RAVENBLACK
TECHNICAL SERVICES

# Ravenblack Products

# Enhanced Sub-tag Suite for Content Intelligence

Version: 1.2.3

Release Date: 2023-08-31

# Table of Contents

# Overview

This is a suite of new "sub-tags" developed by Ravenblack that extend the OpenText WebReports product to provide additional features and functionality that can be used within any WebReport or ActiveView. These sub-tags are designed as "drop-ins" that can be implemented simply by adding two files to a designated folder within Content Server. Although normally a restart is required on each server to activate these drop-in sub-tags, Ravenblack also provides the Ravenblack Sub-tag Loader tool which allows sub-tags to be activated without a restart. (This tool is usually provided in conjunction with this sub-tag suite).

These sub-tags have been broken down into the following categories:

- Development, Support and Debugging
- General Application Development
- Data Saving & Retrieval
- Extensions to Existing Sub-tags

Note that as with all sub-tags, a data tag is required to use these sub-tags. Any kind of data tag can be used (i.e. $constant, &parameter, !variable, and %variable). For some of these sub-tags a literal data tag will be the most appropriate choice:
e.g. `[LL_REPTAG_ '<fileName>' /]`

# Development & Support Sub-tags

## RB_Break

The RB_BREAK sub-tag causes a break in the execution of a WebReport, using Scheduler.debugBreak().

It is only useful when the Eclipse IDE (or builder) is active.

## RB_BuildSubtags

This sub-tag forces a build of all drop-in sub-tags available in the webreports/subtags folder. This build only occurs for whichever thread the sub-tag is executed on.

## RB_CSVersion

This sub-tag returns the current Content Server version and build information. A “FORMAT” option returns the data in a format that is easier to programmatically parse: (x.y.z_yy.q.build), e.g.: 16.2.17_21.3.1604

## RB_MakeTagGuide

This sub-tag forces a rebuild of the tag guide. Normally this tag guide is re-built on a system restart.

## RB_RegisterWithCSApp

This sub-tag can be used to mark any given WebReport as being owned by a particular Content Server Application (CSApp). This ownership is normally created during install but this sub-tag allows developers to create this ownership relationship during development.

## RB_ThreadData

The RB_THREADDATA sub-tag does not require any value in the data tag and returns the thread index by default.

It also supports the following two mutually exclusive parameters:

### TOTALTHREADS

The TOTALTHREADS parameter returns the total number of threads being used on the server.

### BOTH

The BOTH parameter returns the thread index and the TOTALTHREADS value in a JSON format.

Example:

```
{
      "threadIndex":4,
      "totalThreads":8
}
```

# RB_Timer

The RB_TIMER sub-tag provides a way for WebReports developers to set timestamps between multiple points of execution in order to generate reports with delta measurements. Reports can be generated in the output, thread logs or in a unique WebReports log.

## Syntax

The RB_TIMER sub-tag includes a number of syntax variations that will allow the organization set start and end points and identify output location.

### RB_TIMER:SET:<eventString>:[optionalParameter]

This syntax captures a time stamp for different points of time in the execution of a WebReport (or ActiveView).

In addition to any automatic information generated, the SET action accepts additional parameters to create a unique string to output on reporting for each timestamp.
It is possible to specify a string with variable markers in it, followed by one or more data fields to insert into the string. This is equivalent to the OScript STR.Format function, or the printf function in other languages like C or Java.

Example:

```
RB_TIMER:SET:
"Performed %1 action on DataId:%2":[LL_REPTAG_&action /]:[LL_REPTAG=DataId
/]
```

*The %1 marker will be replaced by the value returned by the `&action` tag, and the %2 marker will be replaced by the value returned by the `DataId` (column reference) data tag.*

### RB_TIMER:REPORT:<*optionalParameters*>

The Report option causes a report to be generated showing all set points along with delta measurements between each set point.

Example:

```
Timer data for the "testTimer" timer. Stored time stamps:
*** Pre first data source - TimeStamp: 166312088 Elapsed time: 0 milli-seconds
*** Post first data source - TimeStamp: 166312678 Elapsed time: 590 milli-seconds
*** Post row section data source - TimeStamp: 166316420 Elapsed time: 3742 milli-seconds
*** Post third data source - TimeStamp: 166317017 Elapsed time: 597 milli-seconds
```

This option is always followed by a location parameter which must be one of the following.
ALL: generates a report to all of the locations.
WROutput: generates a report in the output of the WebReport itself.
ThreadLog: generates a report in the standard thread log.
WRLog: uses the `RB_LOG` sub-tag functionality to write the output to a WebReports log file.

Additional information is provided for each location below.

### WROutput

When the `RB_TIMER:REPORT:WROutput` sub-tag syntax is used, it will generate a report that is included in any output from the WebReport.

### ThreadLog

If this location is specified, the report is included in a thread log. Thread logs are normally found in:
`C:\OPENTEXT\<instance_name>\logs\thread_logs`

### WRLog

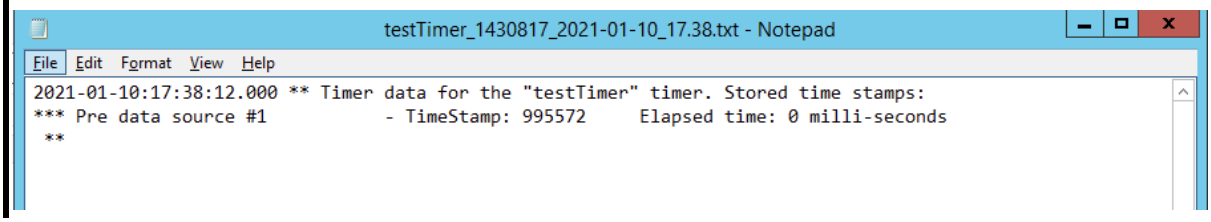
If this location is specified, and the `RB_LOG` sub-tag is installed, a WebReports specific log file will be generated in the following path by default:

```
...\logs\ContentIntelligence_RBlogs
```

The log file will be named according to the timer name.

Example:

```
[LL_REPTAG_'testTimer' RB_TIMER:SET:'Pre data source #1' /]
```




For further information on the naming for log files generated, please refer to the RB_LOG Naming section.

### RB_TIMER:REPORT:<*location*>:RAWVALUES

This sub-tag provides a CSV-type data dump. It defaults to the column marker being a comma (,) and the row marker being End Of Line (usually Line Feed/Carriage Return characters).

An alternative column marker and row marker can be provided.

Example:

```
RB_TIMER:REPORT:WRLog:RAWVALUES:,:|
```

## RB_Trace

The RB_TRACE sub-tag forces a trace log to occur at the point where the sub-tag is executed.

It is most useful when the normal Content Server service is active (i.e. no IDE).

# Application Development Sub-tags

## RB_Assoc

The RB_Assoc expects an Assoc structure in the data tag and performs all of the functions of the existing ASSOC sub-tag but with four additional functions. These functions are:

| Function | Description |
|---|---|
| @keys | Returns an Oscript list with all of the keys in the Assoc specified with the main data tag. This is equivalent to the Assoc.keys() function in Oscript. |
| @isAssoc | Returns true or false depending on whether the data returned in the main data tag is an Assoc or not. |
| @items<br>@values | (Both of these functions do the same thing.) Returns an Oscript list with all of the items (values) in the Assoc specified with the main data tag. This is equivalent to the Assoc.items() function in Oscript. |
| @swapkeyvalues | Returns a new Oscript structure where the values and keys specified in an Assoc in the main data tag, have been swapped. E.g., the new ASSOC can be indexed with a value in order to return the matching key. |

### Syntax

RB_ASSOC:@<*function*>

## RB_ConcatIf

The RB_ConcatIf sub-tag provides a variation on the DECODE or RB_Decode sub-tags. Rather than replacing the original value with a new value when a match is found, this sub-tag pre-pends or appends a new value to the original value. This sub-tag also supports all of the RB_Decode functions.

### Syntax

RB_ConcatIf:BEFORE:<*match*>:<*newvalue*>:<*match*>:<*newvalue*>…

RB_ConcatIf:AFTER:<*match*>:<*newvalue*>:<*match*>:<*newvalue*>…

The syntax for this sub-tag is very similar to RB_Decode but the first parameter is mandatory and is either “BEFORE” or “AFTER” and determines whether a new value (if any) is concatenated before the original data or after it. Note that if there is no match found, then the original value is left as it was.

# RB_CondRowInsert

The RB_CONDROWINSERT sub-tag accepts a data string and determines whether to output that data based on a few defined conditions. This is useful for building lists to determine whether separators should be inserted or not. For example, this can solve the issue when concatenating items in a list and trying to avoid unwanted leading or trailing delimiters. Typically, the data tag is used to return a delimiter such as a comma.

