Documentation

Scriptlets

v1.4

Documentation for the XQueue system

XQueue GmbH, Christian-Pleß-Str. 11-13, 63069 Offenbach am Main, Germany

Inhalt und Ziel	This document provides syntax, components and examples for the newsletter programming language called „Scriptlets“
Typ	Documentation
Version	1.4
Autor	Marcus Beckerle, Anouar Haha
Erstellt	27.03.2015
Letzte Änderung	08.03.2021

Description

"Scriptlets" is the name of our programming language used within Maileon newsletters to realize extensive requirements on data processing and output. Scriptlets allow for example generating random numbers or UUIDs for each contact within a sendout in order to attach it to some link. It allows generating an (MD5) hash of data like the email address or encrypt data with some given key and supports complex processing of data like parsing a string as a datetime object, adding 7 days and printing it in some other format. Also accessing external data sources for each individual contact is possible in order to e.g. display individual products for each contact.
Thus, Scriptlets support a huge number of language elements and even if it is not possible to create an infinite loop, you can use loops, e.g. when iterating over the items of some shopping cart in order to calculate the overall sum or to display the items in a newsletter.

The formal description is provided as BNF in the next section, however, most readers will benefit at the beginning more of some unformal description or the examples later on. Scriptlets are always wrapped in double square brackets [[ ]]. Inside the brackets, a Scriptlet is started with a "%" sign to avoid mixing up regular Maileon-Mergetags and Scriptlets. For Scriptlets a preorder syntax has been chosen, this means that the operator is in the beginning of an expression and the parameters follow behind, separated by a whitespace " ". If an parameter is an expression with some operator itself, it has to be wrapped in single round brackets ().

Simple examples:

```
[[ % md5 'Hello world!']]
[[ % md5 (contact 'EMAIL')]]
```

For date and time formats please refer to
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/SimpleDateFormat.html

Formal Description

This section defines the BNF of the language elements.

```
<expr>                    ::= <void_expr> | <none_void_expr>
<void_expr>               ::= <conditional_statement> | <foreach_statement>
<conditional_statement>   ::= <if_statement> | <elseif_statement> |
                              <else_statement> | <endif_statement>
<if_statement>            ::= "if" {logical_expr>
<elseif_statement>        ::= "elseif" {logical_expr>
<else_statement>          ::= "else"
<endif_statement>         ::= "endif"
<foreach_statement>       ::= <foreach_expr> | <end_foreach> | break
<foreach_expr>            ::= "foreach" <var_expr> "in" <sequence_expr>
<end_foreach>             ::= "endforeach"
<none_void_expr>          ::= <typed_expr> | <untyped_expr>
<typed_expr>              ::= <string_expr> | <date_expr> | <datetime_expr> |
                              <logical_expr> | <numeric_expr> | <sequence_expr>
<untyped_expr>            ::= <assign_expr> | <hash_expr> | <var_expr>
<assign_expr>             ::= "set" <var_expr> <none_void_expr>
<var_expr>                ::= "$" litteral
<string_expr>             ::= string | <string_function> | <untyped_expr>
<numeric_expr>            ::= number | <numeric_function> | <untyped_expr>
<logical_expr>            ::= boolean | <logical_function> | <untyped_expr>
<sequence_expr>           ::= sequence | <sequence_function> | <untyped_expr>
<date_expr>               ::= <date_function> | <untyped_expr>
<datetime_expr>           ::= <datetime_function> | <untyped_expr>
```

Language Elements

The 'mine' Function

- `[[ % mine ]]`

  Causes the message generation to fail. Can be used e.g. to not send an automated transaction mail under certain circumstances.

