

Contents

Omnis Function Reference

Omnis Software Ltd

May 2023 - Updated Oct 2023 Revision 35659

About This Manual

This manual contains a complete list of all the functions available in Omnis Studio. The functions are in alphabetical order, but note that the external functions are prefixed with their respective package name, e.g. Fileops, OJSON, OXML, StringTable, etc.

You can find a complete list of functions in Omnis in the **Catalog** (press F9) under the **Functions** tab and arranged in functional groups. You can use the Search option to find functions that *contain* a given string, e.g. enter 'text' to find all the functions containing 'text'; in this case, the list of function groups (on the left) is filtered showing only those groups that contain any functions matching the search.

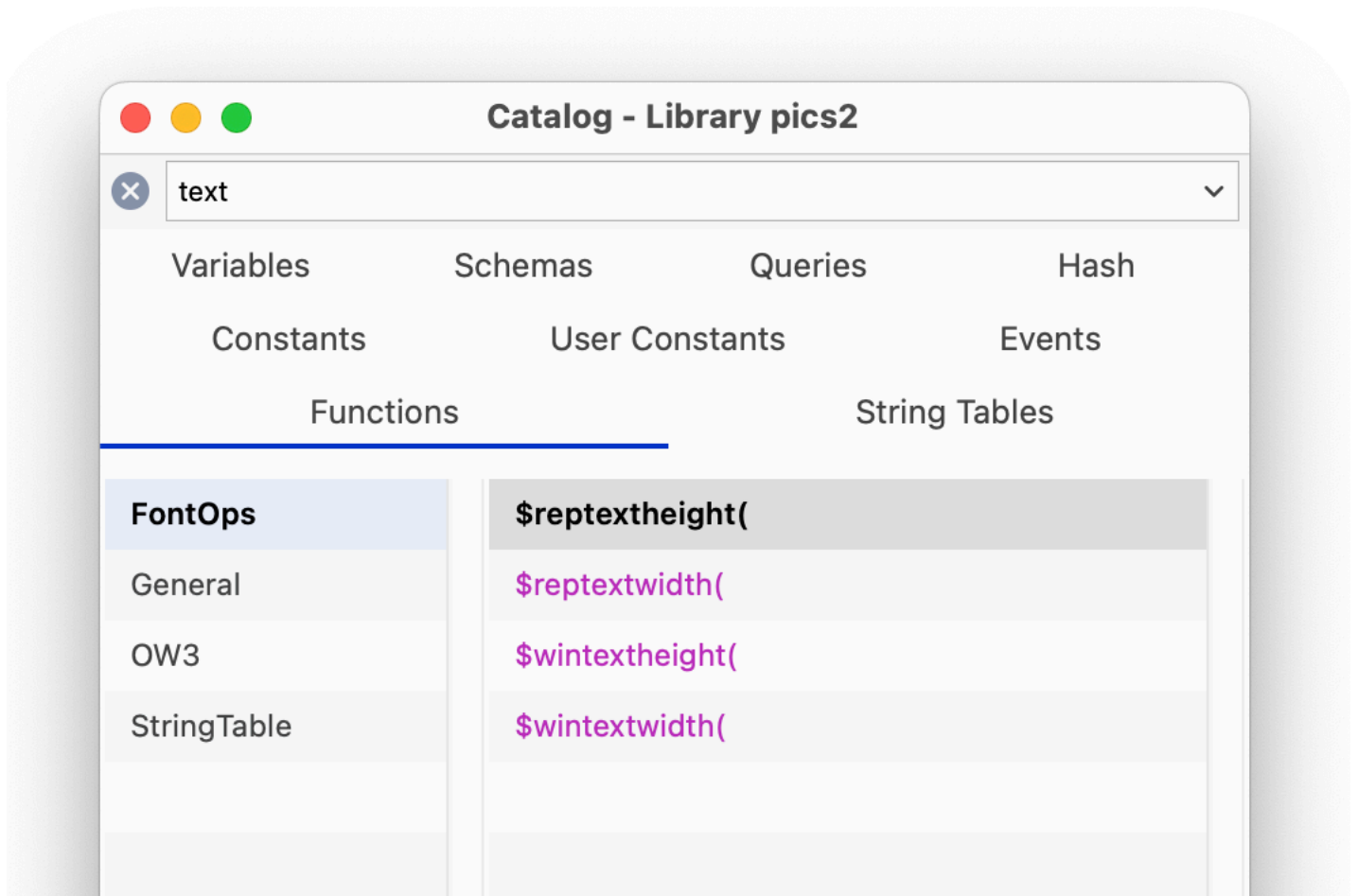


Figure 1:

Function information

Each function listed here has the following information, as well as the syntax, description, and an Omnis code example.

Function group	Execute on Client	Platform(s)
The group within the Omnis Catalog (F9).	Whether or not (YES/NO) the function can be executed in a client method in the JavaScript Client; see also Client functions	Which platform the function is available on, including Windows, macOS, Linux; All indicates the command is available on all platforms

Function Groups

<i>Binary Field</i>	<i>Class</i>	<i>Compression</i>	<i>Date and Time</i>
<i>DNet</i>	<i>External functions</i>	<i>Field</i>	<i>FileOps</i>
<i>Financial</i>	<i>Font handling</i>	<i>FontOps</i>	<i>General</i>
<i>List</i>	<i>LogarithmicLookup</i>		<i>Number</i>
<i>OIMAGE</i>	<i>OJSON</i>	<i>Omnis PDF Device</i>	<i>OmnisCalendar</i>
<i>OmnisIcon Library</i>	<i>ONOTIFY</i>	<i>OW3</i>	<i>OXML</i>
<i>Picture</i>	<i>PortProfile</i>	<i>Random</i>	<i>rcedit</i>
<i>RESTful</i>	<i>String</i>	<i>StringTable</i>	<i>Trigonometric</i>
<i>Unicode</i>			

The following functions are for desktop apps only, not web or mobile apps.

Mouse

Client Functions

The following functions can be executed in a client method in the JavaScript Client.

Function

abs()
acos()
alt()
asin()
atan()
atan2()
br()
callprivate()
cap()
charcount()
cmd()
con()
contains()

Function

cos()
ctrl()
dadd()
dat()
dim()
asof 35991 dpart()
dtsy()
dtd()
dtm()
dtw()
dty()
endswith()
errcode()
errtext()
exp()
flag()
fmdatettime()
fntshortdate()
fntshorttime()
getdatettime()
getticks()
asof 35914 hexcolor()
asof 35914 hsla()
iconidwithbadge()
int()
isclear()
isclient()
iseven()
isnull()
isodd()
asof 35925
iso8601toomnis()
isunicode()
join()
jst()
left()
len()
list()
ln()
locale()
log()
low()
max()
mid()
min()
mod()
msgcancelled()
not()
asof 35925
omnistoiso8601()
pick()
pos()
pwr()
replace()
replaceall()
rgb()
rgba()
right()

Function
<i>rnd()</i>
<i>row()</i>
<i>rpos()</i>
<i>shift()</i>
<i>sin()</i>
<i>split()</i>
<i>sqr()</i>
<i>startswith()</i>
<i>stgettext()</i>
<i>style()</i>
<i>styledtohtml()</i>
<i>tan()</i>
<i>tracelog()</i>
<i>trim()</i>
<i>unichr()</i>
<i>unicode()</i>
<i>upp()</i>
<i>useradians()</i>

Copyright info

The software this document describes is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. Names of persons, corporations, or products used in the tutorials and examples of this manual are fictitious. No part of this publication may be reproduced, transmitted, stored in a retrieval system or translated into any language in any form by any means without the written permission of Omnis Software.

© Omnis Software, and its licensors 2023. All rights reserved. Portions © Copyright Microsoft Corporation. Regular expressions Copyright (c) 1986,1993,1995 University of Toronto. © 1999-2023 The Apache Software Foundation. All rights reserved. This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Specifically, this product uses Json-smart published under Apache License 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>) © 2001-2023 Python Software Foundation; All Rights Reserved. The iOS application wrapper uses UICKeyChainStore created by <http://kishikawakatsumi.com> and governed by the MIT license. Omnis® and Omnis Studio® are registered trademarks of Omnis Software. Microsoft, MS, MS-DOS, Visual Basic, Windows, Windows Vista, Windows Mobile, Win32, Win32s are registered trademarks, and Windows NT, Visual C++ are trademarks of Microsoft Corporation in the US and other countries. Apple, the Apple logo, Mac OS, Macintosh, iPhone, and iPod touch are registered trademarks and iPad is a trademark of Apple, Inc. IBM, DB2, and INFORMIX are registered trademarks of International Business Machines Corporation. ICU is Copyright © 1995-2023 International Business Machines Corporation and others. UNIX is a registered trademark in the US and other countries exclusively licensed by X/Open Company Ltd. Portions Copyright (c) 1996-2023, The PostgreSQL Global Development Group Portions Copyright (c) 1994, The Regents of the University of California Oracle, Java, and MySQL are registered trademarks of Oracle Corporation and/or its affiliates SYBASE, Net-Library, Open Client, DB-Library and CT-Library are registered trademarks of Sybase Inc. Acrobat is a registered trademark of Adobe Systems, Inc. CodeWarrior is a trademark of Metrowerks, Inc. This software is based in part on ChartDirector, copyright Advanced Software Engineering (www.advsofteng.com). This software is based in part on the work of the Independent JPEG Group. This software is based in part of the work of the FreeType Team. Other products mentioned are trademarks or registered trademarks of their corporations.

OIMAGE

Functions

OIMAGE.\$getDimension(\$image,\$result)

OJSON

Functions

OJSON.\$arraytolist() OJSON.\$count() OJSON.\$formatjson()
OJSON.\$json() OJSON.\$list() OJSON.\$listtoobjectarray()
OJSON.\$objectarraytolist()

Omnis PDF Device

Functions

Omnis PDF Device.\$encrypt()
Omnis PDF Device.\$setdocinfo()
Omnis PDF Device.\$settemp()

Omnis ICalendar

Functions

OmnisICalendar.\$calendar\$create() OJSON.\$count()

OmnisIcn Library

Functions

OmnisIcn Library.\$getmask() OJSON.\$count()
OmnisIcn Library.\$getpict()

ONOTIFY

Functions

ONOTIFY.\$onoverbadge() OJSON.\$count()
ONOTIFY.\$setbadgecount()
ONOTIFY.\$setbadgeicon()

OW3

Functions

OW3.\$compare() OJSON.\$count()
OW3.\$make() OJSON.\$count()
OW3.\$split() OJSON.\$count()
OW3.\$stpp() OJSON.\$count()
OW3.\$unescapeuritext()

Some of these functions replace the functions in the OWEB function group (now deprecated).

OWEB

Functions

The OWEB functions have been removed, but replaced by the OW3 group.

~~OWEB.\$escape(\$arg)~~ ~~OWEB.\$getrharg(\$arg)~~ ~~OWEB.\$makeuid()~~

OXML

Functions

~~OXML.\$base64(\$str)~~ ~~OXML.\$base64(\$str)~~ ~~OXML.\$form(\$xml)~~ ~~OXML.\$toxml()~~

~~OXML.\$md5(\$str)~~ ~~OXML.\$str(\$httpheader)~~

Picture

Functions

~~pictconvfrom(\$pictconvto)~~ ~~pictconvtype(\$pictformat)~~

~~pictsize()~~

PortProfile

Functions

~~PortProfile.\$getProfile(\$set)~~ ~~PortProfile.\$getProfileNames()~~

Random

Functions

~~getseed()~~ ~~rand()~~ ~~randintrng()~~ ~~randrealrng()~~

~~rolldice()~~ ~~rollstring()~~ ~~setseed()~~ ~~shufflelist()~~

RC Edit

Functions

~~\$setapplication(\$application)~~ ~~\$seticon(\$icon)~~ ~~\$setproductversion(\$version)~~

~~\$setresource(\$resource)~~ ~~\$setversionstring(\$versionstring)~~

**RESTful
Functions**

`formatchunk()` `formathttpdate()` `formatserverresponse()` `parsehttpdate()`
`parsehttpdate()`

**String
Functions**

`asc()` `br()` `cap()` `charcount()`
`chk()` `chr()` `con()` `contains()`
`decstr()` `delchars()` `encstr()` `endswith()`
`isnumber()` `join()` `jst()` `left()`
`len()` `low()` `mid()` `natcmp()`
`pos()` `replace()` `replaceall()` `right()`
`rpos()` `rxpos()` `split()` `startswith()`
`strprbk()` `strspn()` `strtok()` `style()`
`styledtohtml()` `tr()` `upp()`

**StringTable
Functions**

`stgettext()` `StringTable.$columnnumber()`
`StringTable.$columnnumber()` `StringTable.$getcolumn()`
`StringTable.$getcolumn()` `StringTable.$loadcolumn()`
`StringTable.$loadcolumn()` `StringTable.$loadcolumn()`
`StringTable.$restorestringtable()`
`StringTable.$restorestringtable()`
`StringTable.$unloadstringtable()`

**Trigonometric
Functions**

`acos()` `asin()` `atan()` `atan2()`
`cos()` `sin()` `tan()` `useradians()`

**Unicode
Functions**

<i>chartoutf8()</i>	<i>isunicode()</i>	<i>locale()</i>	<i>nfc()</i>
<i>nfd()</i>	<i>ord()</i>	<i>unichr()</i>	<i>unicode()</i>
<i>unicnv()</i>	<i>utf8tochar()</i>		

Number

Functions

<i>abs()</i>	<i>fact()</i>	<i>fmtdp()</i>	<i>int()</i>
<i>iseven()</i>	<i>isodd()</i>	<i>mod()</i>	<i>pwr()</i>
<i>rnd()</i>	<i>sqr()</i>		

abs()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

abs(number)

Description

Returns the magnitude of a real *number* ignoring its positive or negative sign.

