

8 Inline function reference

Inline message blocks are placed into messages using curly braces as in the following example:

```
"The birth date you entered is {value}, you are too young  
to register here."
```

Inline message blocks can contain any valid combination of properties and functions. Inline message functions follow the format:

```
functionName([param 1[, param 2[, ... param n]])
```

The result of the function is embedded into the message at the location of the inline message block.

The following reference describes all functions available within the inline message blocks. For a complete explanation of the use of inline message functions, see [Inline message blocks](#) earlier in this document.

```
cdate(value[, format])
```

Converts the value into a date if possible using the format if supplied. If the value cannot be converted to a date, the original value is returned.

Return value: string

Parameters:

| | |
|--------|---|
| value | A value capable of being parsed as a date. |
| format | An optional format specifier. If this is supplied it must be one of the values, 'DDMMYYYY', 'MMDDYYYY', 'YYYYMMDD', 'YYYYDDMM'. These values indicate which date format should be adopted if the format of the date being passed into the function seems ambiguous. If the format selected is 'MMDDYYYY' then the first digit(s) in the date represent the month value, the second the day value and the third the year value. The format specifier has no affect if the date being passed in as the value is not ambiguous. An example of an ambiguous date is 03/05/04. |

Example:

```
cdate(value, 'DDMMYYYY')
```

```
cdbl(value)
```

Converts the value into a double (floating point number) if it is not already one. If the value cannot be converted, the original value is returned.

Return value: double

Parameters:

| | |
|-------|---|
| value | Any value capable of being parsed as a double.. |
|-------|---|

```
cint(value)
```

Converts the value into an integer if it is not already one. If the value cannot be converted, the original value is returned.

Return value: integer

Parameters:

| | |
|-------|--|
| value | Any value capable of being parsed as an integer. |
|-------|--|

concat

1.3

```
concat(value1[, value2[, ... value n]])
```

Returns the string concatenation of all values passed in.

Return value: string

Parameters:

`value1-n` Any value returned from a function or a property.

cstr

1.3

```
cstr(value)
```

Converts the value into a string if it is not already one. If the value cannot be converted, the original value is returned.

Return value: string

Parameters:

`value` Any value returned from a function or a property.

formatDate

1.3

```
formatDate(date, format[, default])
```

Returns a string representation of the date as specified by the format string. The format string follows the same rules as [Microsoft's custom date time string formatting](#). If the date value passed in cannot be formatted, it is returned as is unless *default* has been provided, in which case the default value is returned instead. Note that the value passed in as the date must be a Date type (DateTime on the server).

Unlike Microsoft's format specifiers, single character format strings do not represent fixed system specific format specifiers on the browser. However, single character format strings are treated this way on the server, so you should avoid using these strings altogether.

This function is fully culture sensitive and uses the culture specified in the Validator control or the web.config file (see [Setting up the web.config file](#)). If the culture cannot be determined from these methods, the culture of the website is used.

The following table describes the format specifiers and the output they produce.

| Format Specifier | Description |
|------------------|---|
| D | The day of the month represented by the numbers 1 through 31. |

| Format Specifier | Description |
|-------------------------|--|
| Dd | The day of the month preceded, if the day is a single digit, by a 0. |
| Ddd | The abbreviated name of the day of the week. |
| Dddd | The full name of the day of the week. |
| M | The numeric month represented by the numbers 1 through 12. |
| MM | The numeric month preceded, if the month value is a single digit, by a 0. |
| MMM | The abbreviated name of the month. |
| MMMM | The full name of the month. |
| Y | The year in the century represented by the numbers 0 through 99. |
| Yy | The year in the century preceded, if the result is a single digit, by a 0. |
| Yyyy | The 4 digit year. |
| H | The hour in the 12 hour clock represented by the numbers 1 through 12. |
| Hh | The hour in the 12 hour clock preceded, if the hour is a single digit, by a 0. |
| H | The hour in the 24 hour clock represented by the numbers 0 through 23. |
| HH | The hour in the 24 hour clock preceded, if the hour is a single digit, by a 0. |
| M | The minute represented by the numbers 0 through 59. |
| Mm | The minute preceded, if the minute is a single digit, by a 0. |
| S | The second represented by the numbers 0 through 59. |
| Ss | The second preceded, if the second is a single digit, by a 0. |
| F | The seconds fractions returned as a single decimal place. |
| Ff | The seconds fractions returned as 2 decimal places. |
| Fff | The seconds fractions returned as 3 decimal places. |
| Ffff | The seconds fractions returned as 4 decimal places. |
| Fffff | The seconds fractions returned as 5 decimal places. |
| Ffffff | The seconds fractions returned as 6 decimal places. |
| Fffffff | The seconds fractions returned as 7 decimal places. |
| T | The first character of the AM/PM designator. |
| Tt | The full AM/PM designator. |
| Z | The time zone offset from UTC of the server in whole hours preceded by a '+' or a '-' character depending on whether the offset is positive or negative. |
| Zz | The time zone offset from UTC of the server in whole hours preceded by a '+' or a '-' character depending on whether the offset is positive or negative. If the offset is a single digit it is preceded with a 0. |
| Zzz | The time zone offset from UTC of the server in whole hours and minutes preceded by a '+' or a '-' character depending on whether the offset is positive or negative, e.g. +10:30. The separator used is culture specific and is determined by the time separator of the culture. |
| : | The culture specific time separator. |