String Functions

- *to_string*: {none_void_expr}
- *lower_case*: {string_expr}
- *md5*: {string_expr}
- *capitalize*: {string_expr}
- *normalize_space*: {string_expr}
- *reverse*: {string_expr}
- *trim*: {string_expr}
- *uncapitalize*: {string_expr}
- *upper_case*: {string_expr}
- *default_if_blank*: {string_expr} {string_expr}
- *default_if_empty*: {string_expr} {string_expr}
- *remove*: {string_expr} {string_expr}
- *remove_end*: {string_expr} {string_expr}

  Removes the second string from the end of the first string

- *remove_end_ignore_case*: {string_expr} {string_expr}
- *remove_start*: {string_expr} {string_expr}
- *remove_start_ignore_case*: {string_expr} {string_expr}
- *substring* <string_expr> <numeric_expr> <numeric_expr>
  - *substring 'abc' 0 2 = 'ab'*
  - *substring 'abc' 2 0 = ''*
  - *substring 'abc' 2 4 = 'c'*
  - *substring 'abc' 4 6 = ''*
  - *substring 'abc' 2 2 = ''*
  - *substring 'abc' -2 -1 = 'b'*
- *substring_after*: {string_expr} {string_expr}
- *substring_after_last*: {string_expr} {string_expr}
- *substring_before*: {string_expr} {string_expr}
- *substring_before_last*: {string_expr} {string_expr}
- *substring_between*: {string_expr} {string_expr} {string_expr}
- *random_alphabetic*: {numeric_expr}
- *random_alphanumeric*: {numeric_expr}
- *random_numeric*: {numeric_expr}
- *to_string_on_off*: {logical_expr}
- *to_string_true_false*: {logical_expr}
- *to_string_yes_no*: {logical_expr}
- *replace*: {string_expr} {string_expr} {string_expr}
- *replace_once*: {string_expr} {string_expr} {string_expr}
- *center*: {string_expr} {numeric_expr} {string_expr}
- *abbreviate*: {string_expr} {numeric_expr}

  Cuts down the string to the given length INCLUDING the three abbreviation indicating dots.
  If nothing will be cut, the dots will not be added.
  [[% abbreviate '123456' 6]] will be evaluated to '123… '
  Thus, the length MUST be greater or equal to 4.

- *abbreviate_middle*: {string_expr} {string_expr} {numeric_expr}
- *repeat*: {string_expr} {numeric_expr}
- *random*: {numeric_expr} {string_expr}
- *boolean_to_string*: {logical_expr} {string_expr} {string_expr}
- *join*: {sequence_expr} {string_expr}
- *concat*: {sequence_expr}
- *unescape_html*: {string_expr}
- *date_to_string*: {date_expr} {string_expr} {string_expr}
- *datetime_to_string*: {datetime_expr} {string_expr} {string_expr}
- *format_date*: {string_expr} {string_expr} {string_expr} {string_expr}
- *format_datetime*: {string_expr} {string_expr} {string_expr} {string_expr}
- *aes_encrypt_hex*: {string_expr} {string_expr}
- *format_number*: {numeric_expr} {string_expr} {string_expr} {numeric_expr}

  [[format_number 1234.560' '.' ',' 2]] →1.234,56

Logical functions

- *to_boolean*: {none_void_expr}
- *exists*: {none_void_expr}
- *or*: {logical_expr}*
- *and*: {logical_expr}*
- *negate*: {logical_expr}
- *xor*: {logical_expr}*
- *is_all_upper_case*
- *is_boolean*: {none_void_expr}
- *is_number*: {none_void_expr}
- *is_date*: {none_void_expr}
- *is_datetime*: {none_void_expr}
- *is_string*: {none_void_expr}
- *is_time*: {none_void_expr}
- *is_sequence*: {none_void_expr}
- *contains_whitespace*: {string_expr}
- *is_all_lower_case*: {string_expr}
- *is_alpha*: {string_expr}
- *is_alphanumeric*: {string_expr}
- *is_alphanumeric_space*: {string_expr}
- *is_alpha_space*: {string_expr}
- *is_blank*: {string_expr}
- *is_empty*: {string_expr}
- *is_not_blank*: {string_expr}
- *is_not_empty*: {string_expr}
- *is_numeric*: {string_expr}
- *is_numeric_space*: {string_expr}
- *is_whitespace*: {string_expr}
- *string_to_boolean*: {string_expr}
- *contains*: {string_expr} {string_expr}
  [[% if (contains (transaction 'someAttribute') 'X') % 'Contains X' % endif]]
- *contains_ignore_case*: {string_expr} {string_expr}
  [[% if (contains_ignore_case (transaction 'someAttribute') 'X') % 'Contains X' % endif]]
- *ends_with*: {string_expr} {string_expr}
- *ends_with_ignore_case*: {string_expr} {string_expr}
- *starts_with*: {string_expr} {string_expr}
- *starts_with_ignore_case*: {string_expr} {string_expr}
- *equals*: {string_expr} {string_expr}
  Vergleicht zwei Strings auf Gleichheit
- *equals_ignore_case*: {string_expr} {string_expr}
- *is_false*: {logical_expr}
- *is_not_false*: {logical_expr}
- *is_not_true*: {logical_expr}
- *is_true*: {logical_expr}

