

Utils

Method Summary

String [#dateFormat](#)(date, format)
Format a date object to a text representation.

String [#getUnicodeCharacter](#)(unicodeCharacterNumber)
Returns a string containing the character for the unicode number.

Boolean [#hasRecords](#)(foundset)
Returns true if the (related)foundset exists and has records.

Boolean [#hasRecords](#)(record, relationString)
Returns true if the (related)foundset exists and has records.

Boolean [#isMondayFirstDayOfWeek](#)()
Returns true when Monday is the first day of the week for your current locale setting.

String [#numberFormat](#)(number, digits)
Format a number to have a defined fraction.

String [#numberFormat](#)(number, format)
Format a number to specification.

Date [#parseDate](#)(date, format)
Parse a string to a date object.

String [#stringEscapeMarkup](#)(textString)
Returns the escaped markup text (HTML/XML).

String [#stringEscapeMarkup](#)(textString, escapeSpaces)
Returns the escaped markup text (HTML/XML).

String [#stringEscapeMarkup](#)(textString, escapeSpaces, convertToHtmlUnicodeEscapes)
Returns the escaped markup text (HTML/XML).

String [#stringFormat](#)(text_to_format, parameters)
Formats a string according to format specifiers and arguments.

String [#stringIndexReplace](#)(text, i_start, i_size, replacement_text)
Replaces a portion of a string with replacement text from a specified index.

String [#stringInitCap](#)(text)
Returns all words starting with capital chars.

String [#stringLeft](#)(text, i_size)
Returns a string with the requested number of characters, starting from the left.

String [#stringLeftWords](#)(text, numberof_words)
Returns the number of words, starting from the left.

String [#stringMD5HashBase16](#)(textString)
Returns the md5 hash (encoded as base16) for specified text.

String [#stringMD5HashBase64](#)(textString)
Returns the md5 hash (encoded as base64) for specified text.

String [#stringMiddle](#)(text, i_start, i_size)
Returns a substring from the original string.

String [#stringMiddleWords](#)(text, i_start, numberof_words)
Returns a substring from the original string.

String [#stringPBKDF2Hash](#)(textString)
Returns the PBKDF2 hash for specified text.

String [#stringPBKDF2Hash](#)(textString, iterations)
Returns the PBKDF2 hash for specified text.

Number [#stringPatternCount](#)(text, toSearchFor)
Returns the number of times searchString appears in textString.

Number [#stringPosition](#)(textString, toSearchFor, i_start, i_occurrence)
Returns the position of the string to search for, from a certain start position and occurrence.

String [#stringReplace](#)(text, search_text, replacement_text)
Replaces a portion of a string with replacement text.

String [#stringReplaceTags](#)(text, scriptable)
Returns the text with %tags%% replaced, based on provided record or foundset or form.

String [#stringRight](#)(text, i_size)
Returns a string with the requested number of characters, starting from the right.

String [#stringRightWords](#)(text, numberof_words)
Returns the number of words, starting from the right.

Number [#stringToNumber](#)(textString)
Filters characters out of from a string and leaves digits, returns the number.

String [#stringTrim](#)(textString)
Returns the string without leading or trailing spaces.

Number [#stringWordCount](#)(text)
Returns the number of words in the text string.

Date [#timestampToDate](#)(date)
Returns a datestamp from the timestamp (sets hours,minutes,seconds and milliseconds to 0).

Boolean [#validatePBKDF2Hash](#)(password, hash)
Validates the given password against the given hash.

Method Details

dateFormat

String dateFormat(date, format)

Format a date object to a text representation.

Parameters

{Date} date – the date

{String} format – the format to output

Returns

String – the date as text

Sample

```
var formattedDateString = utils.dateFormat(dateobject, 'EEE, d MMM yyyy HH:mm:ss');
```

getUnicodeCharacter

String getUnicodeCharacter(unicodeCharacterNumber)

Returns a string containing the character for the unicode number.

Parameters

{Number} unicodeCharacterNumber – the number indicating the unicode character

Returns

String – a string containing the unicode character

Sample

```
//returns a big dot
var dot = utils.getUnicodeCharacter(9679);
```

hasRecords

Boolean hasRecords(foundset)

Returns true if the (related)foundset exists and has records.