Notes:
This sub-tag only works in the row section when used with an unmodified data set that has not been modified by WebReports tags such as INCLUDEIF.

Most of the options for this sub-tag can only be used in the WebReport's row section; however, the "NOTFIRSTITEM" AND "FIRSTITEM" options allow a delimiter to be inserted based on a list that is not created in the row section.

## Syntax

### RB_CONDROWINSERT:FIRSTROW

This option will output the data only for the first row in the data source.

### RB_CONDROWINSERT:NOT:FIRSTROW

This option will output the data for any row in the data source besides the first row.

### RB_CONDROWINSERT:LASTROW

This option will output the data only for the last row in the data source.

### RB_CONDROWINSERT:NOT:LASTROW

This option will output the data for any row in the data source besides the last row.

### RB_CONDROWINSERT:NOTFIRSTITEM

### RB_CONDROWINSERT:FIRSTITEM

### RB_CONDROWINSERT:FIRSTITEM:<*list id*>

These options are used outside of a row section and are used to determine whether the tag data should be inserted based on whether they are the first item in the list or not. Most commonly the NOTFIRSTITEM option will be used to specify whether a comma should be added before a list item.

The optional "list Id" can be any unique string, and is used to identify a particular list of items, allowing for multiple separate lists.

Examples:

| Syntax (in row section) | Output |
|---|---|
| **[LL_REPTAG=DATAID /][LL_REPTAG_","<br>RB_CONDROWINSERT:notLastRow /]** | **12345,12346,12347** |
| **[LL_REPTAG_","<br>RB_CONDROWINSERT:notFirstRow /][LL_REPTAG=DATAID /]** | **12345,12346,12347** |

| Syntax (not in row section) | Output |
|---|---|
| **[LL_REPTAG_"," RB_CONDROWINSERT:notFirstItem /]{data1}<br>[LL_REPTAG_"," RB_CONDROWINSERT:notFirstItem /]{data2}<br>[LL_REPTAG_"," RB_CONDROWINSERT:notFirstItem /]{data3}** | **{data1},{data2},{data3}** |

# RB_Decode

The RB_DECODE sub-tag works in a similar way to the existing DECODE sub-tag but with some added features that allow matching based on comparison operators (e.g. <=, <, >, >=) as well as type testing such as “isNumber” and “IN” testing for lists and strings.

Like DECODE, the basic algorithm is that the tag data value is tested against 1 or more matches. If one of the match conditions is met, then the original value is replaced with a specified new value. If there is a single parameter at the end of a list of pairs, that is assumed to be a default value. If there is no match and no default value, the original value is returned. If there is more than one matching condition, the first match value pair will be used.

Note: Like the standard DECODE sub-tag any string matching or comparisons are case-sensitive.

# Syntax

## RB_DECODE:<match>:<newvalue>[:additional pairs]:[optional default]

This is the simplest usage (which could also be managed by DECODE).

Example: Simple match with case sensitivity

| **Syntax** | **Return Value** |
|---|---|
| [LL_REPTAG_'snickers1'<br>RB_Decode:snickers1:chocolate:Snickers1:nougat /] | chocolate |
| [LL_REPTAG_'SnickersCandy'<br>RB_Decode:snickers1:chocolate:Snickers1:nougat /] | SnickersCandy |

Example: Default Value

| Syntax | Return Value |
|---|---|
| [LL_REPTAG_'9' RB_DECODE:5:five:6:six:7:seven:8:eight:noMatch /] | noMatch |

## RB_DECODE:<operator>:<newvalue>

Unlike DECODE this sub-tag allows operators that are used to compare the tag data with specified match values. These operators use an 'at' (@) sign.

| Operator | Description |
|---|---|
| @=X or @==X | Equal to the match value |
| @<X | Less than the match value |
| @>X | Greater than the match value |
| @<=X | Less than or equal to the match value |
| @>=X | Greater than or equal to the match value |
| @!X or @!=X | Not equal to the match value |

As noted above, using comparison operators means that there can be more than one possible match in the list of possible matches. Note that processing of the comparison list stops with the first comparison that matches the data value.

Example: Comparison operators with multiple matches

| | Return Value |
|---|---|
| [LL_REPTAG_'10' rb_decode:**@<=10:'1-10'**:**@<=100:'LESS THAN 100':3digits** /] | 1-10 |
| [LL_REPTAG_'10' rb_decode:**@<100:'LESS THAN 100'**:**@<=10:'1-10':3digits** /] | LESS THAN 100 |

## RB_DECODE:<function>:<newvalue>

In addition to operators, functions can also be used to compare data values with specified match values.

| Function | Description |
|---|---|
| @isnumber | Checks if data value is a number |
| @isnotnumber | Checks if data value is not a number |
| @isnull | Checks if data value is null (Accepted NULL values: '', " ", ?, null, NULL) |
| @isnotnull | Checks if data value is not null |
| @isinlist | Checks if data value is in a specified list (case sensitive) |
| @isnotinlist | Checks if data value is NOT in a specified list (case sensitive) |
| @isinstring | Checks if the data value is found in a specified string |
| @isnotinstring | Checks if the data value is NOT found in a specified string |

Example:

| Function | Example | Return Value |
|---|---|---|
| @isnumber | `[LL_REPTAG_'35' rb_decode:@isnumber:True:False /]` | True |
| @isnotnumber | `[LL_REPTAG_'abc' rb_decode:@isNotNumber:True:False /]` | True |
| @isnull | `[LL_REPTAG_"" rb_decode:@isNull:True:False /]` | True |
| @isnotnull | `[LL_REPTAG_'null' rb_decode:@isNotNull:True:False /]` | False |
| @isinlist | `[LL_REPTAG_'skittles' RB_DECODE:@isInList{'skittles','snickers', 'mars bar'}:Candy:Fruit /]` | Candy |
| | `[LL_REPTAG_'SKITTLES' RB_DECODE:@isInList{'skittles','snickers', 'mars bar'}:Candy:Fruit /]` | Fruit (as case sensitive) |
| | `[LL_REPTAG_'skittles' RB_DECODE:@isInList[LL_REPTAG_{'skittles', 'snickers'} /]:Candy:Fruit /]` | Candy |
| @isnotinlist | `[LL_REPTAG_'apple' RB_DECODE:@isNotInList{'skittles', 'snickers','mars bar'}:Fruit:Candy /]` | Fruit |
| @isinstring | `[LL_REPTAG_'SKITTLES' RB_DECODE:@isInString[LL_REPTAG_'skittles snickers mars bar' /]:Candy:Fruit /]` | Candy (case insensitive) |

Note that the last examples of **isinlist** and the last example of **isinstring** each show that a list or string value can be replaced by any data tag, as can the entire parameter.

# RB_ForceType

This sub-tag forces Oscript types in string form, into their native types. This is useful in some scenarios where other sub-tags are not detecting types correctly or for sub-tags that expect a native sub-type rather than a string version of that type. It can also be used to force a native type into a string.

## Syntax

### RB_FORCETYPE

### RB_FORCETYPE:forcestring

# RB_GetText

This sub-tag is an extension of the existing OpenText, GETTEXT sub-tag. It provides 2 additional features above the existing sub-tag: specifically, it allows text to be retrieved for any version of the specified object, and it supports a parameter that specifies an ESCAPE format. This escape parameter only currently supports JSON. (*Note, This JSON format is designed to overcome a bug in the Content Server WEB.EscapeJSON where a carriage return can be lost if it is the final character*).

## Syntax

The RB_GETTEXT sub-tag supports one or two parameters

**RB_GETTEXT:**[version number]

**RB_GETTEXT:**[*escape type*]

**RB_GETTEXT:**[version number]:[escape type]

Escape types: JSON

# RB_GetUploadContent

This sub-tag can be used to extract content from an incoming request where a file has been selected for upload from the browser client (using the HTML Form Input element: Type="File"). Alternatively, the request could be built using Javascript and the FormData object to pass files to this sub-tag.

The data tag provides a string specifying the name of the file input element to use.

