Oracle Responsys Built-In Functions Guide

Release 6.24

ORACLE | responsys

Address permission requests, comments, or suggestions about Oracle Responsys documentation to docs@responsys.com.

ORACLE | responsys

Oracle Responsys Built-in Functions

A built-in function is a specific kind of text replacement field that can be used in a campaign message or form document as a placeholder for personalized recipient/submitter information that is automatically substituted when your campaign is distributed.

The following describes these functions and provides usage details and examples.

Using Built-in Functions in HTML Comments

Note that Oracle Responsys evaluates built-in functions, campaign variables, and text replacement fields enclosed in HTML comments just as it evaluates those that appear in "visible" text.

Example: If your HTML message includes a comment like this:

```
<!-- SocialSecurityNumber: $lookup(SSN)$ -->
```

Then the delivered HTML message will contain something like this:

```
<!-- SocialSecurityNumber: 503-55-1212 -->
```

As a comment, the information will not be displayed in the message itself; but it will be available to anyone who views the "source code" for the message.

Nesting Built-in Functions

When you nest functions, make sure only the outermost function is enclosed in dollar signs, as in this example:

```
$cond(eq(lookup(Title), Dr.),
  concat(lookup(Title), space(), lookup(LastName)),
  lookup(FirstName))$
```

Splitting Built-in Functions across Lines

Even though some examples in this document show long functions split across several lines, you must make sure both enclosing dollar signs are on the same physical line in your campaign or form document. They must not enclose a line break character (also known as a "hard return"). The example above, shown on several lines for clarity, should actually appear in your document like this:

```
$cond(eq(lookup(Title), Dr.), concat(lookup(Title), space(), lookup(LastName)),
   lookup(FirstName))$
```

The width of your editor's work area may force part of the total function string (`$cond(...)$`) to "wrap" onto multiple lines, but it **must not** contain any hard returns; otherwise it will not be evaluated as a built-in function, and your recipients will see the code, rather than the expected results.

Built-in Functions

add() function

Usage

```
$add(value1, value2, ...)$
```

This function adds the specified values and returns the result.

and() function

Usage

```
$and(value1, value2, ...)$
```

This function returns a 1 (one) if **all** of the specified values are one of the following (without regard to case):

```
1, y, yes, t, true
```

If even one of the specified values is not among those representations of “true” this function returns a 0 (zero).

avg() function

Usage

```
$avg(value1, value2, ...)$
```

This function returns the average of the specified values.

base64encode() function

Usage

```
$base64encode(string)$
```

This function converts certain characters in the specified string to a base64 encoding of those characters. This can be used to hide personal information included in a query string.

```
http://mycompany.com/optout.jsp?$base64encode(email=,lookup(email),id=,lookup(custid))$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

bazaarvoiceauthstring() function

Usage

```
$bazaarvoiceauthstring(BazaarVoice key String,Recipient Authentication string)$
```

Returns a string that represents a visitor identifier according to our BazaarVoice partner's authentication string specification. This function takes two input parameters, first the BazaarVoice key string, which they provide to each of their clients and second, a customer identifier, which can be either an alphanumeric customer identifier or email address, and should represent the unique identifier that BazaarVoice and Oracle Responsys clients use to track individual email recipients and website visitors. For more information, contact your BazaarVoice representative to better understand the use of their authentication string.

The Recipient Authentication string should take the following form. **Note**: See the BazaarVoice's documentation for more information.

Authentication string: date=YYYYMMDD&userid=<user id>&options...

Example:

$setvars(key, nm7t99d)$

$setvars(userid, lookup(customer_id))$

$setvars(d, todayformat(0,yyyyMMdd))$

$bazaarvoiceauthstring(lookup(key), concat(date=,lookup(d),&userid=,lookup(userid)))$

between() function

Usage

```
$between(testValue, value1, value2)$
```

testValue, **value1**, and **value2** are numbers (integer or floating point).

Note: You cannot use this function to compare strings.

This function returns 1 (one) if **testValue** is between **value1** and **value2**—specifically, if **testValue** is both greater than or equal to **value1** and less than or equal to **value2**; otherwise, this function returns 0 (zero).

Example: Each of the following function calls returns 1:

```
$between(3, 2, 4)$
$between(3, 3, 3)$
$between(round(3.5), 3, 4.0)$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

campaignid() function

Usage

```
$campaignid()$
```

This built-in function returns the internal system identifier of the current campaign (including the launch ID).

campaignmarketingprogram() function

Usage

```
$campaignmarketingprogram()$
```

This function inserts the campaign category value for “marketing program” that is set in the campaign wizard when the campaign is created. This can be useful when passed as a variable in URLs for websites that have tracking software.

Example

```
<a href=http://yoursite.com/page?cid=123&cat=$campaignmarketingprogram()$>…
```

campaignmarketingstrategy() function

Usage

```
$campaignmarketingstrategy()$
```

This function inserts the campaign category value for “marketing strategy” that is set in the campaign wizard when the campaign is created. This can be useful when passed as a variable in URLs for websites that have tracking software.

Example

```
<a href=http://yoursite.com/page?cid=123&cat=$campaignmarketingstrategy()$>…
```

campaignname() function

Usage

```
$campaignname()$
```

This built-in function returns the name of the current campaign.

capitalizewords() function

Usage

```
$capitalizewords(textString)$
```

This built-in function returns **textString** with the first letter of each word converted to uppercase.

This function could be used to make sure a multi-word name is properly capitalized, as in:

```
Dear $capitalizewords(lookup(Name))$,

Thank you for your order!
```

See also lowercase(), uppercase(), and leadingcapital().

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

clickthrough() function

Usage

```
$clickthrough(linkName)$
$clickthrough(linkName, additionalInfo)$
```

linkName is an entry from the link table for your campaign.

The optional **additionalInfo** is a comma-delimited list of names and name-value pairs, in any order, as in:

```
name1, name2=value, name3=value, name4, ...
```

Each name specified without a value (like **name1** and **name4** in the example above) is either a field name or a campaign variable name.

Each name in a name-value pair (like **name2** and **name3** above) is a text replacement field (`$text$`) that appears in the link URL for the specified **linkName** in the link table.

This built-in function returns a personalized form URL, which:

- ❑ Records the fact that the specific user clicked a specific link.
- ❑ Optionally updates additional fields in the link tracking table.
- ❑ Redirects the user to the (optionally customized) destination URL.

Typically, the `clickthrough()` function is used within an `<A HREF>` tag, as in:

```
<A HREF="$clickthrough(golfclubs)$">Click here</A>
```

If the destination URL requires additional information about the recipient—a coupon type or a discount level, for example—you can include that information as replacement fields in the link table. **Example:** to pass a coupon identifier, you must include the replacement field (`$couponId$`) in the link URL you specify in the link table, as in:

```
http://www.plantco.com/products/buy?sku=30021&coupon=$couponId$
```

and specify the replacement fields in the `clickthrough()` call, as in:

```
<A HREF="$clickthrough(rose052, couponId)$">Click here</A>
```

As another example, say your link table contains an offer named SpringDeals with this link URL:

```
http://www.mycom.com/offers/spring.asp?$custId$
```

You could customize the spring.asp page for each recipient who clicks on the offer by passing the appropriate identifier (`custId`) like this:

```
<A HREF="$clickthrough(SpringDeals, custId=758829)$")Tell me more!</A>
```

Notes: You must include the replacement fields in the URL column of the link table, and specify the same fields in the `clickthrough()` call for the link tracking to work.