Another use is, to pass a record and qualified relations string to test multiple relations/foundset at once

Parameters

{JSFoundSet} foundset – the foundset to be tested

Returns

Boolean – true if exists

Sample

```
//test the orders_to_orderitems foundset
if (elements.customer_id.hasRecords(orders_to_orderitems))
{
    //do work on relatedFoundSet
}
//test the orders_to_orderitems.orderitems_to_products foundset to be reached from the current record
//if (elements.customer_id.hasRecords(foundset.getSelectedRecord(), 'orders_to_orderitems.
orderitems_to_products'))
//{
//    //do work on deeper relatedFoundSet
//}
```

hasRecords

Boolean hasRecords(record, relationString)

Returns true if the (related)foundset exists and has records.

Another use is, to pass a record and qualified relations string to test multiple relations/foundset at once

Parameters

{JSRecord} record – A JSRecord to test.

{String} relationString – The relation name.

Returns

Boolean – true if the foundset/relation has records.

Sample

```
//test the orders_to_orderitems foundset
if (elements.customer_id.hasRecords(orders_to_orderitems))
{
    //do work on relatedFoundSet
}
//test the orders_to_orderitems.orderitems_to_products foundset to be reached from the current record
//if (elements.customer_id.hasRecords(foundset.getSelectedRecord(),'orders_to_orderitems.
orderitems_to_products'))
//{
//    //do work on deeper relatedFoundSet
//}
```

isMondayFirstDayOfWeek

Boolean **isMondayFirstDayOfWeek()**

Returns true when Monday is the first day of the week for your current locale setting.

Returns

Boolean – true if Monday is first day of the week in current locale

Sample

```
if(utils.isMondayFirstDayOfWeek())
{
    //a date calculation
}
```

numberFormat

String **numberFormat**(number, digits)

Format a number to have a defined fraction.

Parameters

{**Number**} number – the number to format

{**Number**} digits – nr of digits

Returns

String – the resulting number in text

Sample

```
var textualNumber = utils.numberFormat(16.749, 2); //returns 16.75
```

numberFormat

String **numberFormat**(number, format)

Format a number to specification.

Parameters

{**Number**} number – the number to format

{**String**} format – the format

Returns

String – the resulting number in text

Sample

```
var textualNumber2 = utils.numberFormat(100006.749, '#,###.00'); //returns 100,006.75
```

parseDate

Date **parseDate**(date, format)

Parse a string to a date object.

Parameters

{**String**} date – the date as text

{**String**} format – the format to parse the to date

Returns

Date – the date as date object

Sample

```
var parsedDate = utils.parseDate(datestring,'EEE, d MMM yyyy HH:mm:ss');
```

stringEscapeMarkup

String **stringEscapeMarkup**(textString)

Returns the escaped markup text (HTML/XML).

Parameters

{**String**} textString – the text to process

Returns

String – the escaped text

Sample

```
var escapedText = utils.stringEscapeMarkup('<html><body>escape me</body></html>')
```

stringEscapeMarkup

String **stringEscapeMarkup**(textString, escapeSpaces)

Returns the escaped markup text (HTML/XML).

Parameters

{**String**} textString – the text to process

{**Boolean**} escapeSpaces – indicating to escape spaces

Returns

String – the escaped text

Sample

```
var escapedText = utils.stringEscapeMarkup('<html><body>escape me</body></html>')
```

stringEscapeMarkup

String **stringEscapeMarkup**(textString, escapeSpaces, convertToHtmlUnicodeEscapes)

Returns the escaped markup text (HTML/XML).

Parameters

{**String**} textString – the text to process

{**Boolean**} escapeSpaces – indicating to escape spaces

{**Boolean**} convertToHtmlUnicodeEscapes – indicating to use unicode escapes

Returns

String – the escaped text

Sample

```
var escapedText = utils.stringEscapeMarkup('<html><body>escape me</body></html>')
```

stringFormat

String **stringFormat**(text_to_format, parameters)

Formats a string according to format specifiers and arguments.