## Example:

This HTML sample is built with WebReport tags:

**<FORM NAME=myForm ACTION="[LL_reptag_urlprefix /]" METHOD="post" ENCTYPE="multipart/form-data">**

**[LL_REPTAG_$inputWR LLURL:REPORT URLTOPOST /]**

**Source file: <INPUT NAME=inputFile TYPE="FILE" VALUE="">**

**<INPUT TYPE=submit VALUE="press me">**

**</FORM>**

This WebReport syntax would result in the following HTML snippet:

**<FORM NAME=myForm ACTION="/cs_rbts1/cs.exe" METHOD="post"**

**ENCTYPE="multipart/form-data">**

**<INPUT TYPE="HIDDEN" NAME="func" VALUE="ll">**

**<INPUT TYPE="HIDDEN" NAME="objId" VALUE="1287010">**

**<INPUT TYPE="HIDDEN" NAME="objAction" VALUE="RunReport">**

**Source file: <INPUT NAME=inputFile TYPE="FILE" VALUE="">**

**<INPUT TYPE=submit VALUE="press me">**

**</FORM>**

Based on this snippet, the RB_GETUPLOADCONTENT would be placed in the WebReport referenced by _$inputWR (1287010) and the following sub-tag syntax would take the source from any file the user had selected, and store it in the variable “fileSource”.

**[LL_REPTAG_"inputFile" RB_GETUPLOADCONTENT SETVAR:fileSource /]**

Note, this sub-tag is only designed for text-based content.

# RB_JSONBuild

The RB_JSONBUILD sub-tag creates JSON structures according to the fields specified. Some general notes are:

- Any opening or closing syntax (where needed) is handled by the sub-tag, i.e. there is no need to specify when an object or array has been started or ended.
- No quotes, commas or colons (as related to JSON) are required. The syntax required is only based on standard WebReports sub-tag syntax.
- For incrementally building up multiple, nested objects, the output of this tag should be assigned to a variable (SETVAR sub-tag) and then used as the input for subsequent uses of RB_JSONBUILD.
- Where objects are being nested, it is usually easiest to build the sub-object first and then add it to the larger object. It is also possible to nest syntax.
- Any string values will be JSON escaped (e.g. ESCAPEJSON). It is not currently possible to disable this escaping.

# Syntax

## RB_JSONBUILD OBJECT:<name>:<value>[:optional additional fields]

This syntax is used to build a JSON object by specifying one or more name value pairs.

Example:

```
[LL_REPTAG_'' RB_JSONBuild:object:fruit:apple setVar:fruitList /]
```

Following this syntax, the variable fruitList would contain:

```
{"fruit":"apple"}
```

Example: Adding fields to an existing object

```
[LL_REPTAG_!fruitList
RB_JSONBuild:object:color:"Light Green" setVar:fruitList /]
```

Following this syntax, the variable fruitList would contain:

```
{     "fruit":"apple",
      "color":"Light Green"
}
```

Example: Multiple fields specified

```
[LL_REPTAG_''
RB_JSONBuild:object:fruit:apple:color:"Light Green" setVar:fruitList /]
```

This example is equivalent to the previous 2 examples

## RB_JSONBUILD ARRAY:<*item1*>:<*items1-n*>

This syntax is used to create JSON arrays from multiple sub-tag parameters.

Example:

```
[LL_REPTAG_''
RB_JSONBuild:ARRAY:Orange:Mango:"Water Melon" setVar:fruitList /]
```

Following this syntax, the variable fruitList would contain:

```
["Orange","Mango","Water Melon"]
```

As with the OBJECT option, this syntax allows incremental building.

Example:

```
[LL_REPTAG_''
RB_JSONBuild:ARRAY:Apple:Pear setVar:fruitList /]
```

Following this syntax, the variable fruitList would contain:

```
["Orange","Mango","Water Melon","Apple","Pear"]
```

# RB_Log

The RB_LOG sub-tag provides a way for WebReports developers to produce application specific log files (outside of the normal thread logs). This sub-tag allows a developer to create customized messages to be included in these customized log files, either for debugging and support or to supplement an application.

## Configuration

### Naming

Log files will be named as follows: `<fileName>_<wrid>_<yyyy-mm-dd_hh.mm>`, where:

- `<fileName>` is a unique name specified by the data tag. This field is mandatory as it serves as an identifier to reference a given file. Note, it is possible to have spaces in the file name.
- `<wrid>` is the ID of the WebReport that includes the RB_LOG subtag.
- `<yyyy-mm-dd_hh.mm>` is the date and time when the file was first opened.

### Location

Log files will be generated in the following path by default:

```
...\logs\ContentIntelligence_RBlogs
```

It is also possible to set a different path via configuration or by setting a variable in the application. To set a different default via an INI file, a setting can be added to either the ravenblack.ini, or the opentext.ini files. The setting in ravenblack.ini should be made under [RBSubtagSuite], and if the opentext.ini is used, the setting should be made under the `[WebReports]` section. In either case the setting key is `RB_LOG_FOLDER`.

### Opentext.ini example:

[WebReports]
……
RB_LOG_FOLDER=DebugTest1

### Ravenblack.ini example:

[RBSubtagSuite]
RB_LOG_FOLDER=DebugTest1

In the above examples, the new log folder would be created as `C:\OPENTEXT\<instance_name>\logs\`**`DebugTest1`**`_RBlogs`

Note: After doing this, remember to restart the Content Server service for the configuration to take effect.

### Setting the log path in the application

Sometimes it is desirable to set a logfile path on a per application basis. This is done by setting a variable named RB_LOG_FOLDER to the path name.

Examples:

```
[LL_REPTAG_$MyApplicationNAME SETVAR:RB_LOG_FOLDER /]
[LL_REPTAG_"Security Violations" SETVAR:RB_LOG_FOLDER /]
```

Note: RB_LOG_FOLDER is a fixed variable name that must be used to set a new log path. The data tag provides the actual path that will be used.

## Syntax

The RB_LOG sub-tag includes a number of syntax variations that allows a developer to customize their log files.

### RB_LOG:OPEN

This syntax opens a file for writing.

In cases where there are a lot of transactions, this tag enables the log file to remain open until closing. This is to remove the need to open and close the log file after each WRITE action.

This OPEN action generates a status change message in the log file:

```
YYYY/MM/DD/:HH:MM ** Logfile opened **
```

#### Behaviour

If `OPEN` is initiated on an already open file, it will return a *quiet error*.

Note: A *quiet error* means that an error marker is set internally but no error message is returned. To force a full error message, add the `ONERROR` sub-tag to force a verbose error message to be returned.

### RB_LOG:WRITE:[*outputString*]

This syntax writes lines of information to the log file as specified in the WebReport.

In addition to any automatic information generated, it accepts additional parameters to create a unique string to output to the log file.

It is possible to specify a string with markers in it followed by one or more data fields. This is equivalent to the OScript STR.Format function, or the printf function in other languages like C or Java.

Example:

```
RB_LOG:WRITE:"Performed %1 action on
DataId:%2":[LL_REPTAG_&action /]: [LL_REPTAG=DataId /]"
```

*The %1 marker will be replaced by the value returned by the `&action` tag, and the %2 marker will be replaced by the value returned by the `DataId` (column reference) data tag.*

### Behaviour

Even if a file doesn't call the `OPEN` sub-tag in the code, `WRITE` will automatically open, write, and close the file. If the file is already open, the WRITE action will not open or close the file.

### RB_LOG:CLOSE:[*outputString*]

This syntax closes the log file after writing. This should be used in conjunction with the `OPEN` sub-tag.

This action also accepts (optionally) additional string parameters to `WRITE` a unique string to the log file prior to the close operation.

By default, this action generates this status change message in the log file:

```
YYYY/MM/DD/:HH:MM ** Logfile closed **
```

## WebReports Variable Interactions

Any actions performed by RB_LOG create a WebReports variable using this naming convention: `RB_log|<fileName>.`

`Example:`

Using the syntax `[LL_REPTAG_'FILE1' RB_LOG:OPEN /],`
a WebReports variable is created, equivalent to `SETVAR:RB_LOG|FILE1`

This could be viewed with a variable tag, e.g. `[LL_REPTAG_!RB_LOG|FILE1 /].` Moreover, the content of this variable could be operated on using any appropriate sub-tags.

Additionally, using the `CURRENTVAL` sub-tag following RB_LOG will return the current contents of the variable.

These log file variables contain an OScript ASSOC structure, thus the ASSOC sub-tag may be useful to interpret this variable. The ASSOC currently includes the following components:

```
FileVar (the file variable created), LogFile (the full file
path), FileStatus (open/closed). E.g.:
A<1,?,'fileStatus'='OPEN','fileVar'=U<File(-
107)=xxxxxx>,'LogFile'='C:/……./logs/ContentIntelligence_RBlogs
/logfile1_885992_2020-08-16_23.34.txt'>
```

# RB_RunSearch

The RB_RunSearch sub-tag provides the ability to execute Content Server searches without the need to create a saved search query.

## Syntax

The data tag is generally used to specify an LQL based search term to be run by the search. (LQL is explained in some depth within the Content Server search help.) It is also possible to have a blank data tag and specify search terms using parameters passed to the sub-tag.

### RB_RunSearch

This simple syntax will simply run a search using a set of one or more LQL terms specified in the data tag.