The link name must be the first argument in the `clickthrough()` call, but you do not need to specify the replacement fields in the same order in which they appear in the URL.

Be sure to enclose the entire HREF string in double quotation marks, as shown above.

If the link tracking table contains fields with the same names as those specified in the **additionalNames** parameter, those fields in the link tracking table will be filled with the corresponding values from the distribution list or supplemental data sources when the user clicks the tracked link.

These are the special characters allowed for link names added to a link table:

A-Z a-z 0-9 ! - = @ _ [ ] { }

charat() function

Usage

```
$charat(string, index)$
```

Returns the character at the index of a given string. The index is zero-based.

commalist() function

Usage

```
$commalist(string1, string2, ...)$
```

This built-in function concatenates the listed elements in the form “A, B, C, and D.”

Example: If

```
$lookuprecords(Garden, Tools, Email, Sue@mycom.com, itemsOrdered)$
```

returns

“shovel,rake,hoe”

then

```
$commalist(lookuprecords(Garden, Tools, Email, Sue@mycom.com, itemsOrdered))$
```

returns

“shovel, rake, and hoe”

and

```
$commalist(fork or weeder, lookuprecords(Garden, Tools, Email, Sue@mycom.com,
    itemsOrdered), pick)$
```

returns

"fork or weeder, shovel, rake, hoe, and pick."

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

concat() function

Usage

```
$concat(stringList)$
```

stringList is a comma-delimited list of strings to be concatenated.

This function returns a single string composed of the multiple strings in **stringList**. Any empty string in **stringList** is replaced with a single space character in the resulting string.

You could use `concat()` to construct a salutation based on whether the recipient is a doctor (all on a single line):

```
$cond(eq(lookup(Title), Dr.),
  concat(lookup(Title), space(), lookup(LastName)),
  lookup(FirstName))$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

cond() function

Usage

```
$cond(testValue, match, noMatch)$
```

testValue is the value you want to test.

match is the value you want returned if the specified **testValue** is one of the following (without regard to case):

```
1, y, yes, t, true
```

noMatch is the value you want returned if the specified **testValue** is not equal to any of the values listed above.

Typically, `cond()` is used to convert the presence or absence of an indicator to an appropriate text string.

Example: If the `TellMeMore` field contains "Yes" then

```
We're $cond(lookup(TellMeMore), happy, sorry)$ to hear you
$cond(lookup(TellMeMore), are, aren't)$ interested in our services.
```

produces

```
We're happy to hear you are interested in our services.
```

If the `TellMeMore` field is empty – or if it contains any value besides `1`, `y`, `yes`, `t`, or `true` – the result is:

```
We're sorry to hear you aren't interested in our services.
```

Other Values

If you need to test for a different value, use the results of an `eq()` function call as the first argument to `cond()`, as in:

```
$cond(eq(lookup(color), blue), blue, not blue)$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

Multiple Values

If you need to test for multiple values, you can use the `select()` built-in function.

containsvalue() function

Usage

```
$containsvalue(nameList)$
```

`nameList` is a comma-delimited list of field names and campaign variable names. Each of the names that is a field must be in the distribution list or in a supplemental data source linked through **`data extraction`**.

This function returns a 1 (one) if **any** of the specified names contains a value, or a 0 (zero) if **all** of the names are empty.

count() function

Usage

```
$count(valueList)$
```

This function returns the number of comma-delimited elements in **valueList**.

dateformat() function

Usage

```
$dateformat(datevalue, offset, format)$
```

datevalue is a date value extracted from a timestamp field in the campaign list data

offset is 0 (for today), +n, or -n. (n is an integer number of days.)

format is a logical combination of the specifiers in the table below. The format string cannot include commas.

Examples:

```
$dateformat(lookup(lastpurchase_date),5,yyyy-MM-dd)$
$dateformat(lookup(lastpurchase_date),-5,yyyy-MM-dd)$
```

See `todayformat()` built-in for details on the time **format** option specifiers.

div() function

Usage

```
$div(value1, value2)$
```

This function divides **value1** by **value2** and returns the result.

Example:

```
$div(6, 3)$
```

returns

```
2
```

divformat() function

Usage

```
$divformat(dividend, divisor [,format, locale] )$
```

This function divides value1 by value2 and returns the result.

dividend is the number to be divided

divisor is the number use to divide the dividend

format is an optional formatting code consisting of [width][.precision] value. The optional width is a non-negative integer indicating the minimum number of characters to be written to the output. The optional precision is a non-negative integer used to restrict the number of decimal places. If no format is provided, there will be no minimum width and two decimal places.

locale is an optional two letter localization code (`en`, `de`, `fr`) from ISO 639-1. If no locale is provided, then the campaign locale is used.

Examples:

```
$divformat(lookup(amtpurchase),100,10.2)$
$divformat(lookup(amtpurchase),100)$
$divformat(lookup(amtpurchase),100,12.2,de)$
$divformat(lookup(amtpurchase),100,12.5)$
```

document() function

Usage

```
$document(folderName, documentName)$
$document(folderName, documentName, pairs(name1, value1, name2, value2, ...))$
$document(folderName, documentName1, documentName2, ...)$
$document(folderName, documentName1, documentName2, ..., pairs(name1, value1, name2
    , value2, ...))$
$document(url)$ where url is a fully qualified URL
```

This built-in function returns the entire contents of one or more documents for insertion into the document where the `document()` function is used.

If multiple documents are specified, their contents are concatenated, with files separated by `\n` (for text documents) or
 (for HTML documents). This distinguishes the `document()` function from the `documentnobr()` function.

You can use the optional `pairs()` subfunction to populate text replacement fields in the documents. Fields you specify in `pairs()` take precedence over like-named campaign variables and fields in the distribution list or supplemental data sources.

Example: If the distribution list contains a `FIRSTNAME` field, you can override the value in the distribution list by specifying `pairs(FIRSTNAME, Adam)` in the `document()` call. The typical use, of course, is to specify names that **`do not`** occur in the distribution list, supplemental data sources, or campaign variables.

Note that `pairs()` is useful **only** in the context of certain built-in functions, as shown here.

It is not necessary to include the document's extension (.txt, htm., or .aol)—the `document()` function accounts for extensions automatically. Also, if multiple documents share the same root name (MyDoc.txt and MyDoc.htm, for example), the `document()` function returns the one with the same file type as the document where the function is used.

The `$document(url)$` form of this function can be used to pull content from a fully qualified web resource location. This form of the built-in can have a slowing effect on the launch of a campaign due to internet latency, and the need to pull content from an external location. **Note**: While it may provide a useful solution in some cases, it should be used with care to prevent a slow launch. Be sure to test launch rates prior to committing to place a campaign that uses this form of the `document()` function into production.

Note: If the inserted document includes commas, it may be appropriate to enclose the `document()` function in a call to the `escapecommas()` function when using the returned value as an argument to another built-in function.

Caution: If the `document()` function is included in the return value of a `lookup()` call, and the `lookup()` call appears in a follow-up campaign that is sent in response to a form campaign, you should replace the `lookup()` call with a call to `lookuptable()`. This ensures that Oracle Responsys uses the appropriate format (HTML, AOL, or text) for the inserted document.

---

documentnobr() function

Usage

```
$documentnobr(folderName, documentName1, documentName2, ...)$
$documentnobr(folderName, documentName1, documentName2, ..., pairs(name1, value1, name2, value2, ...))$
```

This built-in function is identical to the `document()` function, except that the specified documents are inserted with **no** intervening characters.