Parameters

{**String**} text_to_format – the text to format

{**Object**[]} parameters – the array with parameters

Returns

String – the formatted text

Sample

```
// the format specifier has the syntax: %[argument_index$][flags][width][.precision]conversion
// argument index is 1$, 2$ ...
// flags is a set of characters that modify the output format
// typical values: '+'(The result will always include a sign), ','(The result will include locale-specific
grouping separators)
// width is a non-negative decimal integer indicating the minimum number of characters to be written to the
output
// precision is a non-negative decimal integer usually used to restrict the number of characters
// conversion is a character indicating how the argument should be formatted
// typical conversion values: b(boolean), s(string), c(character), d(decimal integer), f(floating number), t
(prefix for date and time)
// Date/Time Conversions (used after 't' prefix):
// 'H'      Hour of the day for the 24-hour clock, formatted as two digits with a leading
zero as necessary i.e. 00 - 23.
// 'I'      Hour for the 12-hour clock, formatted as two digits with a leading zero as
necessary, i.e. 01 - 12.
// 'k'      Hour of the day for the 24-hour clock, i.e. 0 - 23.
// 'l'      Hour for the 12-hour clock, i.e. 1 - 12.
// 'M'      Minute within the hour formatted as two digits with a leading zero as necessary,
i.e. 00 - 59.
// 'S'      Seconds within the minute, formatted as two digits with a leading zero as
necessary, i.e. 00 - 60 ("60" is a special value required to support leap seconds).
// 'L'      Millisecond within the second formatted as three digits with leading zeros as
necessary, i.e. 000 - 999.
// 'p'      Locale-specific morning or afternoon marker in lower case, e.g. "am" or "pm". Use
of the conversion prefix 'T' forces this output to upper case.
// 'z'      RFC 822 style numeric time zone offset from GMT, e.g. -0800.
// 'Z'      A string representing the abbreviation for the time zone.
// 'B'      Locale-specific full month name, e.g. "January", "February".
// 'b'      Locale-specific abbreviated month name, e.g. "Jan", "Feb".
// 'h'      Same as 'b'.
// 'A'      Locale-specific full name of the day of the week, e.g. "Sunday", "Monday"
// 'a'      Locale-specific short name of the day of the week, e.g. "Sun", "Mon"
// 'C'      Four-digit year divided by 100, formatted as two digits with leading zero as
necessary, i.e. 00 - 99
// 'Y'      Year, formatted as at least four digits with leading zeros as necessary, e.g.
0092 equals 92 CE for the Gregorian calendar.
// 'y'      Last two digits of the year, formatted with leading zeros as necessary, i.e. 00
- 99.
// 'j'      Day of year, formatted as three digits with leading zeros as necessary, e.g. 001
- 366 for the Gregorian calendar.
// 'm'      Month, formatted as two digits with leading zeros as necessary, i.e. 01 - 13.
// 'd'      Day of month, formatted as two digits with leading zeros as necessary, i.e. 01 -
31
// 'e'      Day of month, formatted as two digits, i.e. 1 - 31.

// common compositions for date/time conversion
// 'R'      Time formatted for the 24-hour clock as "%tH:%tM"
// 'T'      Time formatted for the 24-hour clock as "%tH:%tM:%tS".
// 'r'      Time formatted for the 12-hour clock as "%tI:%tM:%tS %Tp". The location of the
morning or afternoon marker ('%Tp') may be locale-dependent.
// 'D'      Date formatted as "%tm/%td/%ty".
// 'F'      ISO 8601 complete date formatted as "%tY-%tm-%td".
// 'c'      Date and time formatted as "%ta %tb %td %tT %tZ %tY", e.g. "Sun Jul 20 16:17:00
EDT 1969".

utils.stringFormat('%s Birthday: %2$tm %2$te,%2$ty',new Array('My',new Date(2009,0,1))) // returns My Birthday:
01 1,2009
utils.stringFormat('The time is: %1$tH:%1$tM:%1$tS',new Array(new Date(2009,0,1,12,0,0))) // returns The time
is: 12:00:00
utils.stringFormat('My %s: %2$.0f, my float: %2$.2f',new Array('integer',10)) // returns My integer: 10, my
float: 10.00
utils.stringFormat('Today is: %1$tc',new Array(new Date())) // returns current date/time as: Today is: Fri Feb
20 14:15:54 EET 2009
utils.stringFormat('Today is: %tF',new Array(new Date())) // returns current date as: Today is: 2009-02-20
```

String **stringIndexReplace**(text, i_start, i_size, replacement_text)

Replaces a portion of a string with replacement text from a specified index.

Parameters

{String} text – the text to process

{Number} i_start – the start index to work from

{Number} i_size – the size of the text to replace

{String} replacement_text – the replacement text

Returns

String – the changed text string

Sample

```
//returns 'this was a test'
var retval = utils.stringIndexReplace('this is a test',6,2,'was');
```

stringInitCap

String **stringInitCap**(text)

Returns all words starting with capital chars.