### RB_RunSearch:<search terms>

This simple syntax is equivalent to the previous syntax, but the search terms are specified as an argument instead of via the data tag which is blank in this example.

### RB_RunSearch:LQL_term1:<search terms>:[search options]

This syntax is also equivalent to the previous two examples. In the previous examples, LQL_term1 is assumed by default. Note: only one LQL parameter is currently supported but this term can include multiple search clauses and behaves like the full text search box in the search interface.

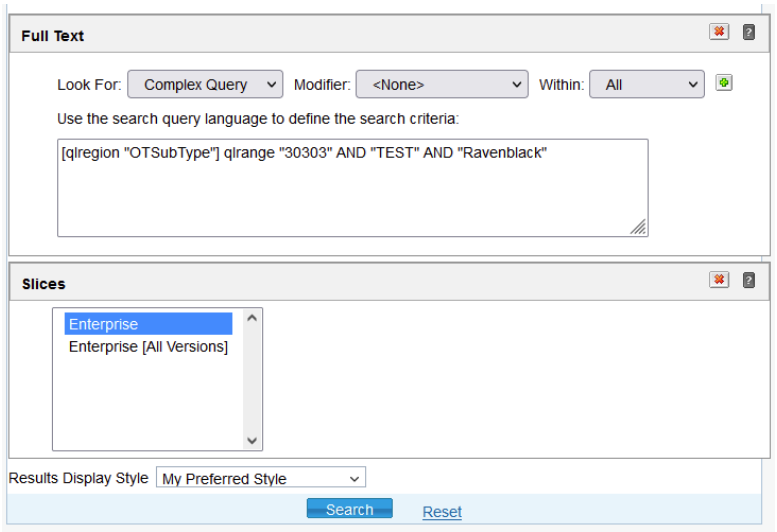
In addition to text or LQL search terms, the following options are supported:

- SLICE: (ENTERPRISE, ENTERPRISEALLVERSIONS)
- STARTROW: (1-n)
- MAXROWS: (1-n)
- OUTPUTMODE:
  - CONTENTS: Returns a RecArray containing all the results of the search.
  - COUNTS: Returns an Assoc with actualRows and TotalRows fields.
  - ALL: Returns an Assoc with the contents RecArray and count fields.

Example:

```
[LL_REPTAG_'[qlregion "OTSubType"] qlrange "[LL_REPTAG_&objectType /]"
AND "[LL_REPTAG_&searchTerm /]" AND "Ravenblack"'
RB_RUNSEARCH:SLICE:ENTERPRISE:MAXROWS:100 /]
```

Assuming that &searchTerm = ‘TEST’ and &objectType=30303, this search would be equivalent to the search query shown below.




# RB_SaveError

The RB_SAVEERROR sub-tag allows an error message to be saved to a variable during the execution of a sub-tag. This can be used where allowing a sub-tag to return an error message could disrupt syntax or break the logic flow. It can also be used to aid in detecting and catching error messages as opposed to normal data being returned. The saved error message can then be used elsewhere in the reportview.

```
{
 "data:"[LL_REPTAG_$Obj NODEINFO:NAME RB_SAVEERROR:errorMsg /] /]",
 "error":[LL_REPTAG_!errorMsg DECODE:'':false:true /],
 "errorMsg:"[LL_REPTAG_!errorMsg /]"}
}
```

In this example, if the NODEINFO:NAME sub-tag returned an error, the data field would resolve to a blank string but the variable “errorMsg” would now contain this error. This error is then used appropriately to set an error field to be returned in the JSON structure.

## Syntax

The RB_SAVEERROR sub-tag only currently supports one parameter with the variable name to use for storing the error.

### **RB_SAVEERROR:***<variable name>*

Note that this sub-tag currently only stores the most recent error message.

# RB_ServerName

The RB_SERVERNAME sub-tag returns the name of the current server. It supports three different formats:
RB_SERVERNAME
RB_SERVERNAME:hostname
Both of these variants return the host name as stored for the server running the Content Server instance - using Oscript System.hostName().

### RB_SERVERNAME:iniserver

Returns the server name as stored in the INI file under [general].

### RB_SERVERNAME:displayname

This variant returns a longer, more descriptive server name as configured and stored in the opentext.ini file under [general] – displayservername.

# RB_StrFormat

The RB_STRFORMAT tag provides a string formatting function that is equivalent to the OScript STR.Format function, or the printf function in other languages like C or Java.

Example:

```
RB_STRFORMAT:
"Performed %1 action on DataId:%2":
[LL_REPTAG_&action /]:[LL_REPTAG=DataId /]
```

The %1 marker will be replaced by the value returned by the `&action` tag, and the %2 marker will be replaced by the value returned by the `DataId` (column reference) data tag.

# RB_SubTypeConvert

This sub-tag overlaps with some standard sub-tags such as NODEINFO and LLURL but provides a multi-function sub-tag that can covert from three different data inputs to multiple outputs, one of which includes miscellaneous sub-type properties that are not available in any other sub-tag.

## Syntax

The input type expected from the data tag is determined by one of three “FROM” parameters as shown below. The output of this sub-tag is determined by one of three “TO” parameters all the current options are shown in this table.

| FROM Parameter | TO Default |
|---|---|
| FROMDATAID,<br>FROMSUBTYPE<br>FROMNAME | TONAME<br>TOSUBTYPE<br>TOASSOC |

The FROM parameter is mandatory, If a TO option is not provided the defaults are as follows:

| FROM Parameter | TO Default |
|---|---|
| FROMDATAID, FROMSUBTYPE | TONAME |
| FROMNAME | TOSUBTYPE |

## RB_SUBTYPECONVERT:<FROM parameter>:[optional TO parameter]

This is the basic syntax. The following examples show the various FROM and TO parameters along with examples. All of the FROM examples use the default TO parameter.

## RB_SUBTYPECONVERT:FROMSUBTYPE

If a valid sub-type number is specified, this option will lookup the sub-type and return information as specified by the TO parameter (TONAME by default in this example).

Example:

```
[LL_REPTAG=Subtype RB_SUBTYPECONVERT:FROMSUBTYPE /]
```

For a sub-type of 144 this would return: Document

## RB_SUBTYPECONVERT:FROMDATAID:

If a valid DataId is specified, this option will look up the correct sub-type and return information as specified by the TO parameter (TONAME by default in this example).

Example:

```
[LL_REPTAG=DataId RB_SUBTYPECONVERT:FROMDATAID /]
```

If the Data Id was for a folder, this would return: Folder

## RB_SUBTYPECONVERT:FROMNAME

If a valid subtype name is specified in the data tag, this will lookup the correct sub-type and return information as specified by the TO parameter (TOSUBTYPE by default in this example).

Examples:

```
[LL_REPTAG_'ActiveView'  RB_SUBTYPECONVERT:fromname /]  30309
[LL_REPTAG_'Document'    RB_SUBTYPECONVERT:fromname /]  144
[LL_REPTAG_'Folder'      RB_SUBTYPECONVERT:fromname /]  0
```

## RB_SUBTYPECONVERT:<*FROM parameter*>:TOSUBTYPE

This TO parameter allows a sub-type number to be returned from either a name or DataId.

## RB_SUBTYPECONVERT:<*FROM parameter*>:TONAME

This TO parameter allows a sub-type value to be converted into the correct sub-type name.

## RB_SUBTYPECONVERT: :<*FROM parameter*>:TOICON

The TOICON parameter uses the sub-type information looked up by the FROM parameter and returns the server path for the icon associated with that sub-type.

## RB_SUBTYPECONVERT:<*FROM parameter*>:TOASSOC

The TOASSOC parameter uses the subtype information looked up by the FROM parameter (FROMSUBTYPE by default in this example) and returns an ASSOC structure with all of the currently supported information fields for this sub-type. These are currently:

| | |
|---|---|
| SUBTYPE | The sub-type number. |
| SUBTYPENAME | The sub-type name. |
| ICONPATH | The server path for the icon associated with a sub-type. |
| ISCONTAINER | Returns true if the sub-type object is a container. |
| HASASSOCIATEDVOLUME | Returns true if the sub-type object is a volume and has an associated workspace. This is true of some volumes like Projects. (See example below for more explanation.) |
| ISVOLUME | Returns true if the sub-type object is a volume. |

Example of volume/associated volume:

| | |
|---|---|
| `[LL_REPTAG_'201'`<br>`RB_SUBTYPECONVERT:FROMSUBTYPE:TOASSOC /]` | Subtype=201<br>subtypeName=Project Workspace<br>iconPath=project/16workspace.gif<br>isContainer=true<br>hasAssociatedVolume=false<br>isVolume=false |
| `[LL_REPTAG_'202'`<br>`RB_SUBTYPECONVERT:FROMSUBTYPE:TOASSOC /]` | Subtype=202<br>subtypeName=Project<br>iconPath=project/16project.gif<br>isContainer=true<br>hasAssociatedVolume=true<br>isVolume=true |

This example shows a project and its associated workspace and the values that are returned by the TOASSOC parameter in each case.