- *negate*: {logical_expr}
- *seq_contains*: {sequence_expr} {none_void_expr}
- *seq_contains_any*: {sequence_expr} {sequence_expr}
- *seq_contains_none*: {sequence_expr} {sequence_expr}
- *seq_contains_all*: {sequence_expr} {sequence_expr}
  Checks if sequence1 contains all elements of sequence2
- *gt*: {numeric_expr} {numeric_expr}
- *ge*: {numeric_expr} {numeric_expr}
- *eq*: {numeric_expr} {numeric_expr}
  Vergleicht numerische Werte auf Gleichheit
- *ne*: {numeric_expr} {numeric_expr}
- *lt*: {numeric_expr} {numeric_expr}
- *le*: {numeric_expr} {numeric_expr}

Numeric Functions

- *to_number*: {none_void_expr}
- *add*: {numeric_expr} {numeric_expr}
- *length*: {string_expr}
- *mul*: {numeric_expr} {numeric_expr}
- *sub*: {numeric_expr} {numeric_expr}
- *div*: {numeric_expr} {numeric_expr}
- *pow*: {numeric_expr} {numeric_expr}
- *ceil*: {numeric_expr}
- *floor*: {numeric_expr}
- *round*: {numeric_expr}
- *abs*: {numeric_expr}
- *max*: {sequence_expr}
- *min*: {sequence_expr}
- *sum*: {sequence_expr}
- *seq_size*: {sequence_expr}
- *mod*: {numeric_expr} {numeric_expr}
- *datetime_to_time* <datetime_expr>
  Translates a datetime into a timestamp
- *date_to_time* <date_expr>
  Translates a date into a timestamp
- *parse_number*: {string_expr} {string_expr} {string_expr}
  [[parse_number '1.234,560' '.' ',']]

Sequence Functions

- *split*: {string_expr} {string_expr}
- *split* (contact 'listField' ';')
- *csv*: {string_expr}
- *seq_add_all*: {sequence_expr} {sequence_expr}
- *seq_add*: {sequence_expr} {none_void_expr}
  Does not change the input sequence, make sure to save the result

- *seq_replace*: {sequence_expr} {sequence_expr}

Object Functions

- *item* {sequence_expr} {numeric_expr}
  [[% item $someSequence 0]]
  The index starts with 0

Date Functions

- *now*
- *to_date*: {string_expr} {string_expr}
- *cast_date*: {string_expr}
- *time_to_date*:{numeric_expr}

Datetime Functions

- *date_sent*
  The datetime of the sendout, also works for triggermails
- *to_datetime*: {string_expr} {string_expr}
- *cast_datetime*: {string_expr}
- *time_to_datetime*: {numeric_expr}
- *add_seconds*: {date_or_datetime_expr} {numeric_expr}
- *add_minutes*: {date_or_datetime_expr} {numeric_expr}
- *add_hours*: {date_or_datetime_expr} {numeric_expr}
- *add_days*: {date_or_datetime_expr} {numeric_expr}
- *add_weeks*: {date_or_datetime_expr} {numeric_expr}
- *add_months*: {date_or_datetime_expr} {numeric_expr}
- *add_years*: {date_or_datetime_expr} {numeric_expr}

Examples

Multiple Statements

```
[[
      % (contact 'FIRSTNAME') % ''
      %
            (
                  contact
                  'custom property'
                  'default value'
            ) % ' '
      % (transaction 'some.path') % ' '
      # comment
      % set $var (join (transaction 'some.list' ','))
      % $var
]]
```

