a number by 100 and inserts a localized percentage symbol in the result string.	0.3697 ("%#0.00") -> %36.97
"‰"	Per mille placeholder	Multiplies a number by 1000 and inserts a localized per mille symbol in the result string.	0.03697 ("#0.00‰") -> 36.97‰
"E0" "E+0" "E-0" "e0" "e+0" "e-0"	Exponential notation	If followed by at least one 0 (zero), formats the result using exponential notation. The case of "E" or "e" indicates the case of the exponent symbol in the result string. The number of zeros following the "E" or "e" character determines the minimum number of digits in the exponent. A plus sign (+) indicates that a sign character always precedes the exponent. A minus sign (-) indicates that a sign character precedes only negative exponents.	987654 ("#0.0e0") -> 98.8e4 1503.92311 ("0.0##e+00") -> 1.504e+03 1.8901385E-16 ("0.0e+00") -> 1.9e-16
\	Escape character	Causes the next character to be interpreted as a literal rather than as a custom format specifier.	987654 ("\###00\#") -> #987654#

'*string*' "*string*"	Literal string delimiter	Indicates that the enclosed characters should be copied to the result string unchanged.	68 ("# ' degrees'") -> 68 degrees 68 ("#' degrees'") -> 68 degrees
;	Section separator	Defines sections with separate format strings for positive, negative, and zero numbers.	12.345 ("#0.0#;(#0.0#);-\0-") -> 12.35 0 ("#0.0#;(#0.0#);-\0-") -> -0- -12.345 ("#0.0#;(#0.0#);-\0-") -> (12.35) 12.345 ("#0.0#;(#0.0#)") -> 12.35 0 ("#0.0#;(#0.0#)") -> 0.0 -12.345 ("#0.0#;(#0.0#)") -> (12.35)
Other	All other characters	The character is copied to the result string unchanged.	68 ("# °") -> 68 °

String Formatting

In order to perform string formatting, the properties used must not be null or empty.

Concat

Returns a new string that is created by joining two values together

<eval for='{cdofield1} + " " + {cdofield2}'></eval>

<eval set='new_value' value='{cdofield1} + " " + {cdofield2}'></eval>

<eval set='new_value' value='{cdofield1} + "static text"'></eval>

Remove

Returns a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.

<eval for='{cdofield1}.Remove(2)'></eval>

Replace

Returns a new string in which all occurrences of a specified Unicode character in this instance are replaced with another specified Unicode character.

<eval set=’new_value’ value='{cdofield1}.Replace(“Test”, “”)'></eval>

<eval set=’new_value’ value='{cdofield1}.Replace(“Old”, “New”)'></eval>

SubString

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

<eval set=’new_value’ value='{cdofield1}.Substring(5,10)'></eval>

Retrieves a substring from this instance. The substring starts at a specified character position and continues to the end of the string.

<eval set=’new_value’ value='{cdofield1}.Substring(5)'></eval>

ToLower

Returns a copy of this string converted to lowercase.

<eval set=’new_value’ value='{cdofield1}.ToLower()'></eval>

ToUpper

Returns a copy of this string converted to uppercase.

<eval set=’new_value’ value='{cdofield1}.ToUpper()'></eval>

Trim

Returns a new string in which all leading and trailing spaces are removed.

<eval set=’new_value’ value='{cdofield1}.Trim()'></eval>

Comparing Values for conditional statements or creating loops using Queries

Contains

The .Contains(value) function returns a boolean value by evaluating the parent string if it contains the specified value.

```
<eval set='test_contains' value='body._where(function(x) {
                x.{cdofield1}.Contains("Test")
        })'>
</eval>
```

EndsWith

The .EndsWith(value) function returns a boolean value by evaluating the parent string if it ends with the specified value.

```
<eval set='test_EndsWith' value='body._where(function(x) {
                x.{cdofield1}.EndsWith("Test")
        })'>
</eval>
```

StartsWith

The .StartsWith(value) function returns a boolean value by evaluating the parent string if it starts with the specified value.

```
<eval set='test_StartsWith' value='body._where(function(x) {
                x.{cdofield1}.StartsWith("Test")
        })'>
</eval>
```