# Data Read/Write Sub-tags

This set of sub-tags provides a variety of levels or data storage and retrieval, ranging from data base tables (including KINI entries), to server preferences, and even thread-based storage.

Warning: some of these sub-tags allow access to standard OpenText tables and files such as the KINI table and the opentext.ini config file. We have made these sub-tags only available to users with System Administrator privileges, but the developer can create a purpose built WebReport set to "run-as" an admin type user. These sub-tags provide useful functionality but require care and attention during development. Partners and or customers can opt out of deploying these sub-tags if you have any concerns. For application specific storage you should consider either:

- RB_FormDBRead/Write (custom WebForms tables).
- CSAppsKiniRead/Write (only permits a section defined by a CSAppName).
- RB_RbPrefsRead/Write (Ravenblack.ini dedicated INI file).

## RB_KiniRead

The RB_KINIREAD sub-tag reads entries from the KINI table.

### Syntax

This sub-tag expects the data tag to specify a valid IniSection name.

RB_KINIREAD:GET:<*IniKeyword*>

This syntax is used to return values from the KINI table section specified by the data tag and using the specified keyword.

## RB_KiniWrite

The RB_KINIWRITE sub-tag adds, edits, and/or deletes entries from the KINI table. Only users with System Administrator privileges can use this sub-tag.

### Syntax

This sub-tag expects the data tag to specify a valid IniSection name.

RB_KINIWRITE:ADD:<IniKeyword>:<IniValue>

The ADD action creates a new item with the identified IniKeyword and IniValue. This essentially does the same as SET but should be used for new entries.

RB_KINIWRITE:SET:<IniKeyword>:<IniValue>

The SET action rewrites the IniValue referenced by the specified IniKeyword.

RB_KINIWRITE:DELETE:<*IniKeyword*>

The DELETE action removes the identified IniKeyword from the section.

# RB_CSAppKiniRead

The RB_CSAPPKINIREAD sub-tag works the same as the RB_KINIREAD sub-tag. It also reads entries from the KINI table but is exclusively used for Content Server Applications (CSApps).

Note that for this sub-tag and the CSAppKiniWrite sub-tag, there is a concept of application "ownership". If a WebReport was bundled with a CSApp that is installed on your system, then it is owned by that CSApp and includes a field within its data to recognize this ownership. For applications being newly built, this data field will not be applied unless you build the app, uninstall it, and then re-install it. Alternatively, you can use the RB_RegisterWithCSApp sub-tag to manually add a WebReport to a specific CSApp.
Note: the Ravenblack Application Analyzer provide a feature to register all ActiveViews or WebReports in a selected CSApp.

## Syntax

This sub-tag expects the data tag to specify a valid IniSection which also matches a valid (installed) CSApp.

RB_CSAPPKINIREAD:GET:<*IniKeyword*>

This syntax is used to retrieve a unique KINI entry, using the specified keyword and the specified IniSection (CSApp name).

# RB_CSAppKiniWrite

The RB_CSAPPKINIWRITE sub-tag works the same as the RB_KINIWRITE sub-tag but works exclusively with Content Server Applications. It also adds, edits, and/or deletes entries from the KINI table but is exclusively used for Content Server Applications (CSApps).

Note: Only users with System Administrator privileges can use this sub-tag. To allow the sub-tag to be used in an application you may use the "Run-as" feature for any WebReport that contains this sub-tag.

## Syntax

The sub-tag requires at least one action is specified from: ADD, SET, DELETE. It expects the data tag to specify a valid IniSection which also matches a valid (installed) CSApp.

RB_CSAPPKINIWRITE:ADD:<*IniKeyword*>:<*IniValue*>

The ADD action creates a new item with the identified IniKeyword and IniValue. It can be used with any existing IniSection that matches a CSApp, even if the containing WebReport is not owned by that app.

For cases where a CSApp IniSection has not yet been created, the ADD action can be only be used to add a brand-new section if the containing WebReport is owned by the CSApp referenced in the IniSection.

### RB_CSAPPKINIWRITE:SET:<*IniKeyword*>:<*IniValue*>

The SET action rewrites the IniValue corresponding to the specified IniKeyword. This action can also be used to ADD a new IniKeyword to an existing IniSection but cannot be used if the section doesn't exist.

### RB_CSAPPKINIWRITE:DELETE:<*IniKeyword*>

The DELETE action removes the identified IniKeyword from the section, regardless of whether the containing WebReport is owned by the corresponding CSApp.

If the DELETE action would result in deleting the IniSection (no more entries) then the containing WebReport must be owned by the CSApp referenced by the IniSection.

# RB_FormDBRead

The RB_FormDBRead sub-tag is used to retrieve database entries from Content Server tables created by the Content Server Forms module that is included with Content Server. Content Server Forms support different storage mechanisms, but this sub-tag is explicitly used to retrieve data that is stored using database tables, e.g., Submission mechanism = SQL Table.

## Security Notes

- The end user running this sub-tag must have SEE/SEECONTENTS permission applied to whichever object has been specified in the data tag (could be a form or a form template).
- The "flexible mode" noted below provides the ability to construct complex logic expressions allowing form data to be referenced based on multiple columns, values, and conditions. As values are often passed as parameters to a WebReport, this feature has intentionally been designed so that most of the expression is defined in the WebReport (rather than being passed as a parameter). Separating the data parameters from the logic makes it easier for this sub-tag to implement security checks to block attempts to inject SQL. Despite this design, the onus is on the WebReports developer to use this feature as designed. Specifically, only values should be passed as parameters and the rest of the expression (column names, operators, AND, OR) should be specified in the WebReport itself. See the @FILTER directive for more information on building these expressions.
- Note, the security check used for passed parameters includes code from the LiveReports "secure mode", but also adds some protection to avoid any attempts to return more results than was intended by the developer.

# Syntax Modes

This sub-tag is designed to work in one of two modes.

- Simple mode – supports the most common development use cases but has limited flexibility. This mode uses normal sub-tag syntax to specify a few useful parameters.
- Flexible (advanced) mode – allows more complex lookup expressions, and several additional options. These options are specified using a special "directive" syntax, similar to some content control tags, e.g., @COLUMNS is used to specify which columns to return. A full list of directives is specified below.

# General Syntax Notes

This sub-tag expects the data tag to specify a valid form or form Template Id. Either the form or the form template can return any or all objects in the corresponding database table, but in simple syntax mode a form Id will only return entries that were made through that form specific form. Using flexible mode, the results can be constrained in a similar way by using the **@MyFormOnly** directive along with a form Id but by default, all matching results will be returned regardless of which form was used to enter them.

## RB_FormDBRead:GET:<*parameters* >

This syntax uses the optional "GET" action. This syntax is provided for consistency with the RB_FormDBWrite sub-tag, but it is optional and can be omitted. For subsequent examples, the GET parameter will not be shown.

# Simple Mode

## RB_FormDBRead:<lookup column>:<lookup key>:[select column]

This syntax is used to return either a single row of data, or a data value from a single row and column. The lookup column and lookup key are used to determine which row is returned based on a simple equals (=) operator, e.g. SEQ = 12.
Without a column being specified, an entire row is returned using an Oscript Record structure. E.g.:

**R<'name'='Bob Smith','age'=49>**

This structure can be referenced using the RECORD sub-tag, e.g.: **RECORD:name**

If a valid column is specified, then only data from that particular column is returned as its native type.

Note that, if the expression returns more than one match, the first row (based on the order they were originally added) is returned.

# Flexible Mode

## RB_FormDBRead:<directives and parameters>

This syntax is used to return simple data values, multi-column results or multiple rows with configurable columns (among other things). There are several options available using a system of directives and parameters. The basic syntax works like this:

## @directive1:param*1*:param*N*:@directive2….

All parameters that follow a directive, belong to that directive, up until the next directive is found, or there are no more parameters. This is equivalent to passing parameters to a function.