Example

```
Calculate lResult as abs(1002)
# returns 1002
```

```
Calculate lResult as abs('-203.45')
# returns 203.45
```

```
Calculate lResult as abs('12ABC')
# returns 0
```

acos()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

acos(number)

Description

Returns the arc cosine of a *number* in the range 0 to 180 degrees (0 to pi radians if #RAD is true), or returns 0 if the number is not in the range -1 to 1.

Example

```
Calculate lResult as acos(spr(2)/2)
# returns 45
```

alt()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

alt()

Description

Returns true if the Alt or Option key is being pressed.

ann()

Function group	Execute on client	Platform(s)
<i>Financial</i>	NO	All

Syntax

ann(rate,nper,pmt,pv,fv[,prd])

Description

Evaluates an unknown for an annuity and returns the result.

The first five arguments are mandatory, but you can replace any one of them with '?' which is the unknown value returned by the function.

Parameters: *rate* is the interest rate per payment period; *nper* is the number of payment periods, which should be an integer greater than zero; *pmt* is the payment made to you (or by you) at the end of each period; *pv* is the amount paid to you (or by you) at the start of the first period; *fv* is the amount paid to you (or by you) at the end of the final period; *prd* is optional and represents a specific period which must be between 1 and *nper*; it is used for obtaining the split of interest and capital payments in a period.

The convention is that positive values for *pmt*, *pv* and *fv* denote a payment made *to you*, and negative amounts denote a payment made *by you*. It is important to ensure that *rate*, *nper* and *pmt* all refer to the same period length. The annuity is evaluated so that the sum of all payments made to you (or by you) when compounded at the interest rate evaluates to zero.

The following example shows that, a 25 year mortgage for \$30000 at 11% pa interest payable monthly in arrears has monthly payments, paid by you, is equal to:

Example

```
Calculate lAmount as rnd(ann(.11/12,25*12,'?',30000,0),2)
# returns -294.03
```

anna()

Function group	Execute on client	Platform(s)
<i>Financial</i>	NO	All

Syntax

anna(rate,nper,pmt,pv,fv)

Description

Evaluates an unknown for an annuity in advance and returns the result.

This function works in the same way as the *ann()* function except that the annuity is calculated in advance. The payments are assumed to be made at the start of each period instead of the end of each period.

The following example shows what the monthly payment is, using the same arguments as those used for the example for *ann()*, but using the **anna()** function, where the mortgage payments are assumed to be made in advance.

Example

```
Calculate lAmount as rnd(anna(.11/12,25*12,'?',30000,0),2)
# returns -291.36
```

asc()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

asc(string,position)

Description

Returns the ASCII or Unicode value of the character at the *position* in the *string*. The first *position* in the *string* is 1.

The value returned for ASCII characters is between 0 and 255, or -1 if *position* is less than 1 or greater than the length of *string*.

Example

```
Calculate lResult as asc('Quantity',1)
# returns 81, that is the ASCII value of the 1st character Q
```

```
Calculate lResult as asc('Car',3)
# returns 114, that is the ASCII value of the 3rd character r
```

```
Calculate lResult as asc('Train',9)
# returns -1 because 9 is greater than the length of the string
```

asin()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

`asin(number)`

Description

Returns the arc sine of a *number* in the range -90 to 90 degrees (-pi/2 to pi/2 radians if #RAD is true), or returns 0 if the number is not in the range -1 to 1.

Example

```
Calculate lResult as asin(sqr(3)/2)
# returns 60
```

atan()

Function group	Execute on client	Platform(s)
Trigonometric	YES	All

Syntax

`atan(number)`

Description

Returns the arc tangent of a *number* in the range -90 to 90 degrees (-pi/2 to pi/2 radians if #RAD is true).

Example

```
Calculate lResult as atan(1)
# returns 45
```

atan2()

Function group	Execute on client	Platform(s)
Trigonometric	YES	All

Syntax

`atan2(y,x)`

Description

Returns the arc tangent of the point y,x coordinates.

Example

```
Calculate lResult as atan2(1,1)
# returns 45
```

avgc()

Function group	Execute on client	Platform(s)
List	NO	All

Syntax

avgc(listname,column[,ignore-nulls])

Description

Returns the average value for a list column specified by *listname* and *column*. `avgc()` can only be used with columns defined using variables so it cannot be used with lists defined from a SQL class.

If you set *ignore-nulls* to 1, null values are ignored and not counted. If you omit this parameter or it evaluates to zero, nulls are treated as zero values and are counted.

Example

```
Calculate lResult as avgc(lList,lCol1,1)
# returns the average for lCol1 not including null or zero values
```

basecolor()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

basecolor(color)

Description

Returns the 32bit RGB or color constant without any alpha value, extracted from the supplied color property value.

bdif()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

bdif(oldbinary,newbinary)

Description

Returns a binary representation of the differences between *oldbinary* and *newbinary*.

It is useful for comparing different versions of the same file, whether it is an Omnis library, external component, picture or text file, and so on.

Example

```
Calculate lBinDiff as bdif(lOldBinary,lNewBinary)
```

binchecksum()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

binchecksum(*binary*)

Description

Calculates an integer checksum for a *binary* field and returns the result.

Omnis generates the checksum by summing the bytes of the *binary* field, using a 32 bit number, ignoring overflow.

Example

```
Calculate lChecksum as binchecksum(lBinary)
```

bincompare()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

bincompare(*binary1*,*binary2*)

Description

Compares two binary fields, *binary1* and *binary2*. Returns true if they are equal and false if they are not.

Fields of different length are *not* equal, meaning that the rule about extending the length of the shortest field does not apply in this case.

Example

```
Calculate lStatus as bincompare(lBinary1,lBinary2)
```

binfrombase32()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binfrombase32(*vData*)

Description

Decodes the binary or character *vData* from BASE32 and returns the resulting binary data. Returns #NULL if *vData* is not valid BASE32 or an error occurs.

binfrombase64()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binfrombase64(vData[,bURLEncoding=kFalse, bExpectPadding=kTrue, bStripWhitespace=kFalse])

Description

Decodes the binary or character *vData* from BASE64 and returns the resulting binary data. Returns NULL if *vData* is not valid BASE64.

Pass *bURLEncoding* as kTrue to decode the URL-safe form of base64.

Pass *bExpectPadding* as kFalse to not expect any padding in the base64. Typically, this would be passed as kFalse when using URL-safe encoding.

Pass *bStripWhitespace* as kTrue to strip whitespace from the input data (defaults to kFalse).

binfromhex()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binfromhex(*string*)

Description

Returns a binary field value generated from the specified character *string*, typically generated using *binthex*().

The character *string* encodes a hexadecimal value in ASCII. The *string* must not contain a leading 0x or 0X.

Example

Calculate lBinary as binfromhex(lString)

binfromint32()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binfromint32(*int32*)

Description

Returns a binary field value containing the binary representation of the 32 bit integer *int32*.

binfromint64()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binfromint64(*int64*)

Description

Returns a binary field value containing the binary representation of the 64 bit integer *int64*.

binlength()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binlength(*binary*)

Description

Returns the length of a *binary* field in bytes.

Example

Calculate `lLength` as `binlength(lBinary)`

binsearch()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

binsearch(*binary1*,*binary2*[,*start-from*])

Description

Returns the zero-based position of *binary1* in *binary2*, or returns -1 if not found.

start-from is the zero-based index of the byte in *binary2* where the search is to start. If omitted, *start-from* defaults to zero at the start of *binary2*.

Example

Calculate `lPosition` as `binsearch(lBinary1,lBinary2,5)`

bintobase32()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bintobase32(vData)

Description

Encodes *vData* as BASE32 and returns the binary result. *vData* can be either binary or character. If *vData* is character, Omnis converts it to UTF-8 before encoding it as BASE32. Returns #NULL if an error occurs.

bintobase64()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bintobase64(vData[,bURLEncoding=kFalse,bAddPadding=kTrue])

Description

Encodes *vData* as BASE64 and returns the result. *vData* can be either binary or character. If *vData* is character, Omnis converts it to UTF-8 before encoding it as BASE64.

Pass *bURLEncoding* as *kTrue* to use the URL-safe form of base64.

Pass *bAddPadding* as *kFalse* to exclude the padding from the base64 (the one or two = characters appended to the end of the encoded data). Typically, this would be passed as *kFalse* when using URL-safe encoding.

bintohehex()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bintohehex(*binary*)

Description

Returns a character string representing the value of a *binary* field, in ASCII hexadecimal.

Example

Calculate lString as bintohehex(lBinary)

bintoint32()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bintoint32(*binary*)

Description

Returns a 32 bit integer value from the first 4 bytes of a *binary* field.

bintoint64()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bintoint64(*binary*)

Description

Returns a 64 bit integer value from the first 8 bytes of a *binary* field.

bitand()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitand(*binary1*,*binary2*)

Description

Performs an AND operation on *binary1* and *binary2*, and returns the result.

Example

Calculate `lNewBinary` as `bitand(lBinary1,lBinary2)`

bitclear()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitclear(*binary1*,*firstbitnumber*,*lastbitnumber*)

Description

Clears a range of bits (all bits with numbers \geq *firstBitNumber* and \leq *lastBitNumber*) operating directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the *binary1* argument, and returns 1 for success and 0 for failure. If the bit numbers identify some bits which are outside the current length of the binary field, Omnis extends the field by appending bytes with value zero, and clears the bits.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

```
Calculate lStatus as bitclear(lBinary1,lFirstBitNumber,lLastBitNumber)
```

bitfirst()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitfirst(*binary*)

Description

Returns the number of the most significant bit with value 1 in a *binary* field.

The parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

The following example sets lNumber to the bit number of the first bit set to 1. If all bits are zero, **bitfirst()** returns -1.

Example

```
Calculate lNumber as bitfirst(lBinary)
```

bitmid()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitmid(*binary*,*firstbitnumber*,*lastbitnumber*)

Description

Returns a binary value identified as a range of bits of a *binary* field, that is, bitmid() extracts the bits with numbers \geq *firstBitNumber* and \leq *lastBitNumber*.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

In the following example, bit *lFirstBitNumber* of *lBinary1* becomes bit zero of *lBinary2*, and so on.

Example

```
Calculate lBinary2 as bitmid(lBinary1,lFirstBitNumber,lLastBitNumber)
```

bitnot()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitnot(*binary*)

Description

Performs a 1's complement operating directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument *binary1*, and returns 1 for success and 0 for failure.

The parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

```
Calculate lStatus as bitnot(lBinary)
```

bitor()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitor(*binary1*,*binary2*)

Description

Performs an inclusive-OR on *binary1* and *binary2*, and returns the result.

Example

```
Calculate lStatus as bitor(lBinary1,lBinary2)
```

bitrotatel()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitrotatel(*binary*,*count*)

Description

Rotates a *binary* field to the left, by number of bits specified in *count*. Operates directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure. The vacated bits are replaced by the bits shifted off the left-hand end. If the specified number of bits is greater than the bit-length of the field, Omnis returns 0, and the field is unchanged.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

Calculate `lStatus` as `bitrotater(lBinary,lCount)`

bitrotater()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitrotater(*binary,count*)

Description

Rotates a *binary* field to the right by number of bits specified in *count*. Operates directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure. The vacated bits are replaced by the bits shifted off the right-hand end. If the specified number of bits is greater than the bit-length of the field, Omnis returns 0, and the field is unchanged.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

Calculate `lStatus` as `bitrotater(lBinary,lCount)`

bitset()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitset(*binary,firstbitnumber,lastbitnumber*)

bitset(*binary,firstbitnumber[,lastbitnumber]*)

Description

Sets a range of bits (all bits with numbers \geq *firstBitNumber* and \leq *lastBitNumber*) operating directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure. If the bit numbers identify some bits which are outside the current length of the binary field, Omnis extends the field by appending bytes with value zero, and sets the bits.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

If *lastBitNumber* is omitted, the value in *firstBitNumber* is used allowing you to set a single bit.

Example

Calculate `lStatus` as `bitset(lBinary,lFirstBitNumber,lLastBitNumber)`

bitshiftl()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitshiftl(binary,count)

Description

Shifts a *binary* field to the left by a number of bits specified in *count*. Operates directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure. Vacated bits become zero. Bits shifted past bit 0 are lost.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

```
Calculate lStatus as bitshiftl(lBinary,lCount)
```

bitshiftr()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bitshiftr(binary,count)

Description

Shifts a *binary* field to the right by a number of bits specified in *count*. Operates directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure. Vacated bits become zero. Bits shifted past the right-most bit are lost.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

Example

```
Calculate lStatus as bitshiftr(lBinary,lCount)
```

bittest()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

bittest(binary,firstbitnumber,lastbitnumber)

bittest(binary,firstbitnumber[,lastbitnumber])

Description

Tests a range of bits (all bits with numbers \geq *firstBitNumber* and \leq *lastBitNumber*) in *binary*. *binary* must either be a binary field, or a long integer, you cannot use a function in the first argument. Returns a Boolean (true if at least one bit in the range is set).

If any are 1, the function returns 1, otherwise it returns zero.

The first parameter can be a Binary or an Integer; in the case of Binary, you need to provide a variable, and not the result of a calculation.

If *lastBitNumber* is omitted, the value in *firstBitNumber* is used allowing you to test a single bit.

Example

```
Calculate lStatus as bittest(lBinary,lFirstBitNumber,lLastBitNumber)
```

bitxor()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

```
bitxor(binary1,binary2)
```

Description

Performs an exclusive-OR (XOR) on *binary1* and *binary2*, and returns the result.

