Data Rich Text Widget

Table of Contents

Basic usage

The data rich text can be used to include data values and dynamic content using intuitive rich text formatting that is applied to the dynamic content.

Just like a rich text widget, you can edit the content by typing your text and using the toolbar to format the content (e.g., font, size, color, alignment).

In order to include dynamic content, you need to use a JavaScript-like syntax to reference the data table that is queried by the widget. The syntax provides various functions to access the data, operate on it and write it to the rich text output.

The script needs to be included as text between 2 square brackets.

The following example uses the function ***write*** to write a string to the generated output

Example:

Some rich text content:
[
write('Value at row 0 and column 0 is: ' + **value(0, 0)** + '\n');
write('Value at row 1 and column 0 is: ' + **value(1, 0)** + '\n');
]

Output:

Some rich text content:
Value at row 0 and column 0 is: **2007**
Value at row 1 and column 0 is: **2008**

Types

The rich text scripting syntax supports basic types.

Number

A number can be defined in a variable, used with operators, or passed into any function that accepts a number as a parameter.

Example

```
var myNumber = 124;
var mySum = myNumber + 5;
```

String

A string literal using single quotes can be defined in a variable, used with operators, or passed into any function that accepts a string as a parameter.

Example

```
var aString = 'some string value';
var concatenatedString = aString + ' another string';
```

Array

Similar to Javascript, an array can be defined using a square bracket *[]* and a comma delimited list of content, like an array in JavaScript.
An array can be defined as a variable or passed into any function that accepts an array as a parameter.
The assignment operator can be used with an array to assign values to the array.

Example

```
var anArray = [1,2,3];
anArray.push(4);
anArray[4] = 5;
stringify(anArray);
```

Object

Similar to Javascript, an object can be defined using braces *{}* and a comma delimited list of content, like an object in JavaScript.
An object can be defined as a variable or passed into any function that accepts the object as a parameter.
The assignment operator can be used with an object to assign values to the array.

Example

```
var props = {
   'class': 'myClassName'
};
props.style ='color: red;';
tag('div', props, 'my div content');
```

Operators

Addition (+)
The addition operator (+) produces the sum of numeric operands or string concatenation.

Subtraction (-)
The subtraction operator (-) subtracts the two operands, producing their difference.

Multiplication (*)
The multiplication operator (*) produces the product of the operands.

Division (/)
The division operator (/) produces the quotient of its operands where the left operand is the dividend, and the right operand is the divisor.

Equality (==)
The equality operator (==) checks whether its two operands are equal, returning a Boolean result.

Inequality (!=)
The inequality operator (!=) checks whether its two operands are not equal, returning a Boolean result.

Less than(<)
The less than operator (<) returns **true** if the left operand is less than the right operand, and **false** otherwise.

Less than or equal (<=)
The less than or equal operator (<=) returns **true** if the left operand is less than or equal the right operand, and **false** otherwise.

greater than(>)
The greater than operator (>) returns **true** if the left operand is greater than the right operand, and **false** otherwise.

greater than or equal (>=)
The greater than or equal operator (>=) returns **true** if the left operand is greater than or equal the right operand, and **false** otherwise.

Logical AND (&&)
The logical AND (&&) operator for a set of operands is true if and only if all of its operands are true.

Logical OR (||)
The logical OR (||) operator for a set of operands is true if and only if one or more of its operands is true.

Logical NOT (!)

The logical NOT (!) converts true to false and vice versa.

Increment (++)

The increment operator (++) increments (adds one to) its operand and returns a value.

If used postfix, with operator after operand (for example, `x++`), the increment operator increments and returns the value before incrementing.

If used prefix, with operator before operand (for example, `++x`), the increment operator increments and returns the value after incrementing.

Decrement (--)

The decrement operator (--) decrements (subtracts one from) its operand and returns a value.

If used postfix, with operator after operand (for example, `x--`), the decrement operator decrements and returns the value before decrementing.

If used prefix, with operator before operand (for example, `--x`), the decrement operator decrements and returns the value after decrementing.

Statements

var

The **var** statement declares a globally scoped variable, optionally initializing it to a value. Multiple variables can be declared using a comma separator.

syntax

```
var var1 [,var2,..][= statememt]
```

Example

```
[
var color = 'red';

if (color == 'red') {
  write( 'It is red');
} else {
   write('It is not red');
}
]
```

Result

```
It is red
```

if...else

The **if** statement executes a statement if a specified condition is true. If the condition is false, another statement can be executed.

Multiple **if...else** statements can be nested to create an **else if** clause.