As another extension of the common sub-tag syntax, any parameter can be specified in one of these formats:

| Parameter Method | Example | Description |
|---|---|---|
| Simple Value | :<string value>: | A single value is passed to a directive. |
| Comma separated string | :\|Value1,value2,value3\|: | Multiple comma separated values are converted into multiple parameters. In this example, one sub-tag string parameter becomes three. Note that the pipe character \| is used on each side of the string to specify that the text is CSV. |
| Oscript List | :{'value1', 'value2', 'value3'}: | Multiple values specified in an Oscript list, are converted into multiple parameters. In this example, one sub-tag list parameter becomes three. |
| Oscript Assoc | :A<1,?,'Age'=44,'Name'='Bob'>: | Multiple values specified as name/value Assoc fields, are converted into parameters. This format is normally used where a column/value pair are required; however, where a list is required, the field name and field value are broken out into a pair of values. |
| Oscript Record | :R<'Age'=44, Name'='Bob'>: | Multiple values specified as name/value Record fields, are converted into parameters. This format is normally used where a column/value pair are required; however, where a list is required, the field name and field value are broken out into a pair of values. |

| | | |
|---|---|---|
| JSON Array | :[ "value1", "value2", "value3"]: | Multiple values specified in a JSON format array, are converted into multiple parameters. In this example, one sub-tag string/JSON parameter becomes three. Note that this JSON structure doesn't normally require quoting unless an additional double quote is required in one of the array values. |
| JSON Object | : '{"Age":44}': | Multiple values specified in a JSON Object are converted into parameters. This format is normally used where a column/value pair are required; however, where a list is required, the field name and field value are broken out into a pair of values.<br>Note that in this example, the JSON structure had to be wrapped in quotes as the colon would otherwise confuse the syntax. Where a data tag is used, these quotes would not be necessary. |

The array, list, or CSV type parameters allow multiple parameters to be specified in a single sub-tag parameter, allowing lists of parameters to be specified using data tags if necessary.

For example:

… **RB_FORMDBREAD:@COLUMNS:[LL_REPTAG_&parmList /]@MULTIROWS**

The following examples show how some of these methods would look if hard coded in the WebReport but more often these values would be passed through tags.

**...@COLUMNS:|value1,value2,value3|:@MULTIROWS**…

Or

**...@COLUMNS:{'value1', 'value2', 'value3'}:@MULTIROWS**…

Are equivalent to:

**...@COLUMNS:value1:value2:value3:@MULTIROWS**…

# Directives

This table shows the currently supported directives, what they are used for and how many parameters they expect.

| Directive | Params | Description |
|---|---|---|
| @SQLTABLENAME | 0 | Returns the name of the SQL table associated with the form template (even if a form Id has been specified with the data tag). This directive is always used on its own.<br>E.g. **RB_FORMDBREAD:@SQLTABLENAME** |
| @FILTER (**Mandatory** unless SQLTABLENAME is used.) | 2-N | This directive is used to create an expression to specify which rows to return. It supports a simple and complex approach. The simple approach is the same as used in the Simple Mode, i.e.:<br><lookup column>:<lookup key>.<br>To create more flexible expressions, a string format approach is used consisting of:<br>• An Expression string including operators, but using %1, %2, etc. to specify values.<br>• 1 to N number of parameters for insertion.<br>Example:<br>**"Seq > %1 AND Name = ' %2' ":12:Bob: …**<br>Would resolve to:<br>**"Seq > 12 AND Name = 'Bob' "**<br>This approach has been used to allow values to be passed to WebReports without the need to pass full SQL syntax which can create a risk of SQL injection. As the values are separate from the expression, this sub-tag is able to screen the input for any insertion attempts.<br>Note that if you need to specify any percent signs within the expression template, you must use double percent signs.<br>E.g.<br>" Name LIKE '%%**%4**%%' " |
| @COLUMNS | 1-N | Specifies one or more columns to be returned. If not specified, all columns are returned.<br>Examples:<br>**@COLUMNS:Name:Age: …**<br>**@COLUMNS:{ 'Name', 'Age'}: …**<br>**@COLUMNS:\|Name,Age\|: …** |
| @FORMAT | 1 | Specifies which format to use for the returned data. This option currently only supports OSCRIPT and JSON, and defaults to Oscript. See the next chart for further information on how these formats work. |

| Directive | Params | Description |
|---|---|---|
| @MULTIROWS | 0 | Specifies that all rows will be returned rather than only the first one. If this directive is not used, only one row will be returned even if more than one match is found. Note that, in this case, the first row is returned (based on the order they were originally added). |
| @MYFORMONLY | 0 | Specifies that if a form Id was used in the data tag, only entries created with this form will be returned. |
| @SORTCOLUMNS | 1-N | Specifies one or more sort columns to be used. If not specified, data is sorted by DataId,Seq,RowSeqNum,IterationNum.<br>Note that if the @MULTIROWS directive is not used, then only one row is returned and the SORTCOLUMNS option is redundant (and will generate an error).<br><br>Examples:<br>**@SORTCOLUMNS:Name:Age: …**<br><br>**@SORTCOLUMNS:{ 'Name', 'Age'}: …**<br><br>**@SORTCOLUMNS:\|Name,Age\|: …** |
| @SQLDEBUGON | 0 | Allows verbose error messages to be used by developers. By default, a SQL error does not return any information about the SQL syntax. This option should not be used on a production system. |

## Output Formatting

| Results | Mode | Oscript | JSON |
|---|---|---|---|
| Single value from row/column. | Simple | Simple Number/String value | |
| Single row. | Simple | Record structure | Object |
| Single row, **no** @multirows directive. | Flexible | Record structure | Object |
| Single row, **has** @multirows directive. | Flexible | RecArray structure (only 1 row) | Array of Objects |
| Multiple rows | Simple | Record structure (only 1 row) | Object |
| Multiple rows, **has** @multirows directive. | Flexible | RecArray structure (multiple rows) | Array of Objects |

# Examples

This section provides some usage examples. We assume a valid form Id has been provided for all examples.

## Simple Mode: Lookup column/key, no select column

| | |
|---|---|
| Syntax | **RB_FORMDBREAD:Seq:[LL_REPTAG_&SeqNo /] /]** |
| Output | **R<'VolumeID'=-2000,'DataID'=139695,'VersionNum'=0,'Seq'=1,'RowSeqNum'=1, 'IterationNum'=1,'Name'=Bill Smith, 'Age'=49,'Address'='11 Sunshine boulevard'>** |

## Simple Mode: Lookup column/key, select column specified

| | |
|---|---|
| Syntax | **RB_FORMDBREAD:Seq:[LL_REPTAG_&SeqNo /]:Name /]** |
| Output | **Bill Smith** |

## Flexible Mode: Return the name of the SQL Table

| | |
|---|---|
| Syntax | **RB_FORMDBREAD:@SQLTableName** |
| Output | **PersonnelData** |

## Flexible Mode: Return a single row, two columns

| | |
|---|---|
| Syntax | **RB_FORMDBREAD:**<br>**@FILTER:DataId:[LL_REPTAG_$form2 /]:@COLUMNS:Name:Age /]** |
| Output | **R<'Name'='Bill Smith', 'Age'=49>** |

## Flexible Mode: Multiple Rows, complex filter expression

| | |
|---|---|
| Syntax | **RB_**formdbread:<br>**@FILTER:"Age = %1":[LL_REPTAG_&AGE /]:@COLUMNS:Name:Age:**<br>**@MULTIROWS /]** |
| Output | **V{<'Name','Age'><'Bill Smith',49><'Bob Jones',49>}** |

## Flexible Mode: Only return rows entered by the referenced form

| | |
|---|---|
| Syntax | **[LL_REPTAG_$form2 RB_FORMDBREAD:FILTER:"Age > %1":**<br>**[LL_REPTAG_&AGE /]:@COLUMNS:Name:Age /]:@MYFORMONLY /]** |
| Output | **V{<'Name','Age'><'Bill Smith',49><'Bob Jones',49>}** |

### Flexible Mode: Multiple Rows, two columns, Sort by Name, JSON format

| | |
|---|---|
| Syntax | **RB_formdbread:**<br>**@FILTER:"Age = %1":[LL_REPTAG_&age /]:**<br>**@COLUMNS:Name:Age:@MULTIROWS@SORTCOLUMNS:Name:@FORMAT:JSON /]** |
| Output | **[{"Name":"Bill Smith","Age":49},{"Name":"Bob Jones","Age":49}]** |

### Flexible Mode: Two-part filter expression, Columns specified with CSV, Sort columns specified using a list

| | |
|---|---|
| Syntax | **RB_formdbread:@FILTER:"DataId = %1 AND Age > %2":**<br>**[LL_REPTAG_$form /]:[LL_REPTAG_&age /]:@COLUMNS:\|Name,Age\|:**<br>**@MULTIROWS:@SORTCOLUMNS:{'Name','Age'} /] /]** |
| Syntax using list param tags | **RB_formdbread:@FILTER:"DataId = %1 AND Age > %2":**<br>**[LL_REPTAG_&parmList /]:@COLUMNS:[LL_REPTAG_&columns /]:**<br>**@MULTIROWS:@SORTCOLUMNS:[LL_REPTAG_&sortcols /] /]** |
| Output | **V{<'Name','Age'><'Craig Getty',61><'Bill Smith',49>}** |

# RB_FormDBWrite

The RB_FormDBWrite sub-tag is used to add, edit, or delete database entries from Content Server tables created by the Content Server Forms and WebForms modules. Content Server forms support different storage mechanisms, but this sub-tag is explicitly used to manage data that is stored using database tables, e.g., Submission mechanism = SQL Table.