Example

```
Calculate lNewBinary as bitxor(lBinary1,lBinary2)
```

br()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
br()
```

Description

Returns a style sequence that can be used as a line break in styled text when using the JavaScript client (equivalent to `style(kEscJsNewline)`).

bundif()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

```
bundif(differences,binary)
```

Description

Restores an older version of a *binary* file using the *differences* created by the *bdif()* function. The differences must be passed to an older version of the same *binary* file.

Example

Calculate `lBinary2` as `bundif(lBinDiff,lBinary1)`

bytecon()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

`bytecon(binary1,binary2)`

Description

Concatenates two binary fields *binary1* and *binary2*, and returns the result.

Note that **bytecon()** concatenates *binary2* on to the end of *binary1*.

Example

Calculate `lNewBinary` as `bytecon(lBinary1,lBinary2)`

byteget()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

`byteget(binary,byteNumber)`

Description

Returns the value (0-255) of the byte at the specified *byteNumber* in *binary*. Returns -1 if an error occurs.

bytemid()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

`bytemid(binary1,firstbytenumber,lastbytenumber)`

Description

Returns a binary value identified as a range of bytes in a binary field (all bytes between *firstByteNumber* and *lastByteNumber* inclusive).

The following example sets *lBinary2* to the value generated by extracting bytes *lFirstByteNumber* to *lSecondByteNumber* inclusive of *lBinary1*. Thus byte 0 of *lBinary2* becomes byte *lFirstByteNumber* of *lBinary1*, and so on.

Example

```
Calculate lBinary2 as bytemid(lBinary1,lFirstByteNumber,lSecondByteNumber)
```

byteset()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

```
byteset(binary,bytenumber,value)
```

Description

Sets byte *byteNumber* in *binary* to the specified *value*, operating directly on *binary*. Returns a Boolean (true for success).

The function operates directly on the argument, and returns 1 for success and 0 for failure.

Example

```
Calculate lStatus as byteset(lBinary1,lByteNumber,lValue)
```

callprivate()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

```
callprivate(method[,parameters...])
```

Description

Calls the private *method* with the specified *parameters* and returns its return value.

cap()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
cap(string)
```


Description

Returns the capitalized representation of a *string*, that is, the first letter of each and every word in the string is capitalized.

Example

```
Calculate lResult as cap('gRaVeS, hutton, MONKS')
# returns 'Graves, Hutton, Monks'
```

```
Calculate lResult as cap('on the 8TH day')
# returns 'On the 8th Day'
```

cdif()

Function group	Execute on client	Platform(s)
<i>Class</i>	NO	All

Syntax

cdif(oldclass,newclass)

Description

Returns a list of differences between two classes *oldclass* and *newclass*, the first parameter is the older class, and the second parameter is the newer class.

The **cdif()** function returns a binary representation of differences between two Omnis library classes of the same type, for example, you can compare two versions of the same window class.

An #ERRCODE value of 8095 means that the classes are identical. If an error occurs during execution, the flag is set to false, and #ERRCODE and #ERRETEXT will contain the error number and text.

Example

```
Calculate lOldClass as $windows.window1.$classdata
Calculate lNewClass as $windows.window2.$classdata
Calculate lDiffList as cdif(lOldClass,lNewClass)
```

charcount()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

charcount(string,char)

Description

Returns the number of occurrences of the character *char* in the *string*.

Example

```
Calculate lString as 'Omnis Studio is great'  
Calculate lCount as charcount(lString,'i')  
# returns 3
```

```
If charcount(lString,'i')>2  
  # returned True  
End If
```

chartoutf8()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	NO	All

Syntax

chartoutf8(*string*)

Description

Returns the binary value which is the UTF-8 encoding of the character *string*.

chk()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

chk(*string1*,*string2*,*string3*)

Description

Returns true or false depending on a character-by-character comparison of *string1* with *string2* and *string3* using the ASCII value of each character for the basis of the comparison.

Firstly, each character of *string2* is compared with the corresponding character of *string1* to ensure that, for each character, *string2* <= *string1*. A character is said to be less than or greater than another character if its ASCII code is less than or greater than the ASCII code of the corresponding character. Secondly and provided *string2* <= *string1*, each character of *string1* is compared with the corresponding character of *string3* to ensure that, for each character, *string1* <= *string3*. If *both* conditions are true, that is *string2* <= *string1* and *string1* <= *string3* are both satisfied, the function returns true, otherwise it returns false.

Example

```
Calculate lStatus as chk('b',' ','c')  
# returns true because b > ' ' and b < c
```

```
Calculate lStatus as chk('B','B','C')  
# returns true because B = B and B < C
```

```
Calculate lStatus as chk('SD04','AA00','ZZ99')  
# returns true, since for each character of the respective strings,  
# it is true that SD04 > AA00 and SD04 < ZZ99
```

```
# that is, S>=A, D>=A, 0>=0, 4>=0
# and S<=Z, D<=Z, 0<=9, 4<=9
```

```
Calculate lStatus as chk('SDA4','AA00','ZZ99')
# returns false, since in comparing the strings,
# SDA4 and ZZ99, the character A>9
```

```
Calculate lStatus as (chk('SDA4','AA00','ZZ99')+1=0+1)
# returns true
```

chr()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

```
chr(asciiCode1[,asciiCode2]...)
```

Description

Returns a string formed by concatenating the supplied ASCII character codes.

Any argument with a value less than zero or greater than 255 is ignored.

Only normal printable characters should be stored in Character or National fields. Also, since Omnis uses the character with ASCII value 0 as the end of string marker, this means that if you use this character in any other way, the part of the string following the 0 value is ignored. Control characters in the data file may also cause problems when trying to import or export data. Records with index fields which contain characters with ASCII value 255 may not have the correct index order. It is safe, however, to have unprintable characters in the text for the Transmit text commands.

Example

```
Calculate lResult as chr(66,111,111,107)
# returns 'Book'
```

```
Calculate lResult as chr(257,-1,66,111,111,107)
# returns 'Book', the first two parameters are ignored
```

cmd()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

```
cmd()
```

Description

Returns true if the Cmd key (macOS) or Control key (other platforms) is being pressed.

cmp()

Function group	Execute on client	Platform(s)
<i>Financial</i>	NO	All

Syntax

cmp(rate,periods)

Description

Returns the compound interest multiplier for a given interest *rate* over a given number of *periods*.

That is, *cmp()* evaluates the expression $(1+(rate/100))^{periods}$; the interest rate is given by the argument *rate/100*.

Example

```
Calculate lAmount as cmp(10,10) ## = (1+(10/100))10  
# returns 2.59 (to 2 decimal places)
```

```
Calculate lAmount as cmp(15,25) ## = (1+(15/100))25  
# returns 32.92 (to 2 decimal places)
```

```
Calculate lAmount as cmp(5,0.5) ## = (1+(5/100))0.5  
# returns 1.02 (to 2 decimal places)
```

cubintolist()

Function group	Execute on client	Platform(s)
<i>List</i>	NO	All

Syntax

cubintolist(*binary*)

Description

Returns the list created by converting *binary* to a list, where *binary* was created in non-Unicode Studio by assigning a list to a binary variable.

Example

```
# Assign a list to a binary variable created in a non-Unicode Studio  
Calculate lBinary as lOldList
```

```
# Converts a binary variable to a list in a Unicode Studio  
Calculate lNewList as cubintolist(lBinary)
```

complementarycolor()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

complementarycolor(*color*)

Description

Returns the complementary color given the passed *color* value. A complementary color is a color on the opposite side of the color wheel.

compress()

Function group	Execute on client	Platform(s)
<i>Compression</i>	NO	All

Syntax

compress(*binary*)

Description

Compresses the data stored in the specified *binary* variable and returns the binary compressed data.

It uses the ZLIB compression algorithm to compress the binary variable. To uncompress the result, and return the original uncompressed value, use *uncompress()*.

Example

Calculate `lCompressed` as `compress(lOriginal)`

con()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

con(*string1*,*string2*[,*string3*]...)

Description

Returns a string by concatenating two or more *string* values.

con() has a limit of 100 parameters. You can exceed this limit by using Calculate *CVAR1* as `con(CVAR2, CVAR3)` where *CVAR2* has 99 items and *CVAR3* has 99 items, and so on.

Example

```
Calculate lFirstName as 'Dick'  
Calculate lLastName as 'Rawkins'  
Calculate lName as con(lFirstName,' ',lLastName)  
# returns 'Dick Rawkins'
```

```
Calculate lResult as con('Omnis',' library')  
# returns 'Omnis library'
```

```
Calculate lResult as con('May ',8,'th 200',3)  
# returns 'May 8th 2003'
```

Note the use of spaces in the above examples

contains()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
contains(string,substring[,ignorecase=kFalse])
```

Description

Returns true if the *string* contains the non-empty string *substring*. If *ignorecase* is *kTrue*, the function uses case-insensitive comparison.

Example

```
Calculate lString as 'Build better software faster'
```

```
If con(lString,'software')  
  # returned True  
End If
```

```
Calculate lBoolean as contains(lString,'SOFTWARE',kTrue)  
# lBoolean = True
```

cos()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

```
cos(angle)
```

Description

Returns the Cosine of an *angle* where the *angle* is in degrees (or radians if #RAD is true).

Example

```
Calculate lResult as cos(60)
# returns 0.5
```

ctrl()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

ctrl()

Description

Returns true if the Control key (Cmd key on macOS) is being pressed.

cundif()

Function group	Execute on client	Platform(s)
<i>Class</i>	NO	All

Syntax

cundif(differences,class)

Description

Restores an older version of a *class* using the list of *differences* created by *cdif()*.

The **cundif()** function is used to roll back the changes made to a class after having compared two versions of the same class with *cdif()*. The list of binary *differences* must be passed to an older version of the same class.

Note that you can store multiple sets of differences or “revisions” of a class and, at any time, “reconstruct” an earlier version by successively applying **cundif()** against that particular class.

Example

```
# having created lDiffList with cdif()...
Calculate lOldClass as cundif(lDiffList,lNewClass)
If #ERRCODE
  OK message (Sound bell ) {[#ERRTEXT]}
  Quit method
End If

# now assign the binary field containing the class to a window class
Calculate lVar1 as $windows.window1.$classdata.$assign(lOldClass)
If lVar1=0 ## class in lOldClass is not a valid window
  OK message (Sound bell ) {The assign has failed...}
  Quit method
End If
```

dadd()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dadd(datepart,number,date)

Description

Adds a *number* of date parts to *date*.

The *datepart* argument can be a number of days, months, or quarters depending on the constant you use. The *number* is interpreted as the number of date or time parts or units specified by a *datepart* constant. The *number* argument must be an integer when specifying *datepart* as kYear, kMonth, kWeek, kQuarter, or kCentiSecond (the fractional part of a number is ignored). You can use fractions for the other date parts.

The datepart constants that you can use are: kYear, kMonth, kWeek, kQuarter, kDay, kHour, kMinute, kSecond, kCentiSecond.

Example

Calculate lDate as dadd(kDay,3,#D)

returns May 11, 2003, that is, 3 days are added, assume #D is May 8, 2003

Calculate lDate as dadd(kWeek,1.2,#D)

returns May 16, 2003, that is, 1 week is added, the fraction is ignored, assume #D is May 8, 2003

dat()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dat(datestring|number[,dateformat])

Description

Converts a *datestring* or *number* to a date value using an optional *dateformat* string and returns the result. JavaScript client-executed methods do not support *number*.

If you do not specify a *dateformat*, the first argument is converted using #FD. You can use the following symbols in the *dateformat* string:

Y	Year (03)	d	Day (12th)
y	Year (2003)	W	Day of week (5)
C	Century (20)	w	Day of week (Friday)
M	Month (06)	V	Short day of week (Fri)
m	Month (JUN)	E	Day of year (1-366)
n	Month (June)	G	Week of year (1-52)
D	Day (12)	F	Week of month (1-6)

Example

```
Calculate lDate as dat('May 8th, 03')
# returns '8 MAY 03' if #FD = 'D m Y'
```

```
Calculate lDate as dat('May 8th, 2003','MDY')
# returns 050803
```

```
Calculate lDate as dat(91,'w, d n, y')
# returns 'Monday, 1st April, 1901' i.e. the 91st day of 1901
```

ddiff()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

```
ddiff(datepart,date1,date2)
```

Description

Returns the difference between two dates, *date1* and *date2*, in the units specified by a *datepart* constant; the specified dates are included in the evaluation.

When you specify one of the day of the week constants (kSunday thru kSaturday) as the *datepart* argument, you get the number of occurrences of that day between the two dates. When you specify kYear,kQuarter, kMonth, or kWeek as the *datepart* argument, the function counts the end of years, quarters, months, or weeks between the two dates.

The datepart constants that you can use are: kYear, kMonth, kWeek, kQuarter, kDay, kSunday thru kSaturday, kHour, kMinute, kSecond, kCentiSecond.

Example

```
Calculate lDiff as ddiff(kMonth,"1/31/03","3/1/03")
# returns 2
```

```
Calculate lDiff as ddiff(kDay,"4/8/03",#D)
# returns 30, the number of days between the two dates, assume #D is May 8, 2003
```

decstr()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

```
decstr(string[,key,unicode=ktrue])
```

Description

Decodes *string* (encoded with *encstr()*) using *key* and returns result. *key* can be a string of 1-255 characters in length, previously used with *encstr()*. If omitted, uses the default *key*. If *unicode* false and *string* not marked as Unicode, string must have been encoded by non-Unicode Omnis.

Note that `decstr(encstr(string,key),key) = string`.

Example

```
Calculate lEncoded as encstr('Testing',10)
Calculate lDecoded as decstr(lEncoded,10)
# returns the original string 'Testing'
```

decxtea()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

decxtea(binary,key)

Description

Returns the binary result of decrypting *binary* (previously encoded using *encxtea()*) with the binary *key*; the *key* must be 128 bits long.

delchars()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

delchars(string,position[,length=1])

Description

Returns the string generated by deleting the substring at the specified 1-based *position* and *length* from *string*. If *length* extends past the end of the string it just deletes to the end.

Example