Note: If the inserted document includes commas, it may be appropriate to enclose the `documentnobr()` function in a call to the `escapecommas()` function when using the returned value as an argument to another built-in function.

emaildomain() function

Usage

```
$emaildomain(emailAddress)$
```

emailAddress is a fully specified valid email address of the form **mailName@domain**.

This built-in function returns the domain portion of the specified **emailAddress** (everything to the right of the "@" sign).

Example:

```
$emaildomain(my.name@my-company.com)$
```

returns

```
my-company.com
```

and

```
$emaildomain(your.name@server.your-group.org)$
```

returns

```
server.your-group.org
```

empty() function

Usage

```
$empty(testString)$
```

This function returns 1 (one) if **testString** is empty, contains the empty string (""), or simply is not specified; otherwise, this function returns 0 (zero).

You could use `empty()` to construct a salutation based on the presence or absence of a first name or title in the customer's profile:

```
Dear $cond(empty(lookup(FirstName)), cond(empty(lookup(Title)), Mr. or Ms.,
   lookup(Title)), lookup(FirstName))$ $LastName$,
```

Depending on what information is available for customer Jan Smith, this line will result in one of the following salutations:

- Dear Jan Smith,
- Dear Dr. Smith,
- Dear Mr. or Ms. Smith,

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

endswith() function

Usage

```
$endswith(string, comparison_string)$
```

Returns a 1 if the string ends with the `comparison_string`, and 0 if not.

eq() function

Usage

```
$eq(value1, value2)$
```

value1 and **value2** are the values you want to test for equality.

This function returns 1 (one) if the specified values are equal (disregarding case differences), or 0 (zero) if they are not equal. If you do not specify exactly two arguments, `eq()` returns the empty string ("").

Example: both of the following function calls return 1:

```
$eq(fred, FRED)$
$eq(add(2, 1), 3)$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

escapecommas() function

Usage

```
$escapecommas(value1, value2, ...)$
```

This function replaces each comma in the specified list of values with a special token that prevents the comma from being interpreted as an argument delimiter. After the function is evaluated completely, the tokens are replaced with commas in the final results.

A common use is to pass a date (like January 1, 2001) as a single argument to another function.

Example: if

```
$lookuprecords(HR, Holidays2001, Day, Valentines, Date)$,
$lookuprecords(HR, Holidays2001, Day, MothersDay, Date)$,
$lookuprecords(HR, Holidays2001, Day, FathersDay, Date)$
```

returns

"February 14, 2001, May 13, 2001, June 17, 2001"

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

externalcampaigncode() function

Usage

```
$externalcampaigncode()$
```

This built-in function returns the external campaign code or identifier that is specified by users as part of the campaign definition. This built-in is helpful for use with the External Tracking feature or for direct insertion into campaign messages for use with web analytics tracking systems (as shown in the example below).

Example

```
<a href="http://xyz.com/path/to/page.jsp?cid=$externalcampaigncode()$">
```

firstname() function

Usage

```
$firstname(nameString)$
```

nameString is a text string having one of the following forms:

```
First Last
   Last, First
   First M. Last
   Last, First M.
```

This built-in function returns the first-name portion of **nameString**.

Example: all of the following function calls return "Jane":

```
$firstname(Jane Doe)$
   $firstname(Doe, Jane)$
   $firstname(Jane B. Doe)$
   $firstname(Doe, Jane B.)$
   $leadingcapital(firstname(doe, jane))$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

foreach() function

Usage

```
$foreach(loopVariableName, loopValueList, folderName, documentName)$
   $foreach(loopVariableName, pairslist(fieldCount, fieldList, valueList), folderName, documentName)$
```

loopVariableName is a name for the loop-control variable.

loopValueList is a comma-delimited list of values that you want to use within the specified document as you process that document once for each value. The number of items in the value list determines the number of evaluation loops. In each loop (each evaluation) the current value from **loopValueList** is available for use within the specified document as `$lookup(loopVariableName)$`.

When passed a loop value list, the `foreach()` function evaluates the specified document once for each of the specified loop values and returns the concatenated result. Instances of the document are separated by `\n` (for text documents) or `<BR>` (for HTML documents). This distinguishes the `foreach()` function from the `foreachnobr()` function.

Example: Say the Purchases table uniquely records each purchase at an ice cream parlor and the corresponding reward, along with the purchaser's ID, the store number, and the date of purchase. Along

with other information, your distribution list includes customer IDs (which are the same as the purchaser ID in the Purchases table). You want to send each purchaser a monthly message listing rewards for all of their purchases.

The document Visit.htm contains the following lines:

```
<html><body>
<p>
Date:    $lookuptable(My Folder, Purchases, PurchNum, lookup(purchase), PurchDate)$ <br>
Store:   $lookuptable(My Folder, Purchases, PurchNum, lookup(purchase), StoreNum)$ <br>
Reward:  $lookuptable(My Folder, Purchases, PurchNum, lookup(purchase), Reward)$
</p>
</body></html>
```

Your campaign message, Statement.htm, includes the following lines:

```
<p>Thanks for visiting Just Desserts this month! We'd like to thank you by offering the following
    rewards.</p>
<!-- Keep the entire foreachnobr() call, including both dollar signs, on a single line. -->
$foreachnobr(purchase, lookuprecords(My Folder, Purchases, Purchaser, lookup(CustID), PurchNum),
    My Folder, Visit)$ <p>Be sure to drop in again soon to take advantage of this offer!</p>
```

For each record in your distribution list, `lookup(CustID)` returns the corresponding ID, which is used by `lookuprecords()` to get a list of all the purchase numbers for that customer. For each of those purchase numbers, `foreach()` adds another instance of Visit.htm into Statment.htm, and each of those instances uses the specific value assigned to `purchase` (the loop-control variable) for that instance.

Important: When you nest functions, only the outermost function is enclosed in dollar signs.

foreach() function – Alternative Usage

To make the values of multiple variables available in each "loop" you can use the optional `pairslist() subfunction` in place of the **`valueList`** parameter:

```
$foreach(loopVariableName, pairslist(fieldCount, fieldList, valueList), folderName,
    documentName)$
```

`fieldCount` is the number of fields you want to make available, and **`fieldList`** is a comma-delimited list naming those fields; **`fieldList`** must contain the number of fieldnames specified by **`fieldCount`**.

`valueList` is a list of values you want to assign to the named fields, where each loop uses enough values from the list to assign a value to each of the specified fields. If the number of values is not a multiple of **`fieldCount`**, each field that is "unfilled" on the last loop will be assigned the empty string ("").

Example: `pairslist(3,a,b,c,1,2,3,4,5,6,7)` sets a to 1, b to 2, and c to 3 on the first loop; a to 4, b to 5, and c to 6 on the second loop; a to 7, b to "", and c to "" on the third loop. (Typically, of course, you would obtain the value list from a database lookup, rather than enumerating them as shown in this example.) Note that the optional `pairslist() subfunction` is useful **only** in the context of the `foreach()` function, as shown here.

Given the example used earlier, the `pairslist()` subfunction eliminates the need to call `lookuptable()` three times in every evaluation of Visit.htm. Instead, the `foreach()` call in Statement.htm returns the `PurchDate`, `StoreNum`, and Reward values for all of the iterations in a single database lookup. Another potential advantage is that you don't have to maintain a unique purchase number in the Purchases table.