Comments

```
[[
      # comment 1
      # comment 2
]]
```

Hashes

```
[[
      % contact 'FIRSTNAME'
      % contact 'FIRSTNAME' 'default\'value'
      % mailing 'SUBJECT'
      % account 'NAME'
      % transaction 'some.property'
]]
```

Maps

If the value of $var is a hash (key-value pairs) then you can access some value by calling

```
[[
  % (property $var 'name of the property')
]]
```

Example: Shoppingcart

```
[[
  % set $items (transaction 'cart')
  % foreach $item in $items
  %   property $item 'name'
  % endforeach
]]
```

As SUBJECT or NAME are standard values for mailings, you can also access own custom values of mailings with just specifying the key name, e.g. [[% mailing 'test']]

Standard hashes

```
[[
      % email
```

```
        % firstname
        % lastname
        % title
        % salutation
]]
```

Primitives

```
[[
        % to_string true
        % 'string'
        % -12.34
]]
```

➔ **Output**: truestring-12.34

Variables

```
[[
        % set $myVar 'a'
        % set $myVar2 (upper_case $myVar)
        % $myVar
        % $myVar2
]]
```

➔ **Output**: aA

Dates, Datetimes

```
[[
        % set $myVar
              (to_datetime
                    (transaction 'order.date')
                    'yyyy-MM-dd hh:mm:ss'
              )
        % datetime_to_string $myVar 'EEEE dd.MM.yyyy' 'fr'
]]
```

Current date and time

```
[[ % date_to_string (now) 'dd.MM.yyyy HH:mm:ss' 'de' ]]
```

Print date of yesterday

```
[[ % date_to_string (time_to_date (sub (date_to_time (now))
60000000)) 'dd.MM.yyyy' 'de' ]]
```

Sequences

```
[[
        % set $seq1 {'a', 'b', 'c'}
        % set $seq2 {'a', -1.2, true, (contact 'property'), $seq1}
        % set $seq3 (seq_add $seq1 'd')
        % set $seq3 (seq_add_all $seq1 $seq3)
        % join $seq3 'x'
]]
```

➔ **Output**: axbxcxaxbxcxd

Conditional Statements

```
[[
        % if (equals 'Max' firstname)
```

```
      %     'a'
      % elseif (equals 'Erika' firstname)
      %     'b'
      % elseif
   (or
      (and $boleanVar (contact 'bool'))
      (negate $booleanVar)
   )
      %     'c'
      % else
      %     'd'
      % endif
]]
```

Loops

```
[[
      % set $seq {{'a', 'b'}, {'c', 'd'}, {'e', 'f'}}
      % foreach $i in $seq
      %     foreach $j in $i
      %           $j
      %     endforeach
      % endforeach
      % foreach $i in {1, 2, 3}
      %     $i
      %     if (eq $i 2)
      %           break
      %     endif
      % endforeach
]]
```

➔ **Output**: abcdef12

Usecase Examples

- Calculate MD5 hash over a concatenation of the email address, a transactional field “custom2” and “custom3”

```
  [[ % md5(concat{email, (transaction 'custom2'), (transaction
'custom3')}) ]]
```

- AES encryption of contact field “custom1” with key 0123456789abcdef.
  Please note: the key length has to be exactly 128 bit (16 byte, in this case 16 characters).

```
  [[ % aes_encrypt_hex '0123456789abcdef' (contact 'custom1') ]]
```

- Add 4 Weeks to the date when the mailing has been sent, this works for regular mailings only. For transaction mailings it will be the date of activation, see “date_send”

```
  [[ % datetime_to_string (add_weeks (mailing 'DATE') 4)
'dd.MM.yyyy' 'de']]
```

- Parse String from a transaction as number, take absolute value and print it with 2 digits precission

```
[[ % format_number (abs (parse_number (transaction
'invoicePrepaidAmount') '' ',')) '' ',' 2 ]]
```

- Parse a date like "2019-11-06T02:00:00Z", and print it as "06.11.2019":

```
[[ % date_to_string (to_date (transaction 'effectiveDate') 'yyyy-
MM-dd\'T\'HH:mm:ssX') 'dd.MM.yyyy' 'de' ]]
```