## Security Notes

- The end user running this sub-tag must have “Use "SQL Table" Submission Mechanism” permission applied to the Form object specified by the data tag. The Form Template associated with the Form object should have at least a level of See/SeeContents permissions.
- The “flexible mode” noted below provides the ability to construct complex logic expressions allowing form data to be referenced based on multiple columns, values and conditions. As values are often passed as parameters to a WebReport, this feature has intentionally been designed so that most of the expression is defined in the WebReport (rather than being passed as a parameter). Separating the data parameters from the logic makes it easier for this sub-tag to implement security checks to block attempts to inject SQL. Despite this design, the onus is on the WebReports developer to use this feature as designed. Specifically, only values should be passed as parameters and the rest of the expression (column names, operators, AND, OR) should be specified in the WebReport itself. See the @FILTER directive for more information on building these expressions.
- Note, the security check used for passed parameters includes code from the LiveReports “secure mode”, but also adds some protection to avoid any attempts to return more results than was intended by the developer.

# Syntax Modes

This sub-tag is designed to work in one of two modes. The help for sub-tag RB_FormDBRead includes a more detailed explanation of these modes.

- Simple mode – supports the most common development use cases but has limited flexibility. This mode uses normal sub-tag syntax to specify the data to be edited/deleted and any data to be added or edited.
- Flexible (advanced) mode – allows more complex lookup expressions, and several additional options. These options are specified using a special "directive" syntax, similar to some content control tags, e.g., @DATA is used to specify which data values to insert. All directives supported for this sub-tag are specified in a table under Flexible Mode below.

# General Syntax Notes

Note that this sub-tag is very similar to the RB_FORMDBREAD sub-tag with the exception that this sub-tag has different "actions", different directives, and most actions include data values for addition or editing, specified in pairs of column names and values. For this reason, the documentation for RB_FORMDBWrite focuses on the unique features of this particular sub-tag.

This sub-tag expects the data tag to specify a valid form Id. The syntax and examples that follow, will mostly assume that a valid form Id is provided. Based on the specified form and its associated form Template, all add, edit, or delete actions operate on the SQL table referenced by the form template.

# Syntax

This sub-tag supports the following actions; however, the specific syntax varies according to which of the two modes (simple or flexible) are being used.

RB_FormDBWrite:ADD:*<Data Values>*

RB_FormDBWrite:EDIT:<Lookup Parameters>:<Data Values>

RB_FormDBWrite:EDITALL:<Lookup Parameters>:<Data Values>

RB_FormDBWrite:DELETE:*<Lookup Parameters>*

RB_FormDBWrite:DELETEALL:*<Lookup Parameters>*

## Simple Mode

### RB_FormDBWrite:ADD:

### <Add Column1>:<Add Value1>:[column:value]…

This action and syntax is used to add a new item to the SQL Table. New values will only be added for the specified columns. The remainder will be added as nulls (if nulls are allowed). If an invalid column name is specified, an error is returned.

### RB_FormDBWrite:EDIT:

### <lookup column>:<lookup key>:<Add Column1>:<Add Value1>:[Additional pairs]

Edits a row of data as referenced by the lookup column and lookup key. This could use "SEQ" and a sequence number, or any column/value pair from the data.

The EDIT action will only change the specific fields that have been specified in each column/value pair for addition. Any non-specified fields will remain the same.

If the lookup column/value pair returns more than one result, an error is generated, and the EDIT does not occur. To override this behaviour, and allow the first matching row to be edited, the ONERROR sub-tag can be used to return a blank error (ONERROR:BLANK). This allows the EDIT action to occur. Note that when there are multiple matches, the item with the Seq, RowSeqNum, and IterationNum is the one that is edited. To explicitly allow editing of multiple rows, the EDITALL parameter must be used.

### RB_FormDBWrite:EDITALL:

### <lookup column>:<lookup key>:<Add Column1>:<Add Value1>:[Additional pairs]

Works like EDIT but it allows multiple rows of data (that have the same lookup value) to be changed.

### RB_FormDBWrite:DELETE:

### <lookup column>:<lookup key>

Deletes a row of data as referenced by the lookup column and lookup key. This could use "SEQ" and a sequence number, or any column/value pair from the data. If the column/value pair returns more than one result, an error is generated, and the DELETE does not occur. To explicitly allow deleting of multiple rows, the DELETEALL parameter should be used.

### RB_FormDBWrite:DELETEALL:

### <lookup column>:<lookup key>

Works like EDIT but it allows multiple rows of data (that have the same lookup value) to be changed.

# Flexible Mode

## RB_FormDBWrite:<directives and parameters>

This syntax is used to add, edit and delete records as with the simple mode; however, more complex lookup expressions are used, several different modes of specifying data are available, and some useful options are also provided.

This mode uses a syntax construct called “directives” that is explained in detail under the Flexible Mode section for RB_FORMDBREAD. A table explaining the various advanced methods for specifying parameters can also be found there. The following table illustrates this advanced parameter approach, showing how it would be used to specify data for adding or editing (following the @DATA directive).

These input type examples are all used to specify one or more name/value pairs to be used for adding or editing data values.

| Input Type | Syntax example |
|---|---|
| Sub-tag Method | … :Name:'Bill Smith':Age:49:Address:'11 Sunshine boulevard' |
| Oscript List | {'Name','Bill Smith', 'Age',49,'Address','11 Sunshine boulevard'} |
| JSON Array | ["Name","Bill Smith", "Age",49,"Address","11 Sunshine boulevard"] |
| OScript Record | R<'Name'='Bill Smith','Age'=49,'Address'='11 Sunshine boulevard'> |
| Oscript Assoc | A<1,?,'Name'='Bill Smith','Age'=49,'Address'='11 Sunshine boulevard'> |
| JSON Object | {"Name"="Bill Smith", "Age"=49,"Address"="11 Sunshine boulevard"} |
| CSV Method | \|Name,Bill Smith,Age,49,Address,11 Sunshine boulevard\| |

The Array, Object, Assoc, Record or CSV type parameters allow multiple parameters to be specified in a single sub-tag parameter, allowing groups of parameters to be specified in single data tags if necessary.

# Directives

The following table shows the currently supported directives, what they are used for and how many parameters they expect.

| Directive | Params | Description |
|---|---|---|
| @FILTER (Mandatory for EDIT, EDITALL, DELETE, DELETEALL) | 2-N | This directive is used to create an expression to specify which rows to return. It supports a simple and complex approach. The simple approach is the same as used in the Simple Mode, i.e.:<br><lookup column>:<lookup key>.<br>To create more flexible expressions, a string format approach is used consisting of:<br>• An Expression string including operators, but using %1, %2, etc. to specify values.<br>• 1 to N number of parameters for insertion.<br><br>Example:<br>**"Seq > %1 AND Name = ' %2' ":12:Bob: …**<br><br>Would resolve to:<br>**"Seq > 12 AND Name = 'Bob' "**<br><br>This approach has been used to allow values to be passed to WebReports without the need to pass full SQL syntax which can create a risk of SQL injection. As the values are separate from the expression, this sub-tag is able to screen the input for any insertion attempts.<br>Note that if you need to specify any percent signs within the expression template, you must use double percent signs.<br>E.g.<br>" Name LIKE '%%**%4**%%' " |
| @DATA (Mandatory for ALL, EDIT, EDITALL) | 2-N | Specifies name/value pairs for the column name and the value to insert for add or edit actions.<br>For adding or editing actions, only the specified columns are changed, any unspecified columns are left alone or as NULL (if NULLs are allowed). |
| @FORMAT | 1 | Specifies which format to use for any return from the operation. This option currently only supports OSCRIPT and JSON, and defaults to Oscript. See the Output chart below for further information on how these formats work for different responses. |
| @RESPONSE | 1 | Specifies what to return from the operation. Defaults to blank string. Supported response types:<br>**BLANK** – Empty string<br>**REFNUMS** – Either one sequence number or a list.<br>**STATUS** – A status record with fields to .<br>**See the response type chart below for further information on responses.** |

| | | |
|---|---|---|
| @SQLDEBUGON | 0 | Allows verbose error messages to be used by developers. By default, a SQL error does not return any information about the SQL syntax. This option should not be used on a production system. |

## Response Types

| @RESPONSE:<type> | Normal | Error |
|---|---|---|
| BLANK | <blank string> | Normal error string |
| REFNUMS | List of 1 or more sequence numbers (format determines whether Oscript list or JSON array) | Normal error string |
| STATUS | Returns an Oscript Assoc or a JSON object with status fields: | |
| | - error: false<br>- errorMsg:""<br>- refnums: (List/array) of Sequence numbers. For ADD action, the number of the newly added item is returned; for edit and delete actions a list of the sequence numbers acted upon is returned. | - Error:true<br>- errorMsg:"<error string"<br>- refnums:[] |

## Output Formatting

| Results | Oscript | JSON |
|---|---|---|
| New record number (SEQ)<br>Edited record number<br>Deleted record number | String Value | |
| Multiple record numbers | List | Array |
| Status Record:<br>error=true/false<br>errorMsg="text"<br>refnums={1,2,3} | ASSOC | Object |

## Examples

This section provides some usage examples. We assume a valid form Id has been provided for all examples.