```
Calculate lString as delchars('String',2,2)
# lString is now 'Sing'
```

dim()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dim(datestring,number)

Description

Increments a *datestring* by a *number* of months and returns the result.

Months containing different numbers of days are accounted for. For example, Jan 31 96 increased by one month gives Feb 29 96, but beware, Feb 29 96 decreased by one month (using a negative value) gives Jan 29 96, not Jan 31 96.

Example

```
Calculate lDate as dim(dat('5/8/03'),15)
# returns 'AUG 8 04', if #FD = 'm D Y'
```

```
Calculate lDate as dim(dat('5/8/03'),-1)
# returns 'APR 8 03', if #FD = 'm D Y'
```

dname()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

dname(datepart,date)

Description

Returns the name of the day or month of a specified *date*, depending on a *datepart* constant which can be either kMonth or kDay.

Example

```
Calculate lDate as dname(kMonth,#D)
# returns May, assumes #D is May 8, 2003
```

DNet.\$addclass()

Function group	Execute on client	Platform(s)
<i>DNet</i>	NO	All

Syntax

DNet.\$addclass(*FileName*)

Description

Loads classes from *FileName* (or FileNames if comma-separated).

DNet.\$basefolder()

Function group	Execute on client	Platform(s)
<i>DNet</i>	NO	All

Syntax

DNet.\$basefolder()

Description

Returns the base folder of the .NET classes.

DNet.\$getenum()

Function group	Execute on client	Platform(s)
<i>DNet</i>	NO	All

Syntax

DNet.\$getenum(EnumName)

Description

Returns the value of the fully qualified specified enum (e.g. System.IO.FileMode.Open).

dpart()

Function group	Execute on client	Platform(s)
upto 35990 <i>Date and Time</i>	NO	All
asof 35991 <i>Date and Time</i>	YES	All

Syntax

dpart(datepart,date)

Description

Returns a number that represents a part of the specified *date* depending on the *datepart* constant used.

This is useful when you want to know the week number (that is, the week of the year; use *kWeek*), the day of the year or the day of the week, and so on.

The datepart constants that you can use are: *kYear*, *kMonth*, *kWeek*, *kDayofYear*, *kQuarter*, *kMonthofQuarter*, *kWeekofQuarter*, *kDayofQuarter*, *kWeekofMonth*, *kDay*, *kDayofWeek*, *kHour*, *kMinute*, *kSecond*, *kCentiSecond*.

When this function returns the week of the year (*kWeek*) the calculation is based on 1 Jan being the first day of the first week of the year, the last day of the year is week 53.

`dpart(kDayofWeek,date)` assumes Sunday is day 1.

Example

```
Calculate lDate as dpart(kWeek,#D)
# returns 19, the week number, when #D is May 8, 2003
```

```
Calculate lDate as dpart(kMonth,#D)
# returns 5, the month number, when #D is May 8, 2003
```

dtcy()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dtcy(datestring)

Description

Returns the year and century of a *datestring* as a string.

Example

```
Calculate lDate as dtcy(#D)
# returns '2003', assume #D is May 8, 2003
```

```
Calculate lDate as dtcy('05 08 03')
# returns '2003', assume #D is May 8, 2003
```

dtd()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dtd(datestring[,jsnumber=kfalse])

Description

Returns the day part of a *datestring*. For JavaScript client-executed code, *jsnumber* kTrue returns a number rather than a string; in all other cases returns a string unless it is part of a calculation.

Example

```
Calculate lDate as dtd(dat('May 8 03'))
# returns '8th'
```

```
Calculate lDate as con(dtd(dat ('May 8 03')), ' day')
# returns '8th day'
```

```
Calculate lDate as dtd(dat('May 8 03'))+20
# returns 28
```

dtm()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

dtm(datestring[,jsnumber=kfalse])

Description

Returns the month part of a *datestring*. For JavaScript client-executed code, *jsnumber* kTrue returns a number rather than a string; in all other cases returns a string unless it is part of a calculation.

Example

```
Calculate lDate as dtm(dat('May 8 2003'))  
# returns 'May'
```

```
Calculate lDate as dtm(dat('May 8 03'))+20  
# returns 25
```

```
Calculate lDate as dtm(dat(dat('May 8 03')+30))  
# returns 'June'
```

dtw()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

```
dtw(datestring[,jsnumber=kfalse])
```

Description

Returns the day of the week of a *datestring*. For JavaScript client-executed code, *jsnumber* kTrue returns a number rather than a string; in all other cases returns a string unless it is part of a calculation. dtw() assumes Monday is day 1.

Example

```
Calculate lDate as dtw(dat('May 8 1977'))  
# returns 'Sunday'
```

```
Calculate lDate as dtw(dat('May 8 03'))+20  
# returns 24
```

```
Calculate lDate as dtw(dat(dat('May 8 03')+20))  
# returns 'Wednesday'
```

dy()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

```
dy(datestring[,jsnumber=kfalse])
```

Description

Returns the year part of a *datestring*. For JavaScript client-executed code, *jsnumber* kTrue returns a number rather than a string; in all other cases returns a string unless it is part of a calculation.

The string representation of the year part of a date is the set of numeric characters representing the year, that is, 00, 01, 02, 03, and so on, while the numeric representation is the number of years since the start of the century.

Example

```
Calculate lDate as dty(dat('8 May 03'))
# returns '03'
```

```
Calculate lDate as dty(dat('May 8 03'))+20
# returns 23
```

encstr()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

```
encstr(string[,key])
```

Description

Encodes the *string* using *key* and returns the result. *key* can be a string of 1-255 characters in length. If omitted, Omnis uses the default *key*. `encstr()` marks the string to indicate that it was encoded using Unicode Omnis.

The return value of **encstr()** is a string that is difficult to decode without knowing the key. To decode the string, and return the original value, use the function `decstr()`.

Example

```
Calculate lEncoded as encstr('Testing',10)
# encodes the string 'Testing' with the key 10
```

encxtea()

Function group	Execute on client	Platform(s)
<i>Binary Field</i>	NO	All

Syntax

```
encxtea(binary,key)
```

Description

Returns the binary result of encrypting *binary* using the eXtended Tiny Encryption Algorithm (XTEA) with the binary *key*; the *key* must be 128 bits long.

endswith()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
endswith(string,end[,ignorecase=kFalse])
```

Description

Returns true if the *string* ends with the non-empty string *end*. If *ignorecase* is `kTrue`, the function uses case-insensitive comparison.

Example

```
Calculate lString as 'One Code'
```

```
If endswith(lString, 'Code')  
  # returned True  
End If
```

```
Calculate lBoolean as endswith(lString, 'CODE', kTrue)  
# lBoolean = True
```

errcode()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

errcode()

Description

Returns #ERRCODE.

errtext()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

errtext()

Description

Returns #ERRTEXT.

eval()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

eval(*string*)

Description

Evaluates a calculation expressed as a character string and returns the result.

For example, if *lVar2* contains the string '3*4/2', *eval(lVar2)* returns the result 6. You should use this function with extreme care because a runtime error will occur if the string is not a valid calculation. You can use the Test for valid calculation command to test a string before attempting to evaluate it.

Example

```
Calculate lVar2 as '3*lVar1/15.5'  
Test for valid calculation {eval(lVar2)}  
If flag true  
    Calculate lTax as eval(lVar2)  
End If
```

evalf()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

evalf(fieldname|variable)

Description

Evaluates a calculation passed as a string in a character or binary *variable* and returns the result. If *variable* is of type binary, *evalf()* returns the tokenized calculation in *variable* for faster execution of subsequent calls.

The function **evalf()** is faster than the equivalent *eval()*; you should use it when your code will repeat an evaluation several times. You should use it in the *Set search as calculation* command and as the calculation for a window list field.

evalf() takes a single argument that must be a *fieldname* or *variable*. If the contents of this field or variable is the text for a valid calculation, **evalf()** returns the result of the calculation, else a runtime error occurs. At the same time, the tokenized form of the calculation is stored back in the field or variable, so that next time **evalf()** is called, there is no need to tokenize or check the string. Tokenizing a string is part of the interpretation process; once done, Omnis can evaluate the calculation quickly. If you change the contents of the field or variable **evalf()** uses, Omnis will recognize that the new string requires checking and tokenizing.

Example

```
Test for valid calculation {evalf(lSearch)}  
If flag true  
    Set search as calculation {evalf(lSearch)}  
End If  
Find first on lTown (Use search)
```

```
Calculate lVar1 as 'lTax*100'  
Calculate lTotal as lVar2+evalf(lVar1)
```

exp()

Function group	Execute on client	Platform(s)
<i>Logarithmic</i>	YES	All

Syntax`exp(number)`**Description**

Returns e raised to the power of a given *number*, or 1e100 on overflow.

Example

```
Calculate lResult as exp(0.5)
# returns 1.6487
```

fact()

Function group	Execute on client	Platform(s)
<i>Number</i>	NO	All

Syntax`fact(number)`**Description**

Returns the factorial of a *number* rounded to an integer first. If *number* <= 0, 1 is returned, and if *number* >= 70, 1e100 is returned.

Example

```
Calculate lResult as fact(4)
# returns 24, that is 4*3*2*1
```

fday()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax`fday(datepart,date)`**Description**

Returns the date of the first day of the year, month, week or quarter in which the specified *date* falls.

The period is specified using one of the following *datepart* constants: kYear, kQuarter, kMonth, kWeek.

Example

```
Calculate lDate as fday(kWeek,#D)
# returns May 4, 2003 if the start of the week is set to Sunday
```

```
Calculate lDate as fday(kQuarter,#D)
# returns April 1 2003, that is, the first day of the quarter in which today falls, #D is May 8, 2003
```

FileOps.\$changeworkingdir()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$changeworkingdir(*cPath*)

Description

Changes the current working directory to the directory named in *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

\$changeworkingdir() only switches between directories on the same drive, not between drives. The function returns an error code, or zero if successful: see the *FileOps function error codes*.

Note: Applications on macOS do not have a working directory. If you use this function on macOS, it returns the error code kFileOp-sNoOperation.

Example

```
Switch platform()
  Case 'X' ## for macOS
    Quit method ;; working directories not supported
  Case 'U' ## for Linux
    Do FileOps.$changeworkingdir('/omnis/examples') Returns lError
  Default ## for Windows platforms
    Do FileOps.$changeworkingdir('c:\omnis\examples') Returns lError
End Switch
OK message {Working directory is now: [FileOps.$getworkingdir()]}
```

FileOps.\$converthfspathtoposixpath()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	macOS

Syntax

FileOps.\$converthfspathtoposixpath(*cHfsPath*,&*cPosixPath*)

Description

Converts an HFS file path to a POSIX file path. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

An HFS file path is colon-delimited, where a leading colon indicates a relative path, otherwise the first component denotes the volume. A POSIX path is slash-delimited.

On any other platform, other than macOS, **\$converthfspathtoposixpath()** simply copies *cHfsPath* to *cPosixPath*.

Example

```
Do FileOps.$converthfspathtoposixpath(":Applications:Calculator",lPosixPath)
# lPosixPath is returned as "Applications/Calculator"
```

FileOps.\$convertposixpathtohfspath()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	macOS

Syntax

FileOps.\$convertposixpathtohfspath(*cPosixPath*,&*cHfsPath*)

Description

Converts a POSIX file path to an HFS file path. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

An HFS file path is colon-delimited, where a leading colon indicates a relative path, otherwise the first component denotes the volume. A POSIX path is slash-delimited.

On any other platform, other than macOS, **\$convertposixpathtohfspath()** simply copies *cPosixPath* to *cHfsPath*.

Example

```
Do FileOps.$convertposixpathtohfspath("Applications/Calculator",lHfsPath)
# lHfsPath is returned as ":Applications:Calculator"
```

FileOps.\$copyfile()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$copyfile(*cFromPath*,*cToPath*)

Description

Copies the file specified in *cFromPath* to the file *cToPath*. The file named in *cToPath* must not already exist. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

cToPath is the path to destination folder into which the file will be copied; the file to be copied must not already exist in the destination folder.

The function returns an error code, or zero if successful: see the *FileOps function error codes*.

To move a file, rather than copy it, use *\$movefile()*.

Example

```
Do FileOps.$copyfile('c:\omnis\test.txt','c:\examples\test.txt') Returns lError
# copies 'test.txt' to the 'examples' folder
Do FileOps.$copyfile('c:\omnis\test.txt','c:\examples\test2.txt') Returns lError
# copies 'test.txt' to 'test2.txt' in the 'examples' folder
Do FileOps.$copyfile('c:\omnis\test.txt','c:\omnis\test2.txt') Returns lError
# copies 'test.txt' to 'test2.txt' in the same folder
```

FileOps.\$createdir()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$createdir(*cPath*[,bCreateParentDirs=kFalse])

Description

Creates a directory with the pathname *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

With the exception of the last directory, all directories specified in the *cPath* parameter must already be present, unless *bCreateParentDirs* is *kTrue*, in which case any parent directories in *cPath* that do not exist are also created.

If *bCreateParentDirs* is *kTrue* and the directory in *cPath* already exists, *FileOps* will return 0 (success) rather than 101215 (A file with the specified name already exists). If *bCreateParentDirs* is *kFalse* (default), or is omitted, then the usual 101215 is returned if directory in *cPath* already exists.

The function returns an error code, or zero if successful: see the *FileOps function error codes*.

Example

```
Do FileOps.$createdir('c:\omnis\examples\extcomp\clock') Returns lError
# creates the 'clock' folder assuming c:\omnis\examples\extcomp is a valid path
```

FileOps.\$deletefile()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$deletefile(*cPath*[deleteContents=kFalse,recursive=kFalse])

Description

Deletes the file or folder named *cPath*. Pathnames can be over 255 characters. The optional parameters *deleteContents* and *recursive* allow you to delete the contents of a folder (both default to *kFalse*, if omitted).