So Statement.htm includes the following lines,

```
<p>Thanks for visiting Just Desserts this month! We'd like to thank you by offering the following
    rewards:</p>

<!-- Keep the entire foreachnobr() call, including both dollar signs, on a single line. -->
$foreachnobr(purchase, pairslist(3, date, store, reward, lookuprecords(My Folder, Purchases,
    Purchaser, lookup(CustID), PurchDate, StoreNum, Reward)), My Folder, Visit)$

<p>Be sure to drop in again soon to take advantage of this offer!</p>
```

...and Visit.htm contains:

```
<html><body>
$setvars(lookup(purchase))$
<p>
Date:    $lookup(date)$ <br>
Store:   $lookup(store)$ <br>
Reward:  $lookup(reward)$
</p>
</body></html>
```

See setvars() for related information.

Important: When you nest functions, only the outermost function is enclosed in dollar signs.

foreachnobr() function

Usage

```
$foreachnobr(loopVariableName, valueList, folderName, documentName)$
    $foreachnobr(loopVariableName, pairslist(fieldCount, fieldList, valueList), fol
    derName, documentName)$
```

This built-in function is identical to the `foreach()` function, except that instances of the specified document are inserted with **no** intervening characters.

formlink() function

Usage

```
$formlink(<campaignName>, email-address_)$
    $formlink(<formName>, first-name)$
```

You can generate links to personalized versions of this form by including the $formlink(<name of form>, fields)$ built-in function in your campaign message document, where the fields argument allows you to specify what data is available for personalization of the form. Specifically, the fields argument is optional and allows zero or more values to be passed to the form for personalization when the form is displayed. There are two ways to provide personalization data to a form with this built-in function:

- Pass data from the distribution list or supplemental data sources by specifying the name of a field in the distribution list or supplemental data sources. Example: $formlink(FormExample, first_name)$ makes the value of the first_name field available during the personalization of the form. Any reference to $first_name$ or lookup(first_name)$ in the form will result in the substitution of the first_name value for the recipient of the campaign message.
- Pass a literal value for personalization of the form by setting a variable name to a literal value. Example: $formlink(FormExample, id=123)$ allows you to insert a value of 123 wherever a reference to id is found in the form document.

You can generate a link to a blank form by using $formlink(FormExample)$.

Note: In the copy above, FormExample is used as an example of a form name.

ge() function

Usage

```
$ge(value1, value2)$
```

value1 and **value2** are numbers (integer or floating point).

This function returns 1 (one) if **value1** is greater than or equal to **value2**. If **value1** is less than **value2**, this function returns 0 (zero).

Note: You cannot use this function to compare strings.

generateContentFromXML() function

Usage

```
$generateContentFromXML(XML_Data, XSLT_Rules)$
```

This function is available for transforming input XML data into HTML or text content that can be inserted into a campaign message.

The basis for this transformation is a set of XSLT rules. The XML data and XSLT rules can be present in campaign list fields or as campaign variables (defined via the campaign wizard Defaults & Variables screen).

This allows for the use of XML data for each recipient of a campaign message to be converted into HTML and text according to any set of rules. It opens up the possibility for looping of multiple elements of content without the need for a `foreach()` function – allowing all content insertion rules to be defined in terms of a single campaign template, a single set of XSLT rules, and an incoming XML data feed.

Example

```
$generateContentFromXML(lookup(XML),lookup(XSL_Rules)$
```

Note: This function has an incremental slowing effect on the launch rate for a campaign. Be careful not to overuse this function in your templates if launch rate and overall launch duration is important to you. Be sure to test launch rates prior to committing to a campaign template that uses multiple instances of this function.

Tip: Place your XSLT rules in a campaign variable so it can be used for each recipient in a distribution list.

getURLContent () function

Usage

```
$getURLContent(URL, ErrorCode, argument1, value1, argument2, value2, ...)$
```

This function makes an HTTP or HTTPS request on an external service URL, and returns the content of that response.

- **URL** – Must start with http:// or https://.
- **Query string arguments** – Can be specified as pairs of argument/values.
- **ErrorCode** – Represents a value, which if returned by the service, causes the campaign to abort its launch.

Note: This function is a controlled feature, and must be enabled for your account to be functional. It must be used with care since any latency in the response of the external service can slow down a campaign launch. Therefore, it is best used for triggered messages rather than large-batch launched. For static content, consider using the `$document(url)$` function as well, since `document()` has some caching benefits that `getURLContent()` does not support.

Example

Assume that you are working with a product recommendation company that offers a web service for product recommendations. You can request recommendations for each customer ID in your distribution list.

```
$getURLContent(http://someRecommendationService.com/getproducts, ABORT_LAUNCH,
customerID, lookup(customerID))$
```

If necessary, you can feed the result of this call to `generateContentFromXML()` in order to transform an XML response into HTML content.

gt() function

Usage

```
$gt(value1, value2)$
```

value1 and **value2** are numbers (integer or floating point).

This function returns 1 (one) if **value1** is strictly greater than **value2**. If **value1** is less than or equal to **value2**, this function returns 0 (zero).

Note: You cannot use this function to compare strings.

hex () function

Usage

```
$hex(string)$
```

Returns the hexadecimal encoding for an input string. This may be useful in obscuring certain customer attributes in campaign or form content.

Examples:

$hex(10305069) $ = 3130333035303639

$hex(product@responsys.com)$ = 70726f6475637440726573706f6e7379732e636f6d

indexof() function

Usage

```
$indexof(string, search_string [, start_index])$
```

Returns the index of the first instance of the search_string within the string.

Example: $indexof(scottscott,co)$ or $indexof(scottscott,co,4)$

launchid() function

Usage

```
$launchid()$
```

This built-in function returns the launch ID for the campaign in which it is used. Launch IDs are integer values, and for standard and test batch launches start at 1 and increment with each launch. There are special launch ID values for other types of campaign usage, as follows.

- Form-triggered campaign messages: -1
- Real-time campaign messages: -6
- Campaign Preview: -3

The `launchid()` built-in function can be useful when used as part of a set of tracking parameters that are passed via hyperlink URL querystrings to website tracking code.

le() function

Usage

```
$le(value1, value2)$
```

value1 and **value2** are numbers (integer or floating point).

This function returns 1 (one) if **value1** is less than or equal to **value2**. If **value1** is greater than **value2**, this function returns 0 (zero).

Note: You cannot use this function to compare strings.

leadingcapital() function

Usage

```
$leadingcapital(textString)$
```

This built-in function returns **textString** with the first letter converted to uppercase.

This function could be used to make sure a name is properly capitalized in a follow-up letter, as in:

```
Dear $leadingcapital(lookup(GivenName))$,
Thank you for your order!
```

See also `lowercase()` and `uppercase()`.

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

listcontains() function

Usage

```
$listcontains(testValue, valueList)$
```

valueList is a comma-delimited list of values in which you want to check for **testValue**.

This built-in function returns 1 (one) if any of the listed values is the same as the specified **testValue**; otherwise, this function returns 0 (zero). If you specify fewer than two values (including **testValue**), this function returns the empty string ("").