Parameters

{String} text – the text to process

Returns

String – the changed text

Sample

```
//returns 'This Is A Test'
var retval = utils.stringInitCap('This is A test');
```

stringLeft

String **stringLeft**(text, i_size)

Returns a string with the requested number of characters, starting from the left.

Parameters

{String} text – the text to process

{Number} i_size – the size of the text to return

Returns

String – the result text string

Sample

```
//returns 'this i'
var retval = utils.stringLeft('this is a test',6);
```

stringLeftWords

String **stringLeftWords**(text, numberof_words)

Returns the number of words, starting from the left.

Parameters

{String} text – to process

{Number} numberof_words – to return

Returns

String – the string with number of words form the left

Sample

```
//returns 'this is a'
var retval = utils.stringLeftWords('this is a test',3);
```

stringMD5HashBase16

String **stringMD5HashBase16**(textString)

Returns the md5 hash (encoded as base16) for specified text.

NOTE: MD5 (Message-Digest Algorithm 5) is a hash function with a 128-bit hash value, for more info see: <http://en.wikipedia.org/wiki/MD5>

Parameters

{String} textString – the text to process

Returns

String – the resulting hashString

Sample

```
var hashed_password = utils.stringMD5HashBase16(user_password)
```

stringMD5HashBase64

String **stringMD5HashBase64**(textString)

Returns the md5 hash (encoded as base64) for specified text.

NOTE: MD5 (Message-Digest Algorithm 5) is a hash function with a 128-bit hash value, for more info see: <http://en.wikipedia.org/wiki/MD5>

Parameters

{String} textString – the text to process

Returns

String – the resulting hashString

Sample

```
var hashed_password = utils.stringMD5HashBase64(user_password)
```

stringMiddle

String **stringMiddle**(text, i_start, i_size)

Returns a substring from the original string.

Parameters

{String} text – the text to process

{Number} i_start – the start index to work from

{Number} i_size – the size of the text to return

Returns

String – the result text string

Sample

```
//returns 'his'
var retval = utils.stringMiddle('this is a test',2,3);
```

stringMiddleWords

String **stringMiddleWords**(text, i_start, numberof_words)

Returns a substring from the original string.

Parameters

{String} text – to process

{Number} i_start – start word index

{Number} numberof_words – the word count to return

Returns

String – the string with number of words from the left and

Sample

```
//returns 'is a'
var retval = utils.stringMiddleWords('this is a test',2,2);
```

stringPBKDF2Hash

String **stringPBKDF2Hash**(textString)

Returns the PBKDF2 hash for specified text. This method is preferred above the old MD5 hash for enhanced security. It uses a default of 2000 iterations.

NOTE: PBKDF2 is the key hash function for the PKCS (Public-Key Cryptography) standard, for more info see: <http://en.wikipedia.org/wiki/PBKDF2>

Parameters

{String} textString – the text to process

Returns

String – the resulting hashString

Sample

```
var hashed_password = utils.stringPBKDF2Hash(user_password)
```

stringPBKDF2Hash

String **stringPBKDF2Hash**(textString, iterations)

Returns the PBKDF2 hash for specified text. This method is preferred above the old MD5 hash for enhanced security.

NOTE: PBKDF2 is the key hash function for the PKCS (Public-Key Cryptography) standard, for more info see: <http://en.wikipedia.org/wiki/PBKDF2>

Parameters

{String} textString – the text to process

{Number} iterations – how many hash iterations should be done, minimum should be 1000 or higher.

Returns

String – the resulting hashString

Sample

```
var hashed_password = utils.stringPBKDF2Hash(user_password,5000)
```

stringPatternCount

Number **stringPatternCount**(text, toSearchFor)

Returns the number of times searchString appears in textString.

Parameters

{String} text – the text to process

{String} toSearchFor – the string to search for

Returns

Number – the occurrenceCount that the search string is found in the text

Sample

```
//returns 2 as count
var count = utils.stringPatternCount('this is a test','is');
```

stringPosition

Number **stringPosition**(textString, toSearchFor, i_start, i_occurrence)

Returns the position of the string to search for, from a certain start position and occurrence.