Files deleted with **\$deletefile()** are not moved into the Recycled bin or Trash can, *they are deleted irreversibly*. You can delete a folder with **\$deletefile()**, but only if it is empty.

The combination of the parameters has the following results:

- *FileOps.\$deletefile(cPath, kFalse, kFalse)*
(same as if *deleteContents* and *recursive* are omitted) if *cPath* is a folder, the folder will be deleted only if empty, that if no files or directories are present in the folder. On macOS, the function checks if the only file is *.DS_Store* and will delete the folder in that case. If there are hidden files in the folder, the folder and the hidden files will not be deleted.
- *FileOps.\$deletefile(cPath, [kTrue, kFalse])*
if *cPath* is a folder, *all files* inside the folder if *deleteContents* is true, but any subfolders *will not* be deleted. Furthermore, if the folder is empty, the folder will not be deleted.
- *FileOps.\$deletefile(cPath, kFalse, kTrue)*
if *cPath* is a folder, it will be deleted only if empty. If there are any contents, those will not be deleted as *deleteContents* is *kFalse*, even if *recursive* is *kTrue*.
- *FileOps.\$deletefile(cPath, kTrue, kTrue)*
if *cPath* is a folder, it will be wiped off the disk with its contents, if any are present.

- FileOps.\$deletefile(cFilePath)
if cPath is a file, the file will be removed regardless of the deleteContents or recursive values - the deleteContents and recursive parameters do not impact the behavior of deleting files, only the behavior of deleting folders.

The function returns an error code, or zero if successful: see the *FileOps function error codes*.

Example

```
Do FileOps.$deletefile('c:\omnis\examples\extcomp\test1.txt') Returns lError
# deletes 'test1.txt' from 'c:\omnis\examples\extcomp'
Do FileOps.$changelworkingdir('c:\omnis') Returns lError
Do FileOps.$deletefile('test2.txt') Returns lError
# deletes 'test2.txt' at the current folder 'c:\omnis'
Do FileOps.$deletefile('c:\omnis\examples\extcomp\clock') Returns lError
# deletes the 'clock' folder if empty
```

FileOps.\$doesfileexist()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$doesfileexist(cPath)

Description

Returns true if the file or folder named *cPath* exists. Otherwise, it returns false. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

Example

```
Switch platform()
Case 'X' ## for macOS
Do FileOps.$doesfileexist('/Omnis/Tutorial/mylibs.lbs') Returns lStatus
Case 'U' ## for Linux
Do FileOps.$doesfileexist('/omnis/tutorial/mylib.lbs') Returns lStatus
Default ## for Windows
Do FileOps.$doesfileexist('c:\omnis\tutorial\mylibs.lbs') Returns lStatus
End Switch
# lStatus is true if 'mylib.lbs' exists in the Omnis tutorial
```

FileOps.\$filelist()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$filelist(iInclude,cPath[,iInformation,cFilter,&cErrorText])

Description

Returns a list of files, directories, and volumes at the location specified in *cPath*. Errors are reported to *cErrorText*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

The parameter *iInclude* specifies what to include in the list using a combination of the constants *kFileOpsIncludeFiles*, *kFileOpsIncludeDirectories*, and *kFileOpsIncludeVolumes*. Use + to combine the constants.

If you use *kFileOpsIncludeVolumes*, then the function adds a line for each volume to the start of the list. The remaining lines in the list, or all lines if you do not use *kFileOpsIncludeVolumes*, comprise a line for each file or directory in the directory *cPath*. You can restrict the files returned in the list using the optional filter *cFilter*. Note that *cFilter* does not apply to volume or directory names. The filter is a semicolon separated list of file-matching specifications. On all platforms, a file-matching specification can have the form **.<ext>*, to match all files with the extension *<ext>*, or **** to match all files. On macOS, a file-matching specification can also have the form *'TYPE'*, *'TYPE,CREATOR'*, or *'CREATOR'*, where *TYPE* and *CREATOR* are the 4 character, case-sensitive Mac file type and creator. Note that in this case, the single quotes are part of the file-matching specification. Example file-matching specifications:

**.txt* Matches files with *.txt* as their extension

**.txt;*.lbs* Matches files with *.txt* or *.lbs* as their extension

"TEXT";"OO\$\$";.txt* Matches files of type *TEXT*, or created by Omnis Studio, or with *.txt* as their extension.

"OO\$A,OO\$\$" Matches Omnis libraries used under macOS.

You specify the information to be returned by passing information as a combination of the *kFileOpsInfo...* constants listed in the following table. If you omit *iInformation*, only the name is returned. The returned list contains a column for each *kFileOpsInfo...* constant you specify. The columns you specify occur in the same order as the table.

Col name	information constant	description
name	<i>kFileOpsInfoName</i>	name of the file
name83	<i>kFileOpsInfoName83</i>	DOS 8.3 name of the file
fullname	<i>kFileOpsInfoFullName</i>	full pathname of the file
readonly	<i>kFileOpsInfoReadOnly</i>	file's read-only status
hidden	<i>kFileOpsInfoHidden</i>	file's hidden status (* see below)
size	<i>kFileOpsInfoSize</i>	logical size of file
actualsize	<i>kFileOpsInfoActualSize</i>	physical size of file on disk; same as logical size under Windows
created	<i>kFileOpsInfoCreated</i>	date and time the file was created
modified	<i>kFileOpsInfoModified</i>	date and time the file was modified
creator	<i>kFileOpsInfoCreatorCode</i>	the file's creator under macOS, blank under Windows
type	<i>kFileOpsInfoTypeCode</i>	the file's type under macOS, the file extension under Windows

* On Linux, filenames beginning with *."* (hidden files) are not identified as hidden.

The function returns an empty list if an error occurs. The *FileOps* object supports 64-bit integers for file sizes and for offsets, etc in files.

Example

```
Do FileOps.$filelist(kFileOpsIncludeFiles+kFileOpsIncludeDirectories,sys(115)) Returns lFileList
# returns a list of files and directories in the main Omnis folder
```

```
Do FileOps.$filelist(
kFileOpsIncludeFiles,con(sys(115),'external'),kFileOpsInfoName+kFileOpsInfoCreated+kFileOpsInfoSize,'*.dll') R
# returns a list of DLLs in the Omnis\External folder including the name, size, creation date and time of each
```

```
Do FileOps.$filelist(
kFileOpsIncludeFiles,'c:\windows',kFileOpsInfoSize+kFileOpsInfoFullName+kFileOpsInfoReadOnly+kFileOpsInfoCreat
# returns a list of files in the c:\windows folder including the fullname, read-only, size, creation date and
```

FileOps.\$getfileinfo()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$getfileinfo(*cPath*,*iInformation*)

Description

Returns a list of file information for the file named *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

You specify the information to be returned by passing *iInfoFlags* as a combination of the *kFileOpsInfo...* constants listed in the following table. If you omit *iInfoFlags*, only the name is returned. The returned list contains a column for each *kFileOpsInfo...* constant you specify. The columns you specify occur in the same order as the table.

Col name	info flags constant	description
name	<i>kFileOpsInfoName</i>	name of the file
name83	<i>kFileOpsInfoName83</i>	DOS 8.3 name of the file
fullname	<i>kFileOpsInfoFullName</i>	full pathname of the file
readonly	<i>kFileOpsInfoReadOnly</i>	file's read-only status
hidden	<i>kFileOpsInfoHidden</i>	file's hidden status
size	<i>kFileOpsInfoSize</i>	logical size of file
actualsize	<i>kFileOpsInfoActualSize</i>	physical size of file on disk; same as logical size u
created	<i>kFileOpsInfoCreated</i>	date and time the file was created
modified	<i>kFileOpsInfoModified</i>	date and time the file was modified
creator	<i>kFileOpsInfoCreatorCode</i>	the file's creator under macOS, blank under Win
type	<i>kFileOpsInfoTypeCode</i>	the file's type under macOS, the file extension u
linkedname	<i>kFileOpsInfoLinkedName</i>	macOS: the path of the linked file (if any)Windows (if any)
isdirectory	<i>kFileOpsInfoIsDirectory</i>	if the path specified points to a directory
calcdirectorysize	<i>kFileOpsInfoCalcDirectorySize</i>	the size in bytes of the directory on the file syste

The function returns an empty list if an error occurs. The *FileOps* object supports 64-bit integers for file sizes and for offsets, etc in files.

Example

```
Do sys(10) Returns lPath
```

```
# returns the name and path of the current library
```

```
Do FileOps.$getfileinfo(lPath,kFileOpsInfoName+kFileOpsInfoSize+kFileOpsInfoCreated) Returns lFileList
```

```
# returns the name, size, creation date and time of the current library
```

```
Do lFileList.$redefine(lFilename,lSize,lCreated)
```

```
Do lst(lFileList,1,lSize) Returns lSize
```

```
# returns the value in the size column
```

FileOps.\$getfilename()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$getfilename(&*cPath* [,*cPrompt*,*cFilter*,*cInitialDirectory*,*iAppFlags*])

Description

Opens the standard Open file dialog.

You can specify the dialog title in *cPrompt*, and restrict the files displayed using a filter passed in *cFilter*. The filter comprises one or more entries separated by the vertical bar character '|'. Each filter entry has two components, again separated by a vertical bar. The filter entries populate the file-type selection dropdown or popup list. The first component of an entry is the text to display in the dropdown or popup list. The second component is a file-matching specification. On all platforms, a file-matching specification can have the form **.<ext>*, to match all files with the extension *<ext>*, or **** to match all files. On macOS, a file-matching specification can also have the form *TYPE* or *TYPE,CREATOR*, where *TYPE* and *CREATOR* are the 4 character, case-sensitive Mac file type and creator. Example *cFilter* parameters:

Text files	*.txt
Text files	*.txt

You can also specify the initial directory for the Open dialog in *cInitialDirectory*.

The *iAppflags* parameter only applies when running on macOS. It is optional; if specified, it must be a combination of the following constants:

kFileOpsSelectApps	Allows applications to be selectable within the dialog
kFileOpsOpenApps	Allows the dialog to browse the content of applications

Note that *kFileOpsOpenApps* automatically implies *kFileOpsSelectApps*, although it is valid to specify both flags (*kFileOpsSelectApps* + *kFileOpsOpenApps*).

The name and full path of the file selected by the user is returned in the *cPath* parameter. Note that the file is not open: you must do something with the file name and path returned. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

Note that if the file selected is an alias/shortcut/link, then the returned value is the result of resolving the alias/shortcut/link to its target, not the path of the alias/shortcut/link.

\$getfilename() returns true if the user selected a file and pressed OK. Otherwise, it returns false, meaning that the user pressed Cancel.

Example

```
Do FileOps.$getfilename(lPath,'Please locate the Omnis help file','Help files|*.ohf',con(sys(115),'help')) Ret
# returns the name and full path of the file selected, e.g. 'c:\omnis\help\omnis\omnis.ohf'
```

FileOps.\$getlasterror()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$getlasterror([&cErrorText])

Description

Returns any errors in *cErrorText* from the last FileOps method that was executed.

Returns the error code from the last FileOps method executed; also optionally populates *cErrorText* with a description of the error. If no error occurred, the method returns zero and the error text is empty.

FileOps.\$getunixpermissions()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	Linux, macOS

Syntax

FileOps.\$getunixpermissions(*cPath*)

Description

Returns the Unix permissions string for the file identified by *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

If an error occurs, it returns an empty string. If it succeeds, then it returns the standard 10 character Unix file permissions string for the file. The first character is either d for a directory, or - for a file. The remaining 9 characters are split into three groups of three characters. The groups are for the user categories owner, group and other, in that order. The characters within each group have the following meanings:

Character index	Value
1	r means that members of the user category can read the file; - means they cannot.
2	w means that members of the user category can write to the file; - means they cannot.
3	x means that members of the user category can execute the file; - means they cannot.

Example

```
Do FileOps.$getunixpermissions('/omnis/myfile') Returns lPermissions
# returns '-rwxr-xr-x' meaning that the file is only writable by its owner, and readable and executable by all
```

FileOps.\$getworkingdir()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	Windows

Syntax

FileOps.\$getworkingdir()

Description

Returns the current working directory (no parameters required), so is only useful when running in Windows. The function returns an empty string if an error occurs.

Note: Applications on macOS do not have a working directory. If you use this function on macOS, it returns an empty string.

Example

```
Do FileOps.$changeworkingdir(sys(115)) Returns lError
Do FileOps.$getworkingdir() Returns lWorkDir
```

FileOps.\$movefile()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$movefile(*cFromPath*,*cToPath*)

Description

Moves the file *cFromPath* to the new location *cToPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

cFromPath is the pathname of the file to be moved. *cToPath* is the pathname of the destination folder into which the file will be moved (if you include a file name in *cToPath* the function will fail under macOS); the file to be moved must not already exist in the destination folder. The function returns an error code, or zero if successful: see the *FileOps function error codes*.

To copy a file, rather than move it, use *\$copyfile()*.

Note that **\$movefile()** cannot move a file across volumes. Use *\$copyfile()* and *\$deletefile()* instead.

Example

```
Do FileOps.$movefile(
'c:\omnis\examples\extcomp\extcomp.lbs', 'c:\omnis\startup\extcomp.lbs') Returns lError
# moves the library 'extcomp.lbs' to the Omnis\Startup folder
```

FileOps.\$parentdir()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$parentdir(*cPath*)

Description

Returns the pathname of the directory containing the file or directory identified by *cPath*, i.e. its parent directory. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

FileOps.\$putfilename()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$putfilename(&*cPath* [, *cPrompt*, *cFilter*, *cInitialDirectory*, *iAppflags*])

Description

Opens the standard Save file dialog.