You could use `listcontains()` to send a document to a recipient interested in one of a group of offers, as in:

```
$cond(listcontains(shampoo, lookup(ProductsUsed)), document(HairCare, Shampoo),
   nothing())$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

lookup() function

Usage

```
$lookup(name)$
```

name is a field name, a campaign variable name, a hidden field (in a form campaign), or a "local" variable defined with a call to the `setvars()` function. If **name** is a field, it must be in the distribution list or in a supplemental data source linked through data extraction. If **name** is a campaign variable, a hidden field, or a local variable, the distribution list or supplemental data source will not be checked.

This built-in function returns the current value of the specified field or campaign variable. If the specified field or campaign variable does not exist, this function returns the empty string ("").

This function is typically used to determine the value of a field (or campaign variable) so as to pass that value to another function. If you omit the use of `lookup()`, you will pass the **name** of the field rather than its **value**.

Example: if the `person` field contains "jane doe" then

```
$leadingcapital(firstname(person))$
```

returns

"Person"

which is probably not the desired effect. Instead, use

```
$leadingcapital(firstname(lookup(person)))$
```

which returns

"Jane"

Caution: If you use `lookup()` in a follow-up campaign sent in response to a form campaign, and the value returned by the `lookup()` call includes a `document()` function, you should replace the `lookup()` call with a call to `lookuptable()`. This ensures that Oracle Responsys uses the appropriate format (HTML, AOL, or text) for the inserted document.

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

---

lookuprecords() function

Usage

```
$lookuprecords(folderName, dataSource, lookupField, lookupValue, queryField1, query
   Field2, ...)$
```

```
$lookuprecords(folderName, dataSource, pairs(field1, value1, field2, value2, ...), queryField1, queryField2, ...)$
```

folderName is the folder that contains the specified **dataSource**.

dataSource is the data source you want to search.

Whether you specify a single name-value pair

```
lookupField, lookupValue
```

or multiple name-value pairs

```
pairs(field1, value1, field2, value2, ...)
```

Each pair consists of the field you want to check and the value you want to find. Note that the optional `pairs()` subfunction is useful **only** in the context of certain built-in functions, as shown here. Note, too, that the order in which the pairs are evaluated is a function of database query optimization and may not be the same as the order in which they're specified or even the same every time the same pairs are specified.

queryField1, queryField2, ... is a list of fields from which you want to retrieve the corresponding values.

This built-in function returns, as a single comma-delimited list, all values from the specified **queryFields** in all records in which **lookupValue** is found in **lookupField**.

Notes: When you nest functions, only the outermost function is enclosed in dollar signs.

Since the `lookuprecords()` function may return a string that includes commas, it is sometimes appropriate to enclose it in a call to the `escapecommas()` function when using the returned value as an argument to another built-in function.

We recommend that you always use one of the key non-empty fields to identify a record, such as RIID_, CUSTOMER_ID_ or EMAIL_ADDRESS_, as the first queryFields in the lookuprecords() function call to make sure the comma-separated returned values are in the expected order.

Example:

$lookuprecords(Childwear, Responses, State, CA, EMAIL_ADDRESS_, Item, Price, Qty)$

Caution: The Responsys systems joins all the targeting and personalization data sources specified for a campaign to create a single table called the Work List table. If one of the data sources participating in the join has multiple records per recipient, the system selects the latest matching record based on the DEK. The Work List table has one record per email recipient and has all the columns from the data

sources participating in the join. To use this data for personalization in built-ins, we recommend using the lookup() built-in function.

The lookuprecords() built-in fetches the data source specified in the built-in definition on a per recipient basis and does not take advantage of the Work List table that has been created and optimized for the campaign launch. lookuprecords() built-in can cause highly concurrent database queries and can slow down the launch of the campaign. For these reasons, we recommend using the lookuprecords() built-in with caution and only when a specific data source cannot be added to the campaign as a personalization data source (typically when the data source has multiple records per recipient that are to be used for personalization). In most cases adding the required data sources as a personalization data source in the campaign and using the lookup() built-in should satisfy your needs.

lookuptable() function

Usage

```
$lookuptable(folderName, dataSource, lookupField, lookupValues, queryField)$
```

folder is the folder that contains the data source you want to search.

dataSource is the data source you want to search.

lookupField is the field you want to check for the specified **lookupValues**.

lookupValues is a comma-delimited list of one or more values you want to find in the specified **lookupField**.

queryField is the field from which you want to retrieve the corresponding value.

Note: If **lookupValues** specifies a **single** value to look for, this built-in function returns the value that corresponds to the first instance of that value found in **lookupField**.

The following example opens the SalesReps table in the Sales folder, checks the `Region` field for the first instance of the value “Midwest,” and returns the value found in the `RepName` field:

```
$lookuptable(Sales, SalesReps, Region, Midwest, RepName)$
```

If **lookupValues** specifies **multiple** values to look for, `lookuptable()` returns a comma-delimited list of values found, in which each returned value **n** corresponds to the first found instance of the **n**-th value specified in **lookupValue**.

This is equivalent to forming a list of multiple calls to `lookuptable()` with single lookup values.

Example: if

```
$lookuptable(Sales, SalesReps, Region, Northwest, RepName)$
```

returns

"Sarah Smith"

and

```
$lookuptable(Sales, SalesReps, Region, Midwest, RepName)$
```

returns

"Bob Brown"

and

```
$lookuptable(Sales, SalesReps, Region, Southwest, RepName)$
```

returns

"Mary Jones"

then

```
$lookuptable(Sales, SalesReps, Region, Northwest, Midwest, Southwest, RepName)
```

returns

"Sarah Smith, Bob Brown, Mary Jones"

Note: Since the `lookuptable()` function may return a string that includes commas, it is often appropriate to enclose it in a call to the `escapecommas()` function when using the returned value as an argument to another built-in function.

Caution: The Responsys systems joins all the targeting and personalization data sources specified for a campaign to create a single table called the Work List table. If one of the data sources participating in the join has multiple records per recipient, the system selects the latest matching record based on the DEK. The Work List table has one record per email recipient and has all the columns from the data sources participating in the join. To use this data for personalization in built-ins, we recommend using the lookup() built-in function.

lookuptable() built-in fetches the data source specified in the built-in definition on a per recipient basis and does not take advantage of the Work List table that has been created and optimized for the campaign launch. lookuptable() built-in can cause highly concurrent database queries and can slow down the launch of the campaign. For these reasons, we recommend using the lookuptable() built-in

with caution and only when a specific data source cannot be added to the campaign as a personalization data source. In most cases adding the required data sources as a personalization data source in the campaign and using the lookup() built-in should satisfy your needs.

lowercase() function

Usage

```
$lowercase(string)$
```

This built-in function returns the specified **string** with any English characters converted to their lowercase equivalents.

See also `uppercase()` and `leadingcapital()`.

lt() function

Usage

```
$lt(value1, value2)$
```

value1 and **value2** are numbers (integer or floating point).

This function returns 1 (one) if **value1** is strictly less than **value2**. If **value1** is greater than or equal to **value2**, this function returns 0 (zero).

Note: You cannot use this function to compare strings.

max() function

Usage

```
$max(value1, value2, ...)$
```

This function returns the greatest of the specified values.

messageformat() function

Usage

```
$messageformat()$
```

This function returns "H" (HTML), "T" (plain text), "M" (HTML and plain text), or "A" (AOL format), indicating the format of the message sent to a given recipient.

min() function

Usage

```
$min(value1, value2, ...)$
```

This function returns the least of the specified values.

mod() function

Usage

```
$mod(dividend, divisor, decimal_places)$
```

Returns the remainder of the operation: **dividend** % **divisor** rounded to the specified **decimal_places**

More information about the % operator is provided in the Java Language Specification and additional background is provided here.

Examples

```
mod(1893892,188): $mod(1893892,188)$
mod(lookup(RIID_),188): $mod(lookup(RIID_),188)$
mod(lookup(RIID_),188,2): $mod(lookup(RIID_),188,2)$
mod(lookup(RIID_),188),4: $mod(lookup(RIID_),188,4)$
```

mul() function

Usage

```
$mul(value1, value2, ...)$
```

This function multiplies the specified values and returns the result.

ne() function

Usage

```
$ne(value1, value2)$
```

value1 and **value2** are the values you want to test for inequality.

This function returns 1 if the specified values **are not** equal, or 0 if they are equal. If you do not specify exactly two arguments, `ne()` returns the empty string ("").

Example: both of the following function calls return 1:

```
$ne(3, "3")$
   $ne(round(3.6), 3)$
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

nonemptyfields() function

Usage

```
$nonemptyfields(fieldName1, fieldName2, ...)$
```

For each recipient in the distribution list, this built-in function checks the specified fields in the current record, and returns the names of fields that contain a value.