Syntax

```
if ( condition1 )
      statement1
else if ( condition2 )
      statement2
else
      statement3
```

Example

```
[
var color = 'orange';

if (color == 'red') {
   write('It is red');
} else if (color == 'blue'){
   write('It is blue');
} else {
   write('I am not sure');
}

]
```

Result

```
I am not sure
```

for

The **for** statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement (usually a block statement) to be executed in the loop.

Syntax

```
for ([initialization]; [condition]; [final-expression])
      statement
```

Example

```
[
for (var i = 0;i < 3; i++){
   write('Line number ' + i + '\n');
}
]
```

Result

```
Line number 0
Line number 1
Line number 2
```

Functions

Data access functions

value()

Returns a data value for a given row and column indexes. If the value is a measure, the formatted value will be returned as a string. Use the *rawValue* parameter to return a raw number if additional math is required to happen on top of the value.

For backward compatibly with an earlier version of the dynamic rich text widget, this function will add the value to the generated rich text output when the function is used in the body of the script.

Syntax

```
value(rowIndex, columnIndex, rawValue)
```

Parameter values

Parameter	Description
rowIndex	Row index number. Default is **zero**.
columnIndex	Column index number. Default is **zero**.
rawValue	Boolean used to return the raw value of a measure (i.e. non formatted) . Default is **false**

Return value

String or a Number
The default return value is a string that represents the value (category or formatted number). If the value is a measure and the parameter *rawValue* is set to **true**, the return value will be a number.

row()

Returns an array of data values for a given row index. If the value is a measure, the formatted value will be returned as a string. Use the *rawValue* parameter to return a raw number if additional math is required to happen on top of the value.

Syntax

```
row(rowIndex, rawValue)
```

Parameter values

Parameter	Description
rowIndex	Row index number. Default is **zero**.
rawValue	Boolean used to return the raw value of a measure (i.e. not formatted) . Default is **false**

Return value

An array or a string or a number
The default return value is a string that represents the value (category or formatted number). If the value is a measure and the parameter *rawValue* is set to **true**, the return value will be a number.

column()

Returns an array of data values for a given column index. If the value is a measure, the formatted value will be returned as a string. Use the *rawValue* parameter to return a raw number if additional math is required to happen on top of the value.

Syntax

```
column(columnIndex, rawValue)
```

Parameter values

Parameter	Description
columnIndex	Column index number. Default is **zero**.
rawValue	Boolean used to return the raw value of a measure (i.e. non formatted) . Default is **false**

Return value

An array or a string or a number
The default return value is a string that represents the value (category or formatted number). If the value is a measure and the parameter *rawValue* is set to **true**, the return value will be a number.

columnLabel()

Returns the column label for the specified index.

Syntax

```
columnLabel(columnIndex)
```

Parameter values

Parameter	Description
columnIndex	Column index number. Default is **zero**.

Return value

The column label.

columnID()

Returns the column ID for the specified index.

Syntax

```
columnID(columnIndex)
```

Parameter values

Parameter	Description
columnIndex	Column index number. Default is **zero**.

Return value

The column ID.

dataRowCount()

Returns the number of data rows returned in the query

Syntax

```
dataRowCount()
```

Return value

A number that represents the number of data rows.

dataColumnCount()

Returns the number of data columns returned in the query

Syntax

```
dataColumnCount()
```

Return value

A number that represents the number of data columns.

format()

Formats a number using the format setting specified for a given data item.

Syntax

```
format(value, dataItemIndex)
```

Parameter values

Parameter	Description
value	A number to be formatted.
dataItemIndex	The index of the data item to use for the formatting

Return value

String – The formatted value.

Note

You don't have to provide the *dataItemIndex* all the time. If the value parameter is calculated using the *value()* or *column()* functions, the *format()* function will use the right format from the data item column that was used to calculate the value to be formatted..

Example

```
var rawRevenueColumn = column(1, true);
var avgRevenue= avg(rawRevenueColumn);
write(format(avgRevenue));
```

The format function will automatically use the revenue column format in order to format the average value. This is because the average value was calculated using the revenue column.

Result

```
8.78k
```

The value is using the abbreviate format that was set on the revenue column.

Utility functions

getParameter()

Returns the parameter value that was set when using the *actions extension* and when the action is configured as “setParameter”.

Syntax

```
getParameter(name)
```

Parameter values

Parameter	Description
name	A string that represents the parameter name to get.