You can supply a default value for the file name in *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

You can specify the dialog title in *cPrompt*.

You can use *cFilter* on Windows and Linux in a similar way to the *cFilter* parameter of *\$getfilename()*.

You can also specify the initial directory for the Save dialog in *cInitialDirectory*.

The *iAppFlags* parameter is a combination of the following constants:

kFileOpsSelectApps	If passed allows applications to be selectable within the dialog (macOS only)
kFileOpsOpenApps	If passed allows the dialog to browse the content of applications (macOS only)
kFileOpsWindowsDisablePrompt	If passed the file already exists prompt is disabled: see below* (Windows only)

Note that kFileOpsOpenApps automatically implies kFileOpsSelectApps, although it is valid to specify both flags (kFileOpsSelectApps + kFileOpsOpenApps).

*Studio 11 or above: When running on Windows, the dialog opened by *FileOps.\$putfilename()* now prompts if the file already exists, unless kFileOpsWindowsDisablePrompt is passed in the *iAppFlags* parameter. This makes the default behavior compatible with macOS (the file already exists prompt cannot be disabled on macOS).

The name and full path of the file entered by the user is returned in *cPath*. Note that the file is not saved: you must write the code to output the file. **\$putfilename()** returns true if the user selected a file and pressed OK. Otherwise, it returns false, meaning that the user pressed Cancel.

Example

```
Calculate lPath as 'Default.rep'  
# set the default name
```

```
Do FileOps.$putfilename(lPath,'Save print file','*.rep','c:\omnis\examples') Returns lError  
# opens the Save dialog with the title 'Save print file' and returns the name and full path of file entered by
```

FileOps.\$readfile()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$readfile(*cFilePath*,&*vVariable*[,*iEnc*=kUniTypeAuto])

Description

Reads the file *cFilePath* into the variable *vVariable*. Returns an integer error code (zero for success).

cFilePath (Character 100000000) The pathname of the file.

vVariable (Variant) The variable into which the file contents will be read. Binary or Character. If it is Character, the file contents are converted to Character using the encoding specified by the *iEnc* parameter.

iEnc (Integer 32 bit, default is kUniTypeAuto) The encoding used to convert the file contents to Character. A kUniType... constant, not kUniTypeBinary or kUniTypeCharacter. Only used when *vVariable* is a Character variable.

See *FileOps.\$writefile()*.

FileOps.\$rename()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$rename(*cOldname*,*cNewname*)

Description

Renames the file or folder named *cOldname* to *cNewname*.

The function returns an error code, or zero if successful: see the *FileOps function error codes*.

Example

```
Do FileOps.$rename('c:\omnis\libs','c:\omnis\examples') Returns lError
# renames the 'libs' folder to 'examples'
```

```
Do FileOps.$changeworkingdir('c:\omnis\datafile\odbc') Returns lError
Do FileOps.$rename('odbc.txt','readme.txt') Returns lError
# switches to the 'c:\omnis\datafile\odbc' folder and renames 'odbc.txt' to 'readme.txt'
```

FileOps.\$selectdirectory()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$selectdirectory(&cPath[,cPrompt,cInitialDirectory,iAppflags])

Description

Opens the Select folder dialog.

You can specify the dialog title in *cPrompt*, and the initial directory in *cInitialDirectory*.

The *iAppflags* parameter only applies when running on macOS. It is optional; if specified, it must be a combination of the following constants:

kFileOpsSelectApps	Allows applications to be selectable within the dialog
kFileOpsOpenApps	Allows the dialog to browse the content of applications

Note that kFileOpsOpenApps automatically implies kFileOpsSelectApps, although it is valid to specify both flags (kFileOpsSelectApps + kFileOpsOpenApps).

The name and full path of the folder selected by the user is returned in *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier. The function returns true if the user selected a directory and pressed OK. Otherwise, it returns false, meaning that the user pressed Cancel.

Example

```
Do FileOps.$changeworkingdir(sys(115)) Returns lError
Do FileOps.$getworkingdir() Returns lWorkDir
Do FileOps.$selectdirectory(lPath,'Select a folder',lWorkDir) Returns lError
# switches to the 'c:\omnis' folder and prompts the user to select a folder
```

FileOps.\$setfileinfo()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$setfileinfo(*cPath*,*i*InformationFlag,*v*InfoValue,...)

Description

Sets file information for the file named *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

You can use **\$setfileinfo()** to change the read-only and hidden status of a file. You pass the relevant *kFileOpsInfo...* constant in *iInfoFlag*, and the new value in *vInfoSetting*. If desired, you can then pass the other constant and its new value in the next two parameters. The function returns an error code, or zero if successful: see the *FileOps function error codes*.

Example

```
Do FileOps.$setfileinfo('c:\omnis\meths.txt',kFileOpsInfoReadOnly,kTrue,kFileOpsInfoHidden,kFalse) Returns 1Er
# sets the file 'meths.txt' to be read-only and not hidden
```

FileOps.\$setunixpermissions()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	Linux, macOS

Syntax

FileOps.\$setunixpermissions(*cPath*,*cPermissions*)

Description

Sets the Unix permissions for the file identified by *cPath*. Pathnames can be over 255 characters, which was the limit in Studio 8.0.1 or earlier.

The parameter *cPermissions* has the same syntax as the permissions string returned by *\$getunixpermissions()*.

FileOps.\$splitpathname()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$splitpathname(*cPath*, &*cDriveName*, [&*cDirectoryName*, &*cFileName*, &*cFileExtension*])

Description

Splits the path *cPath* into *cDriveName*, *cDirectoryName*, *cFileName*, and *cFileExtension*. Pathnames can be over 255 characters (the limit in Studio 8.0.1 or earlier): the last three parameters are optional (from Studio 10.0.1 onwards).

The function returns zero if successful, or an error code: see the *FileOps function error codes*.

Example

```
Do sys(10) Returns lPath
# returns the name and path of current library

Do FileOps.$splitpathname(lPath,lDrive,lDirName,lFilename,lFileExtn) Returns lError
# under Windows, when lPath = 'C:\OMNIS\EXAMPLES\EXTCOMP.LBS'
# lDrive returns C:
# lDirName returns \OMNIS\EXAMPLES\
# lFileName returns EXTCOMP
# lFileExtn returns .LBS

Do FileOps.$splitpathname('c:\omnis',lDrive,lDirName,lFilename,lFileExtn) Returns lError
# under Windows
# lDrive returns C:
# lDirName returns \
# lFileName returns OMNIS
# lFileExtn returns (empty)

Do FileOps.$splitpathname(
'HD/Omnis/Examples/Extcomp.lbs',lDrive,lDirName,lFilename,lFileExtn) Returns lError
# under macOS ## note the separator is /, not :
# lDrive returns HD
# lDirName returns /Omnis/Examples/
# lFileName returns Extcomp
# lFileExtn returns .lbs

Do FileOps.$splitpathname('/omnis/examples/extcomp.lbs',lDrive,lDirName,lFilename,lFileExtn) Returns lError
# under Linux
# lDrive returns (empty)
# lDirName returns /omnis/examples
# lFileName returns extcomp
# lFileExtn returns .lbs
```

FileOps.\$writefile()

Function group	Execute on client	Platform(s)
<i>FileOps</i>	NO	All

Syntax

FileOps.\$writefile(*cFilePath*, *vVariable* [*iEnc*=kUniTypeUTF8, *bBOM*=kTrue, *bReplace*=kTrue])

Description

Writes the file *cFilePath* with the contents of variable *vVariable*. Returns an integer error code (zero for success).

cFilePath (Character 100000000) The pathname of the file.

vVariable (Variant) The variable into which the file contents will be read. Binary or Character. If it is Character, the file contents are converted to Character using the encoding specified by the *iEnc* parameter.

iEnc (Integer 32 bit, default is kUniTypeUTF8) The encoding used to convert the file contents to Character. A kUniType... constant, not kUniTypeBinary or kUniTypeCharacter. Only used when *vVariable* is a Character variable.

bBOM (Boolean, default is kTrue) If true, and character data is to be encoded as Unicode, \$writefile adds a Unicode Byte Order Marker at the start of the file.

bReplace (Boolean, default is kTrue) Specifies what occurs if the file already exists before calling FileOps.\$writefile(). If *bReplace* is true FileOps.\$writefile() replaces the file; if *bReplace* if false, FileOps.\$writefile() returns an error.

See FileOps.\$readfile().

FileOpsObj.\$closefile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$closefile()

Description

Closes the file.

If you open a file using \$openfile(), \$openresources(), \$create() or \$createtmpfile(), then you should close it by calling this function, when you have finished using it. This function does not return a value.

FileOpsObj.\$createfile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$createfile(cFilePath[,cType,cCreator])

Description

Creates and opens an empty file with pathname cFilePath (replaces an existing file with an empty file). For macOS, cType and cCreator are file and creator types for new file. Returns kTrue for success

If the file already exists, \$createfile() replaces it with an empty file. All directories in the pathname specified in the cFilePath parameter must already be present. The optional parameters are only available on macOS.

FileOpsObj.\$createtmpfile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$createtmpfile()

Description

Creates and then opens a temporary file, ready for you to write data to it. Returns kTrue for success.

\$createtmpfile() returns a boolean value, which is true if the temporary file was successfully created and then opened.

After a file has been opened, after using \$createtmpfile(), the **\$filepath** property of the FileOps object instance will contain the path to the file. This is an alternative or shortcut to returning the file name in the list provided by the \$getfileinfo() function. If there is no open file or the file has been closed, \$filepath is empty.

FileOpsObj.\$getfileinfo()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$getfileinfo(iInfoFlags)

Description

Returns a list containing file information for the open file associated with the object. iInfoFlags is a combination of kFileOpsInfo... constants; see the FileOps.\$getfileinfo static function for a list of kFileOpsInfo constants. If an error occurs, the function returns an empty list.

After a file has been opened, e.g. after using \$createtmpfile(), the **\$filepath** property of the FileOps object instance will contain the path to the file. This is an alternative or shortcut to returning the file name in the list provided by the \$getfileinfo() function. If there is no open file or the file has been closed, \$filepath is empty.

FileOpsObj.\$getfilesize()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$getfilesize()

Description

Returns the size in bytes of the open file associated with the object. If an error occurs, the function returns -1.

FileOpsObj.\$getlasterror()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$getlasterror([&cErrorText])

Description

Returns the error code from the last FileOps object method executed; also optionally populates cErrorText with a description of the error. If no error occurred, returns zero and the error text is empty.

FileOpsObj.\$getposition()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$getposition()`

Description

Returns the current position of the open file associated with the object.

The current position is the byte index to the position in the file where the next function which reads or writes data, will read or write. The first byte in the file has position zero. If an error occurs, the function returns -1.

FileOpsObj.\$isopen()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$isopen()`

Description

Returns true if there is an open file associated with the object, false otherwise.

FileOpsObj.\$openfile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$openfile(cFilePath[,bReadOnly=kFalse])`

Description

Opens an existing file. `cFilePath` is the pathname of the file to be opened. Returns `kTrue` for success.

The `bReadOnly` parameter is optional. It is a boolean which defaults to false. If you pass true, then `$openfile()` will open the file in read-only mode.

`$openfile()` returns a boolean value, which is true if the file was successfully opened.

FileOpsObj.\$openresources()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$openresources(cFilePath[,bReadOnly=kFalse])`

Description

Opens the resource fork of the specified file. This function applies to macOS only. Returns `kTrue` for success.

`cFilePath` is the pathname of the file for which the resource fork is to be opened.

The `bReadOnly` parameter is optional. It is a boolean which defaults to `false`. If you pass `true`, then `$openresources()` will open the resource fork in read-only mode.

`$openresources()` returns a boolean value, which is `true` if the resource fork was successfully opened.

FileOpsObj.\$readcharacter()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

```
$readcharacter(iEnc,&cData[,&iOff=0,iMax=0,bChkUTF8=kFalse])
```

Description

Reads characters encoded using `iEnc` (`kUniType...` (not `Bin/Char`)) into `cData`. Reads entire file, or `iMax` chars from byte offset `iOff` and returns next `iOff`. Returns `kTrue` for success.

FileOpsObj.\$readfile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

```
$readfile(&xVariable[,iStartPos,iLength])
```

Description

Reads data from the open file associated with the object. Binary `xVariable` receives the data read from the file. Returns `kTrue` for success

You are recommended to use `$readcharacter()` to read character data, otherwise you may get unexpected results.

If you omit `iStartPos` and `iLength`, `$readfile()` reads all the remaining data from the current position to the end of the file; you can set the current position using the function `$setposition()`. Otherwise, you can specify the position from where the data is to be read, using the `iStartPos` parameter, and you can specify the number of bytes to read using the `iLength` parameter. `iLength` must not exceed the data available from `iStartPos` to the end of the file.

`$readfile()` returns a boolean value, which is `true` if the data was successfully read. After reading the data, the current position addresses the byte after the last byte read, or end of file, if no further data remains.

FileOpsObj.\$selectfilesinsystemviewer()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

```
$selectfilesinsystemviewer(cFileOrFolder[,IFileList])
```

Description

Opens the system file viewer (Explorer or Finder) and selects the specified file or files.

cFileOrFolder is either the pathname of a single file to select (when \$selectfilesinsystemviewer is called with a single parameter) or the pathname of the folder containing the files to select (when \$selectfilesinsystemviewer is called with 2 parameters).

IFileList is a single column list of file names, specifying the files to be selected in the specified folder.

The function returns an error code, or zero if successful: See the *FileOps function error codes*.

FileOpsObj.\$setfileinfo()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$setfileinfo(iInfoFlag,vInfoValue,...)

Description

Sets a file information item for the open file associated with the object to vInfoValue. iInfoFlag is a kFileOpsInfo... constant. Returns an integer error code, zero for success.