You could use this built-in function to create dynamic content by specifying a complete set of **available** documents, but delivering only the **appropriate** documents for the current recipient.

Example:

```
$document(Newsletter, nonemptyfields(Business, Graphics, Games, Multimedia))$
```

If recipient `fred@freemail.com` has expressed an interest in games and multimedia, the corresponding fields in Fred's record in the distribution list contain some value. The `nonemptyfields()` function returns "Games, Multimedia" and the `document()` function returns the Games and Multimedia documents from the Newsletter folder.

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

nonemptyvalues() function

Usage

```
$nonemptyvalues(fieldName1, fieldName2, ...)$
```

This built-in function checks the specified fields in the current record, and returns the value from each field that contains a value.

You could use this built-in function to create dynamic content by specifying a complete set of **potential** values, but delivering only the **appropriate** values for the current record.

Example:

```
$nonemptyvalues(Business, Graphics, Games, Multimedia)$
```

If recipient fred@freemail.com has expressed an interest as Y in games and multimedia and null in Business and Graphics, the corresponding fields in Fred's record in the distribution list contain non null values in Games and Multimedia. The `nonemptyvalues()` function returns values of the field Games and Multimedia.

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

not() function

Usage

```
$not(value)$
    $not(value1, value2, ...)$
```

If you specify a single value, this function returns a 0 (zero) if that value is one of the following (without regard to case):

```
1, y, yes, t, true
```

If the specified value is not among those representations of “true” this function returns a 1 (one).

If you specify multiple values, this function returns a comma-delimited list of their logical negations.

Example

```
$not(1, 0, Y, x, TRUE, anything else, yes)$
```

returns

0,1,0,1,0,1,0

nothing() function

Usage

```
$nothing()$
```

This function returns a special token that safely represents an empty string. Use this function to prevent email skipping when a field or the result of a built-in expression is empty or null.

Normally, Oracle Responsys won't send a message that contains an empty or null replacement field; but sometimes it is appropriate for a replacement field to be empty.

Example 1:

Here is an example of typical use of this function within the listcontains() function:

```
$cond(listcontains(shampoo, lookup(ProductsUsed)), document(HairCare, Shampoo),
   nothing())$
```

In this example, if `listcontains()` returns zero (false), then the returned value returned will be the value of `nothing()`.

Example 2:

You can use the function by putting the following value in a campaign variable:

```
$nothing()$
```

For example, if the campaign has a variable called `MyField`. Setting this campaign variable to `$nothing()$` prevents the message from skipping when the field is null and is referenced like this: `$MyField$`.

numberformat() function

Usage

```
$numberformat(value,format[,groupSeparatorFlag,negativeParenthesesFlag[,locale]])$
```

`value` is the number to be formatted

`format` is an optional formatting code consisting the following value.

`[optional format flags:+-0][Width[[.Precision][Conversion]` value.

The optional format flags control whether the output contains a sign (+), whether the output is left justified (-), and whether zero padding is applicable (0).

The optional **width** is a non-negative integer indicating the minimum number of characters to be written to the output.

The optional **precision** is a non-negative integer used to restrict the number of decimal places. If no format is provided, there will be no minimum width and two decimal places.

Conversion defines the format of the number: f for decimal or e for exponential notation.

Example: +8.2f

groupSeparatorFlag is a flag (1 = true, 0 = false) that determines whether locale-specific number grouping separators (like commas) are used.

negativeParentheses is a flag (1 = true, 0 = false) that determines whether negative values are displayed in parentheses or not.

locale is an optional two letter localization code (`en`, `de`, `fr`) from ISO 639-1. If no locale is provided, then the campaign locale is used.

Examples

```
$numberformat(lookup(amtpurchase),10.2f,1,1,en)$
$numberformat(lookup(amtpurchase),10.2f,1,1,fr)$
$numberformat(lookup(amtpurchase),+10.2f,0,0,en)$
$numberformat(lookup(amtpurchase),-10.2f,1,0,en)$
```

or() function

Usage

```
$or(value1, value2, ...)$
```

This function returns a 1 (one) if one or more of the specified values are among the following (without regard to case):

```
1, y, yes, t, true
```

If **none** of the specified values is among those representations of “true” this function returns a 0 (zero).

outputencoding() function

Usage

```
$outputencoding(lookup(name))$
```

name is the name of the variable to protect.

This function protects an HTML customization variable on a form against Cross-Site Scripting (XSS) by using output encoding. The function performs output encoding by escaping characters for the personalization variable.

outputjsencoding() function

Usage

```
$outputjsencoding(lookup(name))$
```

name is the name of the variable to protect.

This function protects an JavaScript customization variable on a form against Cross-Site Scripting (XSS) by using output encoding. The function performs output encoding by escaping characters for the personalization variable.

prefilledform() function

Usage

```
$prefilledform(targetCampaignName)$
```

Important: The distribution or prefill list for **targetCampaignName** cannot be a “join”. Therefore, it is strongly recommended that you specify a simple table or external connector.

This built-in function returns a personalized form URL, which displays the prefilled message or form document for the specified target campaign or form.

Typically, the `prefilledform()` function is used within a hypertext link, as in:

```
<A HREF="$prefilledform(golfclubs)$">Tell me more!</A>
```

You could place this hypertext link in a letter, as an alternative to sending a personalized attachment.

rand() function

Usage

```
$rand(value)$
```

This function returns a random number between 0 (zero) and the specified **value**.

randomsubset() function

Usage

```
$randomsubset(defaultValue, maxSubsetSize, valueList)$
```

defaultValue is the string you want to use if the specified **valueList** is empty or if the value of **maxSubsetSize** is 0 or less.

maxSubsetSize is the number of items you want the function to return.

valueList is a comma-delimited list of values, as in:

```
value1, value2, value3, ...
```

This function returns a random subset of **maxSubsetSize** elements from the specified **valueList** (values 1 through **n**). If **valueList** is empty (or if the value of **maxSubsetSize** is 0 or less), `randomsubset()` returns the specified **defaultValue**.

The value list might be the result of another function call, as in:

```
$randomsubset(gardening, 3, nonemptyfields(antiques, astronomy, camping, coffee,
   computers, cooking, homerepairs, music, stamps, videos))$
```

Example: If,

```
$lookup(FavoriteDogs)$
```

Returns:

"afghans, dachshunds, collies, terriers, poodles, boxers"

Then:

```
$randomsubset(cats, 3, lookup(FavoriteDogs))$
```

Might return:

"boxers, afghans, poodles" or "afghans, dachshunds, terriers"

This could be used with `commalist()` as follows:

```
You like $commalist(randomsubset(cats, 3, lookup(FavoriteDogs)))$.
```

To return:

"You like boxers, afghans, and poodles." or "You like afghans, dachshunds, and terriers."

Conversely, if `$lookup(FavoriteDogs)$` returns the empty string (""), then:

```
You like $commalist(randomsubset(cats, 3, lookup(FavoriteDogs)))$.
```

Returns:

"You like cats."

