Text Templates

Last Modified on 06/02/2020 4:59 pm AEDT

Overview

Text templates provide a convenient and safe way to generate **rich-text (HTML)** and **plain-text** within a calculation by using document generation macros.

Consider a calculation that adds strings together to form a rich-text email message, such as:

```
'

' + 'Dear ' + p.[First Name] + ' ' + p.[Last Name] + ',' + '

'  +
'

' + 'You currently have ' + count(p.[Assigned Tasks]) + ' assigned tasks.' + '

'  +
'

' + 'Please review your tasks' + '

'
```

(Assuming that a variable 'p' is present in the Workflow that is assigned to a person.)

This calculation can be rewritten using text templates as follows:

```
html(|

 Dear {p.[First Name]} {p.[Last Name]},



 You current have { count(p.[Assigned Tasks]) } assigned tasks.



 Please review your tasks.


|)
```

Plain Text Templates

Plain text templates can be used to generate text by using the **text(|*content*|)** function. Note that the content within the text function must be contained within two '|' pipe characters.

Free form text, including new lines and (most) special characters, can be used between the pipe characters. Individual calculations and document generation macros can then be placed in the content by using a pair of { } braces.

For example, the result of:

```
text(|The question is 1 + 2 + 3|)
```

will literally be 'The question is 1 + 2 + 3', not 'The question is 6'.

Wheras, the result of:

```
text(|The answer is {1 + 2 + 3} |)
```

will be: 'The answer is 6'.

Important - do not use the text function for generating rich text HTML as this may cause

your calculation to introduce a security vulnerability into your app. Use the html() function instead.

Note that the results of a text template can themselves be used in a calculation, such as the following, which will calculate the length of the generated message.

```
len( text(|The weather is [Weather]|) )
```

Rich Text (HTML formatted) Templates

Rich text HTML templates can be used to generate rich text content by using the **html(|*content*|)** functions.

HTML tags, plain text, and macros can be placed within the HTML template content area, such as:

```
html(|A message formatted with bold text: {[Important Message]}|)
```

The html() and text() functions are similar in use, but the html function has the following important differences:

1. **Macro Encoding** - to ensure that content is correctly and securely displayed
2. **Sanitization** - to protect against potentially malicious generated HTML
3. **HTML Result type** - so that other ReadiNow calculation and platform features know to treat the content as HTML

Refer to the HTML Security Considerations section for more details.

Availability

Plain text and rich text text template are available for use in workflows, calculated fields on forms, and most places that calculations are supported.

They are not, however, available for use in report calculations (either directly or via calculated fields on a report).

Using Document Generation Macros

Most document generation macros can also be used within the **text()** and **html()** functions, except the MERGEFIELD keyword is not required. (This keyword is only required in Word document templates).

Conditionally Including Content

For example, the following plain-text template can be used to include some text only if a certain condition is met:

```
text(|
Welcome [First Name],
{ if count(all([Person]))=100 }
You are the 100th person!
{ end }
This part of the message is always shown.
|)
```

The **if** keyword will conditionally include/exclude a section of the template - up to the corresponding **end** keyword - based on whether the condition is true or false respectively. Take note of the difference between the document generation **if** keyword (one 'f' and no brackets) and the standard **iif(***condition***,** *result-if-true***,** *result-if-false***)** calculation function.

Repeated Content

The following example assumes that a person variable called 'p' has been passed. It uses a HTML template to generate a bullet-point list of tasks:

```
html(|

        { with p.[Assigned Tasks] order by [Name] }
    • {[Name]}
       { end }


|)
```

The

and

HTML tags indicate the start and end of a bullet point list.

The **{with p.[Assigned Tasks] }** macro is a template macro that visits each task record related to the person record. The content between a macro (such as this one) that returns a list of records - and its corresponding **{end}** macro is then repeated - once for each task.

In this case, the repeated content is

- **{[Name]}**

, where

- and

are the HTML tags to start and end each individual bullet point. The repeated content section gets repeated once for every record that was returned by the list calculation - that is once for every task in this example. On each repeat, the current task record become the context record for any further macros within. So the **{[Name]}** macro will show the name of the current task.

A simple list with separators

A common requirement is a comma-separated list of names. Or a list with a HTML line break between each item. These can be achieved respectively by using the {sep} keyword:

```
text(|All tasks:  {with p.[Assigned Tasks}{[Name]}{sep}, {end}|)
```

```
html(|All tasks:  {with p.[Assigned Tasks}{[Name]}{sep}
{end}|)
```

In each case, the content between the **{p.[Assigned Tasks}** list calculation and the **{sep}** keyword is the content that gets repeated, and the content between the **{sep}** keyword and the **{end}** keyword gets used as a separator between each item of content. Note that is a HTML tag that inserts a line-break.

HTML tables

A HTML table can be generated by repeating the row tags of a table. For example:

```
html(|
  {p.[Assigned Tasks]}

  {end}
 Task Name   Due Date

 {[Name]}    {[Due Date]}
|)
```

HTML Security Considerations

HTML Macro Encoding

When the html function is used, the *result* of any calculation macros that are used within the html function will automatically have any special characters encoded to HTML to ensure that they are correctly and securely represented. For example, characters such as apostrophes and angle brackets needs to be converted to HTML encodings such as ' and < in order for email programs and web browsers to treat them correctly.

For example, if a workflow had a variable called '*message*', and the content of message was:

```
We've got a surprise for you!
```

Then the calculation:

```
html(|The message is: {message}|)
```

Will give a correct HTML response of:

```
The message is: We've got a surprise for you! <limited time only>
```

Which, in turn, will correctly be shown in an email program.

HTML Sanitization

The html function provides additional security protection by running a *sanitization check* over the generated rich-text HTML.

The sanitization check ensures that the generated content does not contain any

potentially malicious HTML instructions. For example, if a malicious user were to edit some record data to contain the HTML