The function returns an error code, or zero if successful: See the *FileOps function error codes*.

FileOpsObj.\$setfilesize()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$setfilesize(iNewSize)

Description

Sets the file size to iNewSize bytes, for the open file associated with the object. Any data after the first iNewSize bytes is lost. Returns kTrue for success.

\$setfilesize() returns a boolean, which is true if the new size was successfully set.

FileOpsObj.\$setposition()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

\$setposition(iNewPos)

Description

Sets the current position, for the open file associated with the object. Returns kTrue for success.

The current position is the byte index to the position in the file where the next function which reads or writes data, will read or write. The first byte in the file has position zero. \$setposition() returns a boolean, which is true if the new position was successfully set.

FileOpsObj.\$writecharacter()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$writecharacter(iEnc,cData[,bAppend=kFalse,bBOM=kTrue])`

Description

Writes cData to the file, encoded using the encoding iEnc (kUniType... (not Auto/Bin/Char)). If bAppend is kFalse, the entire file contents are replaced.

bBOM controls if a BOM is added or not. If true, and the data is to be written at the start of the file, a Unicode Byte Order Marker (BOM) is added at the start of the file.

Returns kTrue for success.

FileOpsObj.\$writefile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

`$writefile(xVariable[,iStartPos])`

Description

Writes data to the open file associated with the object. Binary xVariable contains the data to write to the file. Returns kTrue for success.

You are recommended to use \$writecharacter() to write character data, otherwise you may get unexpected results.

By default, \$writefile() writes the data starting at the current position. You can specify an alternative position using the iStartPos parameter.

\$writefile() returns a boolean value, which is true if the data was successfully written. After writing the data, the current position addresses the byte after the last byte written, or end of file.

FileOps Function Error Codes

The following errors are returned from the FileOps functions.

Constant	Code	Description
kFileOpsNoOperation	999	Operation not supported on this platform
kFileOpsUnknownError	998	Unknown error
kFileOpsOutOfMemory	12	Out of memory
kFileOpsParamError	1	Too few parameters passed
kFileOpsOK	0	No Error
kFileOpsDirFull	-33	File/Directory full
kFileOpsDiskFull	101206	Disk full
kFileOpsVolumeNotFound	-35	Specified volume does not exist
kFileOpsDiskIOError	-36	Disk I/O error
kFileOpsBadName	-37	Bad file name or volume name (perhaps zero-length)
kFileOpsFileNotOpen	101202	File not open
kFileOpsEndOfFile	101207	Logical end-of-file reached during read operation

Constant	Code	Description
kFileOpsPositionBeforeStart	-40	Attempt to position before the start of the file
kFileOpsTooManyFilesOpen	-42	Too many files open
kFileOpsFileNotFound	101201	File not found
kFileOpsHardwareVolumeLock	-44	Volume is locked by a hardware setting
kFileOpsFileLocked	101205	File is locked
kFileOpsSoftwareVolumeLock	-46	Volume is locked by a software flag
kFileOpsMoreFilesOpen	-47	One or more files are open
kFileOpsAlreadyExists	101215	A file with the specified name already exists
kFileOpsAlreadyWriteOpen	-49	Only one access path to a file can allow writing
kFileOpsNoDefaultVolume	-50	No default volume
kFileOpsVolumeNotOnline	-53	Volume not on-line
kFileOpsPermissionDenied	101203	Permission denied.
kFileOpsReadOnlyFile	-54	Read only file
kFileOpsVolumeAlreadyMounted	-55	Specified volume is already mounted and on-line
kFileOpsBadDrive	-56	No such drive number
kFileOpsInvalidFormat	-57	Volume lacks Macintosh-format directory
kFileOpsExternalSystemError	-58	External file system error
kFileOpsProblemDuringRename	-59	Problem during rename
kFileOpsBadMasterBlock	-60	Master directory block is bad; must re-initialize volume
kFileOpsCantOpenLockedFile	-61	Cannot open a locked file
kFileOpsDirectoryNotFound	-120	Directory not found
kFileOpsTooManyDirOpen	-121	Too many working directories open
kFileOpsCantMoveToOffspring	-122	Attempted to move into offspring
kFileOpsNonHFSOperation	-123	Attempt to do HFS operation on a non-HFS volume
kFileOpsInternalSystemError	-127	Internal file system error

flag()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

flag()

Description

Returns #F.

fld()

Function group	Execute on client	Platform(s)
<i>Field</i>	NO	All

Syntax

fld(string1[,string2],...)

Description

Returns the value of the field name given by concatenating or combining one or more *string* values.

Example

```
Calculate RATE1 as 10  
Calculate RATE2 as 15
```

```
Calculate lFieldValue as fld('RATE','1')  
# returns 10
```

```
Calculate lFieldValue as fld('RATE','2')  
# returns 15
```

fmtdatetime()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

```
fmtdatetime([newformat])
```

Description

Either sets #FDT to *newformat* and returns the previous value of #FDT, or if no parameter is supplied returns the current value of #FDT.

fmtdp()

Function group	Execute on client	Platform(s)
<i>Number</i>	NO	All

Syntax

```
fmtdp([newdps])
```

Description

Either sets #FDP to *newdps* and returns the previous value, or if no parameter is supplied returns the current value of #FDP.

fmtshortdate()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

```
fmtshortdate([newformat])
```

Description

Either sets #FD to *newformat* and returns the previous value of #FD, or if no parameter is supplied returns the current value of #FD.

fmtshorttime()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

fmtshorttime(*[newformat]*)

Description

Either sets #FT to *newformat* and returns the previous value of #FT, or if no parameter is supplied returns the current value of #FT.

fontlist()

Function group	Execute on client	Platform(s)
<i>Font handling</i>	NO	Windows, Linux

Syntax

fontlist(*listname*)

Description

Returns a list of fonts currently installed in your system, including the font name and type.

Note that fontlist() is deprecated. Use *FontOps.\$replistfonts()* or *FontOps.\$winlistfonts()* instead.

The *listname* parameter is any list field or variable. A return value of 0 indicates no fonts found or some error, otherwise 1 is returned indicating a list was built. You should define the following columns in your list field or variable.

Column 1	Column 2 (Optional)	Column 3 (Optional)
String variable to hold name of the font	Numeric variable to hold value (0..7) for type of font	String variable to hold platform(s) will be a combination of with "Fixed" or "Variable"

Example

```
Set current list lFontList
Define list {lFontName,lFontType,lFontDesc}
If fontlist(lFontList)<>0
  Redraw lists
End If
```

FontOps.\$replistfonts()

Function group	Execute on client	Platform(s)
<i>FontOps</i>	NO	All

Syntax

FontOps.\$replistfonts(*list*)

Description

Populates the specified *list* with the report fonts installed on your system, and indicates whether or not they are truetype.

The list must contain two columns, the first character type, the second boolean. The function returns zero for success, less than zero for failure. Having built the list you can search and manipulate the list using the standard list functions and methods.

Example

```
Do lFontList.$define(lFont,lTrueType)
Do FontOps.$replistfonts(lFontList)
# returns a list of report fonts

Do lFontList.$linecount() Returns lNumberOfFonts
# returns the total number of report fonts

Do tot(lFontList,lTrueType) Returns lNumberOfFonts
# returns the number of truetype fonts

Do totc(lFontList,lTrueType=kFalse) Returns lNumberOfFonts
# returns the number of non-truetype fonts
```

FontOps.\$reptextheight()

Function group	Execute on client	Platform(s)
FontOps	NO	All

Syntax

FontOps.\$reptextheight(font-name|font-table-index,point-size[,font-style,extra-points])

Description

Returns the height in inches/cms (depending on \$usecms preferences) of the specified report font. The units of the returned value are determined by \$root.\$prefs.\$usecms.

Example

```
Do FontOps.$reptextheight('Times',144) Returns lHeight
# returns 2.24 inches / 5.69 cms

Do FontOps.$reptextheight('Times',144,,24) Returns lHeight
# returns 2.57 inches / 6.54 cms
```

FontOps.\$reptextwidth()

Function group	Execute on client	Platform(s)
FontOps	NO	All

Syntax

FontOps.\$reptextwidth(string,font-name|font-table-index,point-size[,font-style])

Description

Returns the width in inches/cms (depending on \$usecms preference) required to draw the *string* using the specified report font. The units of the returned value are determined by \$root.\$prefs.\$usecms.

Example

```
Do FontOps.$reptextwidth('Hello WWW','Arial',24) Returns lWidth
# returns 1.83 inches / 4.66 cms

Do FontOps.$reptextwidth('Hello WWW','Arial',24,kBold+kItalic) Returns lWidth
# returns 1.85 inches / 4.71 cms

Do FontOps.$reptextwidth('Hello WWW',2,72,kBold+kItalic) Returns lWidth
# returns 5.40 inches / 13.81 cms
# note that Courier font is at position 2 in #WIRFONTS
```

FontOps.\$winlistfonts()

Function group	Execute on client	Platform(s)
FontOps	NO	All

Syntax

FontOps.\$winlistfonts(*list*)

Description

Populates the *list* with the window fonts installed on your system, and indicates whether or not they are truetype.

The list must contain two columns, the first character type, the second boolean. The function returns zero for success, less than zero for failure. Having built the list you can search and manipulate the list using the standard list functions and methods.

Example

```
Do lFontList.$define(lFont,lTrueType)
Do FontOps.$winlistfonts(lFontList)
# returns a list of windows fonts

Do lst(lFontList,1,lFont) Returns lFirstFont
# returns the first font in the list

Do lst(lFontList,lFontList.$linecount,lFont) Returns lLastFont
# returns the last font in the list
```

FontOps.\$wintextheight()

Function group	Execute on client	Platform(s)
FontOps	NO	All

Syntax

FontOps.\$wintextheight(font-name|font-table-index,point-size[,font-style,extra-points])

Description

Returns the height in screen units of the specified window font.

You specify the font using either the *font-name* or *font-table-index*. When called with a font table index, **\$wintextheight()** uses the window font system table of the current library which can contain up to 15 fonts numbered 1 to 15. You specify the *point-size* of the font, and you can include a *font-style* constant and a number of *extra-points*.

Example

```
Do FontOps.$wintextheight('Courier',72) Returns lHeight
# returns 100 under Windows
```

```
Do FontOps.$wintextheight('Courier',72,,24) Returns lHeight
# returns 132 under Windows
```

```
Do FontOps.$wintextheight(2,36) Returns lHeight
# returns 50 under Windows
# note that Courier font is at position 2 in #WIWFFONTS
```

FontOps.\$wintextwidth()

Function group	Execute on client	Platform(s)
FontOps	NO	All

Syntax

FontOps.\$wintextwidth(string,font-name|font-table-index,point-size[,font-style])

Description

Returns the width in screen units required to display the *string* using the specified window font.

You specify the font using either the *font-name* or *font-table-index*. When called with a font table index, **\$wintextwidth()** uses the window font system table of the current library which can contain up to 15 fonts numbered 1 to 15. You can include a *font-style* constant, or combination of styles.

Example

```
Do FontOps.$wintextwidth('Hello WWW','Courier',36) Returns lWidth
# returns 243
```

```
Do FontOps.$wintextwidth('Hello WWW','Courier',36,kBold+kItalic) Returns lWidth
# returns 252
```

```
Do FontOps.$wintextwidth('Hello WWW',5,36) Returns lWidth
# returns 240
# note that System font is at position 5 in #WIWFFONTS
```

format()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

format(format,variable)

Description

Applies *format* to *variable* and returns the resulting string. *format* must be a Boolean, date, number or text format: the function expects the type of the *format* to correspond to the data type of the *variable*.

formatchunk()

Function group	Execute on client	Platform(s)
<i>RESTful</i>	NO	All

Syntax

formatchunk(*data*)

Description

Formats *data* and returns a chunk suitable for sending to the client using chunked transfer encoding. *Data* can be character (which Omnis converts to UTF-8) or binary.

formathttpdate()

Function group	Execute on client	Platform(s)
<i>RESTful</i>	NO	All

Syntax

formathttpdate(*omnisdate*)

Description

Formats the Omnis date-time value *omnisdate* (assumed to be in UTC) as an HTTP date header value and returns the resulting string.

formatserversentevent()

Function group	Execute on client	Platform(s)
<i>RESTful</i>	NO	All

Syntax

formatserversentevent(fieldname,fielddata[,fieldname,fielddata]...)

Description

Formats data and returns a value suitable for sending as an event in text/event-stream content. Parameters can be character (which Omnis converts to UTF-8) or binary (UTF-8).

getdatetime()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

getdatetime()

Description

Returns the current system date and time.

getenv()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

getenv(*name*)

Description

Returns the value of the Omnis process environment variable with the specified *name*.

Example

```
# Get the executable search PATH environment variable  
Calculate lVar1 as getenv("PATH")
```

getfye()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

getfye()

Description

Returns the current date of the fiscal year end (note no argument).

Example

```
Calculate lDate as setfye('DEC 31')  
# set the fiscal year end to December 31  
  
Calculate lDate as getfye()  
# returns December 31st set by setfye()
```

getseed()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

`getseed()`

Description

Returns the current content of the seed as an integer number (note no argument).

Example

```
Calculate lResult as getseed()
```

getticks()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	YES	All

Syntax

`getticks()`

Description

Returns the number of ticks elapsed since system boot (for a JavaScript client executed method, since midnight on 1 Nov 2011). A tick is 1/60th of a second. The value can overflow and restart at zero.

getws()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

`getws()`

Description

Returns the day of week which is set as the beginning of the week (note no argument).

The day of the week is returned as one of the datepart constants: kSunday, kMonday, kTuesday, kWednesday, kThursday, kFriday, kSaturday.