| Format Specifier | Description |
|------------------|--------------------------------------|
| / | The culture specific date separator. |

Return value: string

Parameters:

date A date object.
format A format string. This can be any combination of the format specifiers in the table above.

Example:

```
formatDate(cdate(value, 'DDMMYYYY'), 'd MMM yyyy HH:mm:ss',
'Invalid date entered')
```

formatNumber

1.3

```
formatNumber(value, format[, default])
```

Returns a string representation of the number as specified by the format string. The format string follows the same rules as [Microsoft's custom number string formatting](#). If the value passed in cannot be formatted, it is returned as is unless *default* has been provided, in which case the default value is returned instead.

This function is fully culture sensitive and uses the culture specified in the Validator control or the web.config file (see [Setting up the web.config file](#)). If the culture cannot be determined from these methods, the culture of the website is used.

The following table describes the format specifiers and the output they produce.

| Format Specifier | Meaning | Description |
|------------------|------------------------|--|
| 0 | Digit/zero placeholder | If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the result string. The position of the leftmost '0' before the decimal point and the rightmost '0' after the decimal point determines the range of digits that are always present in the result string. The '00' specifier causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with '00' would result in the value 35. |
| # | Digit placeholder | If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the result string. Otherwise, nothing is stored in that position in the result string. |

| Format Specifier | Meaning | Description |
|------------------|--|--|
| | | <p>Note that this specifier never displays the '0' character if it is not a significant digit, even if '0' is the only digit in the string. It will display the '0' character if it is a significant digit in the number being displayed. The '###' format string causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with '###' would result in the value 35.</p> |
| . | Decimal point placeholder | <p>The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as the decimal point separator is culture specific and is determined by the <code>NumberDecimalSeparator</code> property of the <code>NumberFormatInfo</code> being used on the server.</p> |
| , | Thousands separator and number scaling | <p>The ',' character serves two purposes. First, if the format string contains a ',' character between two digit placeholders (0 or #) and to the left of the decimal point if one is present, then the output will have thousand separators inserted between each group of three digits to the left of the decimal separator. The actual character used as the thousands separator is culture specific and is determined by the <code>NumberGroupSeparator</code> property of the <code>NumberFormatInfo</code> being used on the server.</p> <p>Second, if the format string contains one or more ',' characters immediately to the left of the decimal point, then the number will be divided by the number of ',' characters multiplied by 1000 before it is formatted. For example, the format string "0,," will represent 100 million as simply 100. Use of the ',' character to indicate scaling does not include thousand separators in the formatted number. Thus, to scale a number by 1 million and insert thousand separators you would use the format string '#,##0,,'.</p> |

| Format Specifier | Meaning | Description |
|--------------------------------------|------------------------|---|
| % | Percentage placeholder | The presence of a '%' character in a format string causes a number to be multiplied by 100 before it is formatted. The appropriate symbol is inserted in the number itself at the location where the '%' appears in the format string. |
| E0 E+0 E-0 e0 e+0 e-0 | Scientific notation | If any of the strings 'E', 'E+', 'E-', 'e', 'e+', or 'e-' are present in the format string and are followed immediately by at least one '0' character, then the number is formatted using scientific notation with an 'E' or 'e' inserted between the number and the exponent. The number of '0' characters following the scientific notation indicator determines the minimum number of digits to output for the exponent. The 'E+' and 'e+' formats indicate that a sign character (plus or minus) should always precede the exponent. The 'E', 'E-', 'e', or 'e-' formats indicate that a sign character should only precede negative exponents. |
| \ | Escape character | Escapes the character following the backslash as a literal character. |
| ; | Section delimiter | The ';' character is used to separate sections for positive, negative, and zero numbers in the format string. |
| All other characters | | All other characters are copied to the result string as literal characters in the positions they appear. |

Note that for fixed-point format strings (strings not containing an 'E', 'E+', 'E-', 'e', 'e+', or 'e-'), numbers are rounded to as many decimal places as there are digit placeholders to the right of the decimal point. If the format string does not contain a decimal point, the number is rounded to the nearest integer. If the number has more digits than there are digit placeholders to the left of the decimal point, the extra digits are copied to the result string immediately before the first digit placeholder.