Equals

The .Equals(value) function returns a boolean value by evaluating the parent string if it starts with the specified value. This can be accomplished with several formats.

```
<eval set='test_Equals' value='body._where(function(x) {
                x.{cdofield1}.Equals("Test")
        })'>
</eval>

<eval set='test_Equals' value='body._where(function(x) {
                x.{cdofield1} == "Test"
        })'>
</eval>

<eval set='test_Equals' value='body._where(function(x) {
                x.{cdofield1} == x.{cdofield2}
```

```
        })'>
</eval>

<eval set='test_Equals' value='body._where(function(x) {
                x.{cdofield1} != x.{cdofield2}
        })'>
</eval>
```

IndexOf

The .IndexOf(value) function returns the zero-based index of the first occurrence of a specified string within this instance. The method returns -1 if the character or string is not found in this instance.

```
<eval set='test_IndexOf' value='body._where(function(x) {
                x.{cdofield1}.IndexOf("T") != -1
        })'>
</eval>

<eval set='test_IndexOf' value='body._where(function(x) {
                x.{cdofield1}.IndexOf("T") > 1
        })'>
</eval>
```

Sort

The _sort statement allows an array to be sorted by a specific property (or combination of properties) of an array item. The sort function produces another array that contains the original array values, but sorted by the properties specified.

In the below example, a new array ‘test_sort’ is created from all ‘items’ values sorted descending by the {cdofield1} property. The resulting array contains all original items, ordered by the specified property.

```
<eval set='test_sort' value='items._sort(function(x) { x.{cdofield1} })' ></eval>

<eval for='test_sort'>
        <eval for={cdofield1}></eval>
        <br/>
        <eval for={cdofield2}></eval>
</eval>
```

To sort an array by ascending values, use the _sortAsc command

```
<eval set='test_sort' value='items._sortAsc(function(x) { x.{cdofield1} })' ></eval>
```

GroupBy

The _groupBy statement allows an array to be grouped by a specific property of an array item. The groupBy function produces another array that contains the original array values, but grouped by the item key specified.

In the below example, a new array ‘test_group’ is created from all ‘items’ values grouped by the {cdofield1} property. The resulting array contains unique keys with grouped values. The eval for ‘Key’ will output the key of cdofield1.

```
<eval set='test_group' value='items._groupBy(function(x) { x.{cdofield1} })'
></eval>

<eval for='test_group'>
        <eval for='Key'></eval>
</eval>
```

Where

The _where statement allows an array to be filtered by a boolean statement evaluated for each array item. The where function produces another array that contains the original array values where the boolean statement evaluated as true.

In the below example, a new array ‘test_where’ is created from all ‘items’ values where the {cdofield1} property is greater than 100. The resulting array contains only items where the field {cdofield1} was greater than 100.

```
<eval set='test_where' value='items._where(function(x) { x.{cdofield1} > 100 })'
></eval>

<eval for='test_where'>
        <eval for={cdofield1}></eval>
        <br/>
        <eval for={cdofield2}></eval>
</eval>
```

Multiple Commands

All comparison functions can be chained together to form complex logic statements. See examples below for different ways the commands can be utilized together.

```
<eval set='test_time' value=’items._where(function(x) {
        x.{cdofield1}.Contains("Test") &&
```

```
        _root._todate(x.{cdofield2}) < _root._nowutc
})'></eval>

<eval set='test_sort' value='body._where(function(x) {
                x.{cdofield1} != "Test1" &&
                x.{cdofield1} != "Test2" &&
                !x.{cdofield1}.Contains("Test3")
        }).sort(function(x) { _root._todate(x.{cdofield2}) })'>
</eval>
```

Other System Variables & Statements to use within Eval syntax

- _root._nowutc
    - Returns the current time in UTC format. Can be used to compare a field in a CDO and only return current or future-dated records
        - Example: *<eval set='body_vdemand' value='body._where(function(x) { _root._todate(x.start_date) < _root._nowutc })'></eval>*
        - This example creates an array of CDO records where the start_date in the CDO (which is first converted to a date format) is less than the current date and time.
- _root._todate
    - Returns a DateTime value converted from the passed value.
        - Example: <eval set='test_date' value ='_root.todate(x.{cdoField1})'></eval>
- _idx
    - Returns the ID of an item in an array of records starting at 0.
        - Example: *<if condition='_idx < 2'>CODE HERE</if>*
        - This example will only display/run the code inside of this tag when it's processing the first or second item in the array.
- IF/ELSE Statements
    - Proper syntax for IF/ELSE statements are:
    - <eval>
      <if condition='condition goes here'>code</if>
      <elseif condition='condition goes here'>code</elseif>
      <else>code</else>
      </eval>