Parameters

{String} textString – the text to process

{String} toSearchFor – the string to search

{Number} i_start – the start index to search from

{Number} i_occurrence – the occurrence

Returns

Number – the position

Sample

```
//returns 4 as position
var pos = utils.stringPosition('This is a test','s',1,1)
```

stringReplace

String **stringReplace**(text, search_text, replacement_text)

Replaces a portion of a string with replacement text.

Parameters

{String} text – the text to process

{String} search_text – the string to search

{String} replacement_text – the replacement text

Returns

String – the changed text string

Sample

```
//returns 'these are cow 1 and cow 2.'
var retval = utils.stringReplace('these are test 1 and test 2.','test','cow');
```

stringReplaceTags

String **stringReplaceTags**(text, scriptable)

Returns the text with %tags% replaced, based on provided record or foundset or form.

Parameters

{String} text – the text tags to work with

{Object} scriptable – the javascript object or foundset,record,form to be used to fill in the tags

Returns

String – the text with replaced tags

Sample

```
//Next line places a string in variable x, whereby the tag(%TAG%) is filled with the value of the database
column 'company_name' of the selected record.
var x = utils.stringReplaceTags("The companyName of the selected record is %%company_name%% ", foundset)
//var otherExample = utils.stringReplaceTags("The amount of the related order line %%amount%% ",
order_to_orderdetails);
//var recordExample = utils.stringReplaceTags("The amount of the related order line %%amount%% ",
order_to_orderdetails.getRecord(i);
//Next line places a string in variable y, whereby the tag(%TAG%) is filled with the value of the form
variable 'x' of the form named 'main'.
//var y = utils.stringReplaceTags("The value of form variable is %%x%% ", forms.main);
//The next sample shows the use of a javascript object
//var obj = new Object();//create a javascript object
//obj['x'] = 'test';//assign an named value
//var y = utils.stringReplaceTags("The value of object variable is %%x%% ", obj);//use the named value in a tag
```

stringRight

String **stringRight**(text, i_size)

Returns a string with the requested number of characters, starting from the right.

Parameters

{**String**} text – the text to process

{**Number**} i_size – the size of the text to return

Returns

String – the result text string

Sample

```
//returns 'a test'
var retval = utils.stringLeft('this is a test',6);
```

stringRightWords

String **stringRightWords**(text, numberof_words)

Returns the number of words, starting from the right.

Parameters

{**String**} text – to process

{**Number**} numberof_words – to return

Returns

String – the string with number of words form the right

Sample

```
//returns 'is a test'
var retval = utils.stringRightWords('this is a test',3);
```

stringToNumber

Number **stringToNumber**(textString)

Filters characters out of from a string and leaves digits, returns the number. Uses locale decimal separator.

Parameters

{**String**} textString – the text to process

Returns

Number – the resulting number

Sample

```
//returns '65567'
var retval = utils.stringToNumber('fg65gf567');
```

stringTrim

String **stringTrim**(textString)

Returns the string without leading or trailing spaces.

Parameters

{**String**} textString – the text to process

Returns

String – the resulting trimmed string

Sample

```
//returns 'text'
var retval = utils.stringTrim('  text  ');
```

stringWordCount

Number **stringWordCount**(text)

Returns the number of words in the text string.

Parameters

{**String**} text – the text to process

Returns

Number – the word count

Sample

```
//returns '4' as result
var retval = utils.stringWordCount('this is a test');
```

timestampToDate

Date **timestampToDate**(date)

Returns a datestamp from the timestamp (sets hours, minutes, seconds and milliseconds to 0).

Parameters

{**Date**} date – object to be stripped from its time elements

Returns

Date – the stripped date object

Sample

```
var date = utils.timestampToDate(application.getTimestamp());
```

validatePBKDF2Hash

Boolean **validatePBKDF2Hash**(password, hash)

Validates the given password against the given hash. The hash should be generated by one of the stringPBKDF2Hash(password [,iteration]) functions.

NOTE: PBKDF2 is the key hash function for the PKCS (Public-Key Cryptography) standard, for more info see: <http://en.wikipedia.org/wiki/PBKDF2>

Parameters

{**String**} password – the password to test against

{**String**} hash – the hash the password needs to validate to.

Returns

Boolean – true if his hash is valid for that password

Sample

```
if (utils.validatePBKDF2Hash(user_password, hashFromDb)) {
  // logged in
}
```