Different formatting can be applied to a string based on whether the value is positive, negative, or zero. To produce this behavior, a custom format string can contain up to three sections separated by semicolons:

One section:

The format string applies to all values.

Two sections:

The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections:

The first section applies to positive values, the second section applies to negative values, and the third section applies to zeros. The second section might be left empty (by having nothing between the semicolons), in which case the first section applies to all nonzero values.

This type of formatting ignores any pre-existing formatting associated with a number when the final value is formatted. For example, negative values are always displayed without a minus sign when section separators are used. If you want the final formatted value to have a minus sign, you should explicitly include the minus sign as part of the custom format specifier. The following example illustrates how section separators can be used to produce formatted strings.

Return value: string

Parameters:

| | |
|--------|--|
| value | A numeric value. |
| format | A format string as described in the table above. |

Example:

```
formatNumber(value, '$#,##0.00;($#,##0.00);Zero', 'Invalid number')
```

isbool

1.3

isbool(value)

Returns true if the value passed in is a boolean value. If the value is a string, then it is treated as a boolean value if it equals 'true' or 'false' in any case.

Return value: boolean

Parameters:

| | |
|-------|------------------------|
| value | Any value of any type. |
|-------|------------------------|

isdbl

1.3

isdbl(value)

Returns true if the value passed in is a number.

Return value: boolean

Parameters:

| | |
|-------|------------------------|
| value | Any value of any type. |
|-------|------------------------|

```
isint(value)
```

Returns true if the value passed in is an integer. If the value is a number with a value right of the decimal place, it is not treated as an integer and this function therefore will return false.

Return value: boolean

Parameters:

value Any value of any type.

```
isdate(value[, format])
```

Returns true if the value can be converted to a date using the same conditions as [cdate](#).

Return value: boolean

Parameters:

value A value capable of being parsed as a date.
format An optional format specifier. If this is supplied it must be one of the values, 'DDMMYYYY', 'MMDDYYYY', 'YYYYMMDD', 'YYYYDDMM'. These values indicate which date format should be adopted if the format of the date being passed into the function seems ambiguous. If the format selected is 'MMDDYYYY' then the first digit(s) in the date represent the month value, the second the day value and the third the year value. The format specifier has no affect if the date being passed in as the value is not ambiguous. An example of an ambiguous date is 03/05/04.

```
join(array, delimiter1[, delimiter2[, template]])
```

Joins the keys and/or values of an associative array using one or more delimiters and a template. If `delimiter2` and `template` are not supplied, the associative array's values are concatenated together using `delimiter1` as the delimiter. If `delimiter2` is supplied, it is used as the final delimiter, e.g.

```
join(allvalues, ', ', ' and ')
```

This might produce the result “value1, value2, value3 and value4.”

If the template is supplied, it must contain ‘{value}’ and/or ‘{key}’ or any combination of these. It can additionally contain any combination of characters outside the curly braces. For each pair of elements in the associative array, the template is evaluated and ‘{value}’ and ‘{key}’ replaced with the respective value and key from the array.

Return value: string

Parameters:

| | |
|-------------------------|---|
| <code>array</code> | An associative array. |
| <code>delimiter1</code> | The primary delimiter. |
| <code>delimiter2</code> | The final delimiter. This is used only once at the end of the join. |
| <code>template</code> | A string containing a combination of ‘{value}’ and/or ‘{key}’. |

Example:

```
join(allvalues, ', ', ' and ', '{value} ( {key} )')
```

```
lc(str)
```

Returns a lowercase copy of the string passed in.

Return value: string

Parameters:

| | |
|------------------|-----------------|
| <code>str</code> | A string value. |
|------------------|-----------------|

length

1.3

`length(value)`

Returns the length of the string or array passed in.

Return value: integer

Parameters:

| | |
|--------------------|---|
| <code>value</code> | If the value is an array, the length of the array (the number of elements) is returned. If the value is a string, the length of the string is returned. |
|--------------------|---|

slice

1.3

`slice(array, beginIndex[, endIndex])`

Returns a new associative array using the elements the source array starting from `beginIndex` and ending at `endIndex` or the end of the source array if no `endIndex` is provided.

Return value: array

Parameters:

| | |
|-------------------------|---|
| <code>array</code> | The source array. |
| <code>beginIndex</code> | The index position to begin the slice at. |
| <code>endIndex</code> | The index position to end the slice at. |

uc

1.3

`uc(str)`

Returns an uppercase copy of the string passed in.

Return value: string

Parameters:

| | |
|------------------|-----------------|
| <code>str</code> | A string value. |
|------------------|-----------------|