### Simple Mode: Add item

| | |
|---|---|
| Syntax | RB_FORMDBWRITE:ADD:Name:"Bill Smith":Age:49 |
| Output | "" |

## Simple Mode: Edit Item, single column change.

| | |
|---|---|
| Syntax | **RB_FORMDBWRITE:Seq:[LL_REPTAG_&SeqNo /]:Age:50 /]** |
| Output | "" |

## Flexible Mode: Add a record

| | |
|---|---|
| Syntax | **RB_FORMDBWRITE:ADD:**<br>**@DATA:{'Name',[LL_REPTAG_&Name /],'Age',[LL_REPTAG_&Age /]}:**<br>**@RESPONSE:REFNUMS** |
| Output | {7} |

## Flexible Mode: Edit multiple items

| | |
|---|---|
| Syntax | **RB_FORMDBWRITE:EDITALL:**<br>**@FILTER:"Date > %1":[LL_REPTAG=ModifyDate /]:**<br>**@DATA:A<1,?,'Name'='Bill Smith','Age'=49>:**<br>**@RESPONSE:STATUS@FORMAT:JSON** |
| Output | **{"error":false, "errorMessage":"", "refNums":[3,5,9]}** |

## Flexible Mode: Delete multiple items. Fixed expression

| | |
|---|---|
| Syntax | **RB_FORMDBWRITE:DELETEALL:**<br>**@FILTER:"delete = true":**<br>**@RESPONSE:STATUS@FORMAT:JSON** |
| Output | **{"error":false, "errorMessage":"", "refNums":[1,2,3]}** |

## Flexible Mode: Edit an item

| | |
|---|---|
| Syntax | **RB_FORMDBWRITE:EDIT:**<br>**@FILTER:["ID <> %1 AND Name LIKE '%%%2%%' ",-1,"[LL_REPTAG_&srch /]"]:**<br>**@DATA:**<br>**@RESPONSE:STATUS:@FORMAT:OSCRIPT** |
| Output | **A<1,?,'error'=false,'errorMessage'='','refnums'={5}>** |

# RB_IniPrefsRead

The RB_INIPREFSREAD sub-tag read entries from a specific section in the opentext.ini file. It works in a similar way to RB_KINIREAD and RB_CSAPPSKINIREAD.

## Syntax

This sub-tag expects the data tag to specify a valid section in the opentext.ini file.

### RB_INIPREFSREAD:GET:<*fieldname*>

This syntax is used to return the value of a specified field name.

# RB_IniPrefsWrite

The RB_INIPREFSWRITE adds, edits, and/or deletes fields from a specified opentext.ini section. It works in a similar way to
RB_KiniWrite and RB_CSAPPSKINIWRITE.

If the section does not exist in the opentext.ini file, it will create a new one.

Only users with System Administrator privileges can use this sub-tag.

## Syntax

The sub-tag accepts different mandatory actions. It expects the data tag to specify either a valid existing section name in the opentext.ini file or a valid text name to use as a new section name (ADD).

### RB_INIPREFSWRITE:ADD:<*fieldname*>:<*value*>

The ADD action creates a field using the specified field name and populates it with the corresponding value. If the specified section name (in the data tag) does not exist, it will be created.

### RB_INIPREFSWRITE:SET:<*fieldname*>:<*value*>

The SET action modifies the value corresponding to the specified field name.

### RB_INIPREFSWRITE:DELETE:<*IniKeyword*>

The DELETE action removes the specified keyword/value from the section referenced in the data tag.

### RB_INIPREFSWRITE:DELETESECTION:<*IniKeyword*>

The DELETESECTION action removes all of the keyword/values in a specified section, effectively removing that section from the opentext.ini file. Note that the section name will remain and cannot be automatically removed; however, it will not affect any functionality.

# RB_RbPrefsRead

The RB_RBPREFSREAD sub-tag read entries from a specific section in the ravenblack.ini file. It works similarly to RB_INIPREFSREAD but works with the Ravenblack specific INI file.

## Syntax

This sub-tag expects the data tag to specify a valid section in the ravenblack.ini file.

## RB_RBPREFSREAD:GET:<*fieldname*>

The GET action is used to return the value for a specified field name.

# RB_RbPrefsWrite

The RB_INIPREFSWRITE sub-tag adds, edits, and/or deletes fields from a specified ravenblack.ini section. It works in a similar way to RB_INIPREFSWRITE.
Only users with System Administrator privileges can use this sub-tag.

## Syntax

The sub-tag accepts different actions which are mandatory to use the sub-tag. It expects the data tag to specify a valid section in the opentext.ini file.

## RB_RBPREFSWRITE:ADD:<*fieldname*>:<*value*>

The ADD action creates a field using the specified field name and populates it with the corresponding value.
If the section does not exist in the ravenblack.ini file, it will be created.

## RB_RBPREFSWRITE:SET:<*fieldname*>:<*value*>

The SET action modifies the value corresponding to the specified field name.

## RB_RBPREFSWRITE:DELETE:<*IniKeyword*>

The DELETE action removes the specified field from the section referenced in the data tag.

## RB_RBPREFSWRITE:DELETESECTION:<*IniKeyword*>

The DELETESECTION action removes all of the keyword/values in a specified section, effectively removing that section from the ravenblack.ini file. Note that the section name will remain and cannot be automatically removed; however, it will not affect any functionality.

# RB_ThreadVarRead

RB_ThreadVarRead and RB_ThreadVarWrite are used to manage variable values on a per-thread basis. These variables differ from standard WebReports variables in that WebReports variables only exist for the life of the WebReport execution and thread variables exist for the time that the server is running.

The RB_THREADVARREAD sub-tag reads the value stored in a specified variable.

## Syntax

This sub-tag expects the data tag to specify the name of a variable to retrieve.

## RB_THREADVARREAD:GET

This syntax is used to return the value of a variable specified by the data tag.

# RB_ThreadVarWrite

The RB_ThreadVarWrite sub-tag is used to set or delete variables stored for the current thread.

## Syntax

The data tag is used to specify the variable name to be set or deleted.

## RB_THREADVARWRITE:SET:<*value*>

The SET action either adds a variable (if it doesn't already exist) or modifies it.

## RB_THREADVARWRITE:DELETE

The DELETE action removes the specified variable (if it exists) from the set of variables stored for the current thread.

# Beta Sub-tags

These sub-tags are already in limited usage by Ravenblack, but have not been finalized and or fully tested. They are available by request.

## RB_CACHEREAD

Allows a cached item (using the built in Cacheutil functionality) to be returned.

## RB_CACHEWRITE

Allows new cache items to be added or updated.

## RB_CSAPPINFO

Returns information for any given CSApp on the system. If used from a WebReport that is "owned by" a CSApp, this sub-tag defaults to the owning CSApp.

## RB_FILEUTILS

Used to list files and or return file paths.

## RB_GETTEXTFILE

Returns text source from a file on the file system.

## RB_SETEXTDATA

Can be used to set the Extended Data for an object. (this sub-tag name may change)

## RB_USERURL

Equivalent to LLURL but returns URLs to use for various user actions.

## RB_WEBREPORTBUILDER

Can be used to create a WebReport and or set the various pieces of meta data such as constants and parameters.

# About Ravenblack

Ravenblack Technical Services enables users of OpenText Content Intelligence, Perspectives, and Smart View to get more out of their investments in OpenText Content Suite and Extended ECM (xECM) platforms. Owned by Greg Petti, one of the original founders of Resonate Knowledge Technologies (RKT), Ravenblack provides products, consulting, best practice advice, training, and development services to organizations around the world.

For comments, suggestions, or support, please contact us via:
**support@ravenblackts.com**