- Parse a date like "2019-11-06T02:00:00.000+0200", and print it as "06.11.2019":

```
[[ % date_to_string (to_date (transaction 'effectiveDate') 'yyyy-
MM-dd\'T\'HH:mm:ss.SSSX') 'dd.MM.yyyy' 'de' ]]
```

- Check if transaction variable that is given as string is larger or smaller than 0 and print it

```
[[ % if (gt (parse_number (transaction 'amount') '.' ',') 0) % 'It
is larger' % else % 'It is smaller' % endif ]]
```

- Generate UUID v4

```
[[ % upper_case (join {(random_alphanumeric 8),
(random_alphanumeric 4), (random_alphanumeric 4),
(random_alphanumeric 4), (random_alphanumeric 8) } '-')]
```

- Print Birthday

```
[[ % date_to_string (cast_date (contact 'BIRTHDAY')) 'dd.MM.yyyy'
'en']]
```

- Add 3 months to birthday:

```
[[ % datetime_to_string (add_months (cast_date (contact 'BIRTHDAY'))
3) 'dd-MM-yyyy' 'nl']]
```

- Add 3 months to sendout date:

```
[[% datetime_to_string (add_months date_sent 3) 'dd.MM.yyyy
HH:mm:ss' 'de']]
```

Unescaping

`[[ % unescape_html '&#8226;' ]]` Ein Bullet
`[[ % unescape_html '&bull;' ]]` Ein Bullet

Displaying Magento Price Reductions

```
[[
    % if ( is_not_empty '{{MAGENTO-PRODUCT|SPECIAL-PRICE}}' )
        % '{{MAGENTO-PRODUCT|SPECIAL-PRICE|0.00de_DE}} CHF
{{MAGENTO-PRODUCT|PRICE|0.00de_DE}} CHF'
    % else
        % '{{MAGENTO-PRODUCT|PRICE|0.00de_DE}} CHF'
    % endif
]]
```

Displaying Shopware Price Reductions

```
[[
   % if(or(is_empty '{{SHOPWARE-ITEM|PSEUDOPRICE|0.00de_DE}}')
(equals '{{SHOPWARE-ITEM|PSEUDOPRICE|0.00de_DE}}' '0,00'))
      % '{{SHOPWARE-ITEM|PRICE|0.00de_DE}} €'
   % else
     % '{{SHOPWARE-ITEM|PRICE|0.00de_DE}} € statt  {{SHOPWARE-
ITEM|PSEUDOPRICE|0.00de_DE}} €'
   % endif
]]
```

Changelog

Version 1.5 (TBD)

- Added examples
- Adde new methods (sendout_date)

Version 1.4 (27.09.2019)

- New layout for documentation

Version 1.3 (07.02.2019)

- added parse_number: {string_expr} {string_expr} {string_expr}
- added format_number: {numeric_expr} {string_expr} {string_expr} {numeric_expr}
- added examples

Version 1.2 (28.03.2017)

- added substring <string_expr> <numeric_expr> <numeric_expr>

Version 1.1 (29.07.2016)

- added datetime_to_time <datetime_expr>
- added date_to_time <date_expr>
- added time_to_datetime <numeric_expr>
- added time_to_date <numeric_expr>
- added add_seconds <date_or_datetime_expr> <numeric_expr>
- added add_minutes <date_or_datetime_expr> <numeric_expr>
- added add_hours <date_or_datetime_expr> <numeric_expr>
- added add_days <date_or_datetime_expr> <numeric_expr>
- added add_weeks <date_or_datetime_expr> <numeric_expr>
- added add_months <date_or_datetime_expr> <numeric_expr>
- added add_years <date_or_datetime_expr> <numeric_expr>

Unser Serviceteam berät Sie gerne:

069 83 00 898 - 0

XQueue GmbH
Christian-Pless-Str. 11-13
63069 Offenbach am Main

Tel: +49 (0) 69 8300 898 – 0
Fax: +49 (0) 69 8300 898 – 9
Mail: service@xqueue.de
Web: www.xqueue.de

XQUEUE
Advanced E-Mail-Marketing Technologies

MAILEON
EMAIL MARKETING