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

replaceall() function

Usage

```
$replaceall(string, regex, replacement_string)$
```

Replaces all substrings of an input string that match the given regular expression with the given replacement.

Example

```
$replaceall(scoottscoott,oo,uu)$ = scuuttscuutt
$replaceall(123|345|456,\|,*)$: 123*345*456
$replaceall(123|345|456,\|,escapecommas(,))$: 123,345,456
```

replacefirst() function

Usage

```
$replacefirst(string, regex, replacement_string)$
```

Replaces the first substring of an input string that matches the given regular expression with the given replacement.

Example

```
$replacefirst(scoottscoott,oo,uu)$ = scuuttscoott
```

round() function

Usage

```
$round(value)$
```

This function rounds the specified **value** to the nearest integer and returns the result.

securedigest() function

Usage

```
$securedigest(value)$
$securedigest(value, algorithm)$
```

value is a the value you would like to generate a secure digest from.

algorithm is the hashing algorithm used to generate the digest. Two algorithms are supported: 1) Secure Hash Algorithm (SHA), which is the default if no algorithm is specified, and 2) Message-Digest algorithm 5 (MD5). Note that with the MD5 option, the function generates base64 encoded hash output. **Note**: In the function call, use SHA or MD5 to explicitly set the algorithm to be used.

This built-in function generates a one way digest by using two standard hashing algorithms. The function produces output in base64 encoding. This function can be used to deliver encrypted information anywhere in a campaign or form message. It may be useful to pass encrypted promotion codes in the query string of a link URL so the destination website can compare the encrypted promotion code to a list of authorized codes.

Examples

```
$securedigest(lookup(promotioncode))$
$securedigest(concat(lookup(customerid),lookup(promotioncode)), MD5)$
```

secureDigestAsHex() function

Usage

```
$secureDigestAsHex(value)$
$secureDigestAsHex(value, algorithm)$
```

value is a the value you would like to generate a secure digest from.

algorithm is the hashing algorithm used to generate the digest. Two algorithms are supported: 1) Secure Hash Algorithm (SHA), which is the default if no algorithm is specified, and 2) Message-Digest algorithm 5 (MD5). Note that with the MD5 option, the function generates base64 encoded hash output. **Note**: In the function call, use SHA or MD5 to explicitly set the algorithm to be used.

This built-in function generates a one way digest by using two standard hashing algorithms. The function produces output in hexadecimal encoding. This function can be used to deliver encrypted information anywhere in a campaign or form message. It may be useful to pass encrypted promotion codes in the

query string of a link URL so the destination website can compare the encrypted promotion code to a list of authorized codes.

Examples

```
$ secureDigestAsHex(lookup(promotioncode))$
$ secureDigestAsHex(concat(lookup(customerid),lookup(promotioncode)), MD5)$
```

select() function

Usage

```
$select(testValue, value1, return1, value2, return2, ...)$
   $select(testValue, value1, return1, value2, return2, ..., default)$
```

testValue is the value you want to test for.

value1, **value2**, ... are the values you want to compare to **testValue**.

return1, **return2**, ... are the values you want to return if the corresponding **valueN** matches **testValue**.

This built-in function compares the specified **testValue** to each of the specified values (**value1**, **value2**, ...) and returns the value (**return1**, **return2**, ...) that corresponds to the first matching value. If no value matches the specified **testValue**, the `select()` function returns the **default** value if it is specified; otherwise, `select()` returns the empty string ("").

If fewer than three (3) arguments are specified, `select()` returns the empty string (""). If an even number of arguments is specified, the last value is used as a default. If an odd number of arguments is specified, there is no default value.

The `select()` built-in function has the same effect as nesting calls to the `cond()` built-in function, but is much easier to use when you want to test for multiple values.

Example: You could use this (all on one line):

```
$select(lookup(Breed),
afghan, escapecommas(document(DogInfo, Afghans)),
boxer, escapecommas(document(DogInfo, Boxers)),
collie, escapecommas(document(DogInfo, Collies)),
dachshund, escapecommas(document(DogInfo, Dachshunds)),
doberman, escapecommas(document(DogInfo, Dobermans)),
escapecommas(document(GenInfo, AboutOurService)))$
```

rather than this (all on one line):

```
$cond(eq(lookup(Breed), afghan), escapecommas(document(DogInfo, Afghans)),
    cond(eq(lookup(Breed), boxer), escapecommas(document(DogInfo, Boxers)),
    cond(eq(lookup(Breed), collie), escapecommas(document(DogInfo, Collies)),
    cond(eq(lookup(Breed), dachshund), escapecommas(document(DogInfo, Dachshunds)),
    cond(eq(lookup(Breed), doberman), escapecommas(document(DogInfo, Dobermans)),
    escapecommas(document(GenInfo, AboutOurService)))))))$
```

You can also use a segment group name as the test value, as shown below (all on one line):

```
$select(lookup(OwnerSegment),
afghan, escapecommas(document(DogInfo, Afghans)),
boxer, escapecommas(document(DogInfo, Boxers)),
collie, escapecommas(document(DogInfo, Collies)),
dachshund, escapecommas(document(DogInfo, Dachshunds)),
doberman, escapecommas(document(DogInfo, Dobermans)),
escapecommas(document(GenInfo, AboutOurService)))$
```

Notes: Since the documents inserted in the examples above include commas, each call to the `document()` function is enclosed in a call to the `escapecommas()` function. This prevents the commas in the inserted documents from being interpreted incorrectly.

When you nest functions, only the outermost function is enclosed in dollar signs.

setglobalvars() function

Usage

```
$setglobalvars(name1, value1, name2, value2, ...)$
```

When passed a list of name-value pairs, this built-in function makes the specified variable values available globally during the personalization of a message (by means of the corresponding name, as in `$lookup(name1)$`) within the context of the current campaign document or subdocument.

The difference between the `setvars()` function and `setglobalvars()` is that `setlgobalvars()` sets a variable globally regardless of which subdocument it is located in. Conversely, the `setvars()` function only sets the value of a variable in the document or subdocument in which it is called.

Note: If a `setvars()` call in a subdocument uses the same variable name as a previously used `setglobalvars()` call, then the `setvars()` value will obscure the `setglobalvars()` value until the subdocument context is exited.

setvars() function

Usage

```
$setvars(name1, value1, name2, value2, ...)$
$setvars(lookup(loopVariableName))$
```

When passed a list of name-value pairs, this built-in function makes the specified values available (by means of the corresponding name, as in `$lookup(name)$`) within the context of the current document.

Example: You could use `setvars()` to create a convenient "alias" for an unwieldy built-in function call, to set up "local" variables, or simply to override the value of a field or campaign variable.

Note: The values are available only **after** the `setvars()` call, so in most cases the `setvars()` call should appear at the beginning of the document in which you want to use the named variables.

Caution: References to these local variables as `$name$` will look fine in preview mode (and in website campaigns), but they **will not** be replaced correctly in delivered messages. Be sure to use `$lookup(name)$` instead.

With "loop" Functions

In a document inserted by means of a call to the `foreach()` or `foreachnobr()` function, the current value of the loop-control variable is available for use in the inserted document as `$lookup(loopVariableName)$`.

If the `foreach()` or `foreachnobr()` call uses the optional `pairslist()` subfunction, you should start the inserted document with this statement:

```
$setvars(lookup(loopVariableName))$
```

The `lookup(loopVariableName)` call returns a string of the form `~Pairs~n`, an internal key for the current values of all the variables named in the `pairslist()` subfunction. The `setvars(~Pairs~n)` call makes those values available in the inserted document as `$lookup(localVariableName)$`.