Return value

String – The parameter value

getUserValue()

Returns the value from the user profile

Syntax

```
getUserValue(name)
```

Parameter values

Parameter	Description
name	A string that represents the parameter name to get. The supported parameters are: productLocale, contentLocale, userName, fullName, email

Return value

String – The user value

join()

Returns a string that joins the items in an array using a provided separator

Syntax

```
join(valuesArray, separator)
```

Parameter values

Parameter	Description
valuesArray	An array of numbers
separator	String - separator

Return value

String – A string that represents all the values in the array separated using the provided separator.

urlEncode()

Returns a URL encoded value

Syntax

```
urlEncode(value)
```

Parameter values

Parameter	Description
value	A string value

Return value

String – A URL encoded value

stringify ()

Converts the object into a string. Useful to convert a JSON object into a string representation. This is equivalent to the JavaScript function *JSON.stringify()*

Syntax

```
stringify(object)
```

Parameter values

Parameter	Description
object	An object to convert into a string

Return value

String – A string that represents a the JSON object.

write ()

Write the provided value to the rich text output. If multiple parameters are provided, they will be separated by a comma.

Syntax

```
write(value)
```

Parameter values

Parameter	Description
value	A string value to be written to the output

html ()

Creates html output that will be rendered inside the rich text widget. The function accepts a string parameter that contains the html or svg to be added. The content of the html is not allowed to contain any JavaScript calls and cannot contain any of the following elements:

'body', 'embed', 'script', 'object', 'applet', 'meta', 'style', 'link'

Attributes that starts with the keyword *on* (e.g. onload)* are not allowed.

Syntax

```
html(value)
```

Parameter values

Parameter	Description
value	A string value that represents the html value to be added.

Return value

Html output string

tag ()

Creates an html tag that will be rendered inside the rich text widget. The content of the html is not allowed to contain any JavaScript calls and cannot contain any of the following elements:

'body', 'embed', 'script', 'object', 'applet', 'meta', 'style', 'link'
Attributes that start with the keyword *on* (e.g., onload)* are not allowed.

Syntax

```
tag(name, properties, content)
```

Parameter values

Parameter	Description
name	String – tag name
properties	Object – contains all the properties to be added to the html tag (e.g., style, id, class, etc.)
content	The content of to be added inside the tag. Could be any value including a text value, number, or a Dom node returned by the *tag* or *html* functions

Return value

Dom node

Math functions

avg()

Returns the average of an array of values

Syntax

```
avg(valuesArray)
```

Parameter values

Parameter	Description
valuesArray	An array of numbers

Return value

Number – The average value

sum()

Returns the sum of an array of values

Syntax

```
sum(valuesArray)
```

Parameter values

Parameter	Description
valuesArray	An array of numbers

Return value

Number – The sum value

Javascript *Math* object

Use the JavaScript *Math* object to access the JavaScript Math functions.

Syntax

```
Math.functionName(args)
```

functions

`abs(x)`	Returns the absolute value of x
`acos(x)`	Returns the arccosine of x, in radians
`acosh(x)`	Returns the hyperbolic arccosine of x
`asin(x)`	Returns the arcsine of x, in radians
`asinh(x)`	Returns the hyperbolic arcsine of x
`atan(x)`	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
`atan2(y, x)`	Returns the arctangent of the quotient of its arguments
`atanh(x)`	Returns the hyperbolic arctangent of x
`cbrt(x)`	Returns the cubic root of x
`ceil(x)`	Returns x, rounded upwards to the nearest integer
`clz32(x)`	Returns the number of leading zeros in a 32-bit binary representation of x
`cos(x)`	Returns the cosine of x (x is in radians)
`cosh(x)`	Returns the hyperbolic cosine of x
`exp(x)`	Returns the value of Ex
`expm1(x)`	Returns the value of Ex minus 1
`floor(x)`	Returns x, rounded downwards to the nearest integer
`fround(x)`	Returns the nearest (32-bit single precision) float representation of a number
`log(x)`	Returns the natural logarithmof x
`log10(x)`	Returns the base-10 logarithm of x
`log1p(x)`	Returns the natural logarithm of 1 + x
`log2(x)`	Returns the base-2 logarithm of x
`max(x, y, z, ..., n)`	Returns the number with the highest value
`min(x, y, z, ..., n)`	Returns the number with the lowest value
`pow(x, y)`	Returns the value of x to the power of y
`random()`	Returns a random number between 0 and 1
`round(x)`	Rounds x to the nearest integer
`sign(x)`	Returns the sign of a number (checks whether it is positive, negative or zero)
`sin(x)`	Returns the sine of x (x is in radians)
`sinh(x)`	Returns the hyperbolic sine of x
`sqrt(x)`	Returns the square root of x
`tan(x)`	Returns the tangent of an angle
`tanh(x)`	Returns the hyperbolic tangent of a number
`trunc(x)`	Returns the integer part of a number (x)

Date functions

Use the JavaScript *Date* object to access the JavaScript static Date functions.

In order to create a date, use the *date()* function.

date ()

Creates a Javascript date object. The function accepts the parameters generally used in the Javascript *Date* constructor.