See *Alternative Usage* on page 17 for related information.

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

---

space() function

Usage

```
$space()$
```

This function returns a single space (“ ”) character. It is typically used in conjunction with the `cond()` function, when one of the conditional values should be a single space.

startswith() function

Usage

```
$startswith(string, comparison_string)$
```

Returns a 1 if the string begins with the comparison_string and 0 if not.

stringlength() function

Usage

```
$stringlength(string)$
```

Returns the length of a string. Example: $stringlength(0123456789)$

sub() function

Usage

```
$sub(value1, value2)$
```

This function subtracts **`value2`** from **`value1`** and returns the result.

substring() function

Usage

```
$substring(string, start_index [,end_index])$
```

Returns the portion of a string including characters from start_index (inclusive) through end_index (exclusive: end_index-1). The index is zero-based, meaning the first character of the string has an index of 0. The end_index is optional.

Example

```
$substring(lookup(firstname), 3)$
```

todayformat() function

Usage

```
$todayformat(offset, format)$
```

offset is 0 (for today), +**n**, or −**n**. (**n** is an integer number of days.)

format is a logical combination of the specifiers in the table below. **Note:** The format string cannot include commas.

Note: As shown in the table, format specifiers are case-sensitive.

This built-in function returns today's date (optionally offset by n days) in the specified format.

Example: On March 23, 2005:

```
$todayformat(0, yyyy-MM-dd)$
```

returned

```
2005-03-23
```

Additional Examples

Specifiers	Date or time element	Examples
G	Era (AD or BC)	AD
yyyy	Year (4 digits)	2000
yy	Year (2 digits)	00
MMMM	Month in year (full name)	August
MMM	Month in year (short or abbreviated name)	Aug
MM	Month in year (2 digits)	08
M	Month in year (1 or 2 digits, as needed)	7
ww	Week in year (2 digits)	05
w	Week in year (1 or 2 digits, as needed)	5
W	Week in month	2
DDD	Day in year (3 digits)	189
D	Day in year (1, 2, or 3 digits, as needed)	189
dd	Day in month (2 digits)	09
d	Day in month (1 or 2 digits, as needed)	9
F	Day of week in month	2
EEEE	Day in week (full name)	Monday
E	Day in week (short or abbreviated name)	Mon
a	Before/after noon	PM
HH	Hour in day (0 through 23; 2 digits)	00
H	Hour in day (0 through 23; 1 or 2 digits, as needed)	0
kk	Hour in day (1 through 24; 2 digits)	24
k	Hour in day (1 through 24; 1 or 2 digits, as needed)	24
KK	Hour in AM/PM (0 through 11; 2 digits)	00
K	Hour in AM/PM (0 through 11; 1 or 2 digits, as needed)	0
hh	Hour in AM/PM (1 through 12; 2 digits)	12
h	Hour in AM/PM (1 through 12; 1 or 2 digits, as needed)	12
mm	Minute in hour (0 through 59; 2 digits)	30

m	Minute in hour (0 through 59; 1 or 2 digits, as needed)	30
ss	Second in hour (0 through 59; 2 digits)	55
s	Second in hour (0 through 59; 1 or 2 digits, as needed)	55
S	Millisecond (0 through 999)	978
zzzz	Time zone (full name, for named zones)	Pacific Daylight Time
ZZZZ	Time zone (GMT-offset, for unnamed zones)	GMT–08:00
z	Time zone (abbreviated zone name)	PDT
Z	Time zone (offset from GMT)	–0800

Additional Examples

Format	Result for June 1, 2005
`yyyy.MM.dd G 'at' HH:mm:ss z`	`2005.06.01 AD at 12:08:56 PDT`
`EEE MMM d ''yy`	`Wed, Jun 1 '05`
`h:mm a`	`12:08 PM`
`hh 'o''clock' a zzzz`	`12 o'clock PM Pacific Daylight Time`
`K:mm a z`	`0:08 PM PDT`
`yyyyy.MMMMMM.dd GGG hh:mm aaa`	`02005.June.01 AD 12:08 PM`
`EEE d MMM yyyy HH:mm:ss Z`	`Wed 1 Jun 2005 12:08:56-0700`
`yyMMddHHmmssZ`	`050601120856-0700`

Important: Remember **not** to use commas in the format string.

unique() function

Usage

```
$unique(itemList)$
```

itemList is a comma-delimited list of items to be purged of duplicates.

This built-in function removes all duplicates from **itemList**.

Example: If:

```
$lookuprecords(Childwear, Responses, Email, fred@mycom.com, interestAges)$
```

Returns:

"child,infant,child,toddler,infant"

Then:

```
$unique(lookuprecords(Childwear, Responses, Email, fred@mycom.com, interestAges))$
```

Returns:

"child,infant,toddler"

Notes: This function ignores case. In other words, "child," "Child," and "CHILD" are treated as duplicates, and `unique()` will return only one of them in its results.

When you nest functions, only the outermost function is enclosed in dollar signs.

uppercase() function

Usage

```
$uppercase(string)$
```

This built-in function returns the specified **`string`** with any English characters converted to their uppercase equivalents.

See also `lowercase()` and `leadingcapital()`.

urlencode() function

Usage

```
$urlencode(string)$
```

This function converts certain characters in the specified string to representations that can safely be used as part of a URL. This function leaves letters (a–z and A–Z) and digits (0–9) unchanged, changes spaces to plus signs (+), and converts all other characters to their hexadecimal equivalents, such as "%3D" for an equals sign (=).

You could use this function in passing data as part of a response campaign's redirect URL when that data might contain characters (like ampersands, questions marks, and equals signs) that have a special meaning in URLs.

Example: If your response form includes an "Ask Us" field where the recipient will probably type a question, and you want to pass that question to a script for further processing, you could specify a redirect URL like this on the campaign Acknowledgment page (on a single line):

```
http://myserver.mycompany.com/bin/askus.jsp?question=$urlencode(lookup(ASKUS))$&
    custid=$urlencode(lookup(CID))$
```

After the recipient's response has been processed by Oracle Responsys, her browser will be redirected to a URL like this:

```
http://myserver.mycompany.com/bin/askus.jsp?question=Do+you+accept+Diner%27s+Club%3F&
    custid=981%2D722
```

Note: When you nest functions, only the outermost function is enclosed in dollar signs.

varlist()

Usage

```
$varlist(variableCount, variableList, valueList)$
```

Where:

variableCount is the number of variables you want to make available.

variableList is a comma-delimited list naming those variables.

valueList is a list of values you want to assign to the named variables.

This built-in function can be used with the `setvars()` function to set the values of a number of variables with a single function call. This can be particularly helpful when the `valueList` argument is generated by a call to the `lookuptable()`, `lookuprecords()`, or `lookup()` functions.

Note: If there are more values than variable names, then the remaining values are placed in the last variable as a comma-separated string. If there are more variable names than values, some variables at the end of `variableList` will be empty.

Examples

```
$setvars(varlist(4,name,price,url,lookuprecords(folder,Products,sku,lookup(sku),name,p
    rice,url)))$
```

Will lookup a given product SKU in a given product table and populate name, price, and URL variables for use in a campaign message document.

```
$setvars(varlist(4,a,b,c,list,1,2,3,4,5,6,7))$
sets a to 1, b to 2, and c to 3, and  list to 4,5,6,7.
```