Example

```
Calculate lDate as setws(kMonday)
# set the beginning of the week to Monday
```

```
Calculate lDate as getws()
# returns 13, the value of the constant kMonday
```

hexcolor()

Function group	Execute on client	Platform(s)
upto 35913 <i>General</i>	NO	All
asof 35914 <i>General</i>	YES	All

Syntax

hexcolor(*string*)

Description

Returns a color value from the passed hex *string*. The hex *string* format is RRGGBB or RRGGBBAA for alpha support.

hsla()

Function group	Execute on client	Platform(s)
upto 35913 <i>General</i>	NO	All
asof 35914 <i>General</i>	YES	All

Syntax

hsla(*hue,saturation,light[,alpha]*)

Description

Returns a color value from the supplied color components: *hue* range is 0-360; *saturation* range is 0-100; *light* range is 0-100; optional *alpha* range is 0-255.

iconidwithbadge()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

iconidwithbadge(*clcn,iCnt/cBadgeIcn[,kIconBadge...,iBadgeColor,iBadgeTextColor]*)

Description

Returns a formatted string for \$iconid. *clcn* is the main icon (an SVG icon), *iCnt/cBadgeIcn* control the badge. The *kIconBadge...* constants control the display of the badge, and *iBadgeColor* and *iBadgeTextColor* set the colors of the badge and text.

When an icon ID uses an SVG icon name, `iconidwithbadge()` allows you to append additional values to the SVG name to define a badge to be added to the main icon.

The parameters are:

Parameter	Description
<i>clcn</i>	the ID of the primary SVG icon for the object / toolbar object

Parameter	Description
<i>iCnt/cBadgeicn</i>	the count to be displayed on the badge, or the ID of a smaller secondary icon
<i>klconBadge</i>	klconBadgeAlignTop, klconBadgeAlignBottom, or the default is the position set by the OS, also klconB
<i>ibadgecolor</i>	the color of the badge, the default is kJSThemeColorSecondary
<i>ibadgetextcolor</i>	the color of the count, or secondary icon, the default is kJSThemeColorSecondaryText

The following lines of code set up icon badges for buttons:

```
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( 'tablet_mac', 9 ))
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( 'tablet_mac+32x32', 9 ))
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( 'tablet_mac', 99, 0, kDarkGreen, kWhite ))
```

Some Omnis objects used fixed icon sizes, such as menu items or tabbar tabs, therefore when applying a badge to these objects you cannot supply an icon size for the primary icon as the size will be fixed by the object, for example:

```
Do $imenu.NewMenu.$objs.Item.$iconid.$assign(iconidwithbadge( 'tablet_mac', 9 ))
```

When using `iconidwithbadge()` in a client-executed method, the SVG parameters must be URLs, which can be generated with `iconurl()` in server-executed code.

The default icon badge background colour is `kJSThemeColorSecondary`, while the count or secondary icon is `kJSThemeColorSecondaryText` (for window class controls the colors are the standard OS colors).

Badge Options

The constants **klconBadgeAlignTop** and **klconBadgeAlignBottom** can be used in the *klconBadge* parameter in `iconidwithbadge()` to specify the position of the badge. Omitting this or passing 0, Omnis will use the default position for the OS – by default, macOS will draw a badge at the top right of an icon, and Windows at the bottom right.

The constant **klconBadgeBackgroundHide** allows you to hide the default colored circle badge when used with a secondary icon. If the badge has a count and not an icon, the badge background is always drawn and this option ignored. For example:

```
$iconid.$assign(iconidwithbadge( 'tablet_mac', 'star', kBadgeIconHideBackground, kDefault, kRed ))
```

iconurl()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

iconurl(*iconid*)

Description

Returns a URL string from the supplied *iconid* (integer or string for SVG icons) suitable for use with JavaScript client controls that can display an icon. Note `iconurl()` must be executed on the server to obtain the correct URL for the icon.

You can add an icon size constant to the *iconid* to specify the required size, e.g. 'car+32x32', otherwise the icon size is chosen to fill the image area of the control, that is, some controls may restrict the size of icon allowed such as check boxes.

The server will pick the correct resolution for the icon, if they are available in the icon set (1x or 2x), based on the client resolution.

int()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

int(number)

Description

Returns the integer part of a *number*; it does not round to the nearest integer.

Example

```
Calculate lResult as int(23.1056)
# returns 23
```

```
Calculate lResult as int('-2.66')
# returns -2
```

```
Calculate lResult as abs(int(-1.999))
# returns 1
```

isclear()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

isclear(expression)

Description

Returns true if the *expression* has value NULL, zero (for numeric data types only) or empty. The function returns true for *empty* and *false* Boolean values.

isclient()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

isclient() no parameters

Description

Returns true if the code is currently running in JavaScript client-executed code.

iseven()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

iseven(integer)

Description

Returns true if *integer* is an even number.

isfontinstalled()

Function group	Execute on client	Platform(s)
<i>Font handling</i>	NO	Windows, Linux

Syntax

isfontinstalled(*fontname*)

Description

Returns a true or false value indicating whether the named font has been fully installed into your system.

The *fontname* argument can be a string literal, character field or variable with a maximum length of 255.

Example

```
If not(isfontinstalled('07Font'))
  OK message {Cannot run library without '07Font'}
End If
```

isheadless()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

isheadless()

Description

Returns true if Omnis is currently running as a headless server with no GUI.

isleopard()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

isleopard()

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

Returns true if the current operating system is Mac OS X Leopard (10.5) or later.

islion()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

islion()

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

Returns true if the current operating system is Mac OS X Lion (10.7) or later.

ismountainlion()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

ismountainlion()

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

Returns true if the current operating system is Mac OS X Mountain Lion (10.8) or later.

isnull()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

isnull(expression)

Description

Returns true if the *expression* has value NULL.

Also returns true if *fieldname*, a field in the current record, is null. A null value is one where no value has been entered and the field definition is **Can be null** without **Insert as Empty**.

Example

```
Calculate lVar1 as #NULL
Calculate lBoolean as isnull(lVar1)
# returns true because lVar1 is null
```

isnumber()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

isnumber(string[,decimal-char][,thousands-char])

Description

Returns kTrue if the specified *string* can be evaluated as a number; kFalse otherwise.

The optional parameters can be used to define the decimal and thousand separator. If the optional parameters are not specified the default separators are used, a '.' for the decimal and a ',' for the thousand.

Example

```
Calculate lStatus as isnumber(lString)
# lStatus is kTrue if lString can be evaluated as a number
```

iso8601toomnis()

Function group	Execute on client	Platform(s)
upto 35924 Date and Time	NO	All
asof 35925 Date and Time	YES	All

Syntax

iso8601toomnis(ciso8601,bneedtime,bHasTimeZone[,cerrtext])

Description

Converts an ISO8601 date/date-time string to Omnis date-time and returns result (in UTC if cISO8601 contains time zone). Returns NULL and cErrText if an error occurs.

The date string is in the form: YYYY-MM-DDTHH:MM:SS.FFFZ rounding FFF into the hundredths field appropriately. The representations in the date string are:

YYYY-MM-DD	Date: Year, month, day
T	Date time delimiter
HH:MM:SS.FFF	Time: Hours, minutes, seconds, fractions of a second

For more information about ISO8601 look up: https://en.wikipedia.org/wiki/ISO_8601

Note that for a RESTful service, you should always use time zones for input date time values, so you would always pass `bHasTimeZone` as `kTrue` to `iso8601toomnis()` if you are passing `bNeedTime` as `kTrue`.

isodd()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

isodd(integer)

Description

Returns true if *integer* is an odd number.

isoweek()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

isoweek(date)

Description

Returns the ISO 8601 standard week number for the week containing the specified *date*.

Example

```
Calculate lDate as isoweek(#D)
# returns the current ISO week number
```

isoweekstart()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

isoweekstart(year,week)

Description

Returns the date of day 1 of the specified ISO *week* number (1-53) of the specified *year*.

issnowleopard()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

issnowleopard()

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

Returns true if the current operating system is Mac OS X Snow Leopard (10.6) or later.

isunicode()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	YES	All

Syntax

isunicode()

Description

Always returns true because Omnis Studio 5.0 and later always supports Unicode.

isvista()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

isvista([[themed=kfalse]])

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For themed = kFalse, returns true if the current operating system is Windows Vista or later; for themed = kTrue, returns true if the current operating system is Windows Vista or later with a themed appearance.

iswindows10()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

`iswindows10([themed=kfalse])`

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For `themed = kFalse`, returns true if the current operating system is Windows 10 or later; for `themed = kTrue`, returns true if the current operating system is Windows 10 or later with a themed appearance.

iswindows11()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

`iswindows11([themed=kfalse])`

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For `themed = kFalse`, returns true if the current operating system is Windows 11 or later; for `themed = kTrue`, returns true if the current operating system is Windows 11 or later with a themed appearance.

iswindows7()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

`iswindows7([themed=kfalse])`

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For `themed = kFalse`, returns true if the current operating system is Windows 7 or later; for `themed = kTrue`, returns true if the current operating system is Windows 7 or later with a themed appearance.

iswindows8()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

`iswindows8([themed=kfalse])`

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For `themed = kFalse`, returns true if the current operating system is Windows 8 or later; for `themed = kTrue`, returns true if the current operating system is Windows 8 or later with a themed appearance.

iswindows81()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

iswindows81([`themed=kfalse`])

Description

Deprecated. Please note that as of Studio 10.1 revision 23638, individual *is...()* functions are deprecated in favour of the *systemversion()* function- which supports all recent operating system versions.

For `themed = kFalse`, returns true if the current operating system is Windows 8.1 or later; for `themed = kTrue`, returns true if the current operating system is Windows 8.1 or later with a themed appearance.

iswindowsserver()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

iswindowsserver()

Description

Returns true if the current operating system is a Windows Server operating system.

JavaObjs Library.\$addclass()

Function group	Execute on client	Platform(s)
<i>JavaObjs Library</i>	NO	All

Syntax

JavaObjs Library.\$addclass(group,classfilename,[classpath])

Description

Adds a class to the specified group constant.

JavaObjs Library.\$resetclasscache()

Function group	Execute on client	Platform(s)
JavaObjs Library	NO	All

Syntax

JavaObjs Library.\$resetclasscache()

Description

Deletes the Java objects class cache file, allowing the Java objects library to rebuild it the next time Omnis starts up; any classes added with \$addclass will be lost from the cache.

join()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

```
join(list[,delimiter=ktab,column=1,stripwhite=kfalse,quotechar=""])
```

Description

Returns a string of all *column* values in *list*, delimited by *delimiter*. Options *stripwhite* to strip leading and trailing whitespace, and/or *quotechar* to quote column values (escaping quotes as 2 quotes).

jst()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

```
jst(string1,number1[,string2,number2]...)
```

Description

Returns a string containing the specified *string* left or right justified with sufficient spaces added to make a total length specified by *number*.

The **jst()** function also includes concatenation. If *number* is negative the resulting *string* is right justified, if the *number* is positive the *string* is left justified; *number* must be in the range 1 to 999.

```
jst('This is left justified',30)
# gives
|This is left justified |
```

```
jst('This is right justified',-30)
# gives
| This is right justified|
```

When you define the columns for a list, **jst()** lets you fix the column width and using a non-proportional font the list columns will line up properly. For example, the calculation for a list containing the fields CODE and COMPANY could be

```
jst(CODE,7,COMPANY,25)
```

The **jst()** function also includes concatenation, for example:

```
jst(p1,p2,p3,p4,p5,p6,...)
# is the same as
con(jst(p1,p2),jst(p3,p4),jst(p5,p6),...)
```

However it has a limit of 100 parameters, but you can exceed this limit by using *Calculate CVAR1 as jst(CVAR2,CVAR3)* where CVAR2 has 99 items and CVAR3 has 99 items, and so on.

Formatting Strings Using jst()

The **jst()** function can take a string for the second argument instead of a number, that is

```
jst(string1,string1[,string2,string2]...)
```

This form of **jst()** formats the first *string1* argument in a way controlled by the second *string2* argument. The second argument consists of a series of formatting options which you can use separately or together. Each option is represented by one or more characters. The order of the various formatting options is not important but the case is.

^n	(caret) causes the data to be centered in the field n characters wide.jst('abc','^5') ## returns 'abc'jst(FIELD,'^25') ## as a list calculation will center the FIELD values in a column 25 characters wide
£	places a £ sign in front of the data. The data must be a number or an integer.jst(TOTAL,'£') ## returns '£12.12' if TOTAL = 12.12
\$	places a \$ sign in front of the data. The data must be a number or an integer.jst(TOTAL,'\$') ## returns '\$12.12' if TOTAL = 12.12
<	left justification, overriding the default set up in a report class. This is used by the Ad hoc report generator to control the justification of fields.
Pc	causes the part of the field not filled by the data to be filled by character c.jst('abc','-5P*') ## returns '**abc'
X	causes the data to be truncated if its length exceeds the field length. The default is not to truncate.jst('abcdef','4') ## returns 'abcdefjst('abcdef','4X') ## returns 'abcd'
U	causes the data to be converted to upper case.jst('this IS it','U') ## returns 'THIS IS IT'
L	causes the data to be converted to lower case. jst('THIS is IT','L') ## returns 'this is it'
C	causes the data to be capitalized.jst('this is it','C') ## returns 'This Is It'jst('THIS IS IT','C') ## returns 'This Is It'jst('this IS IT','C') ## returns 'This Is It'
Nnn	causes the data to be treated as a fixed decimal number with nn decimal places. If nn is not specified, a suitable number of decimal places is used.jst(0.235,'N') ## returns '0.235'jst(0.235,'N2') ## returns '0.24'
Fnn	causes the data to be treated as a floating decimal number with format specified by nn. The nn argument can be a positive or negative number and has the same meaning as described for the #FDP variable. If nn is not specified, it defaults to the current value of #FDP. jst(12.35,'-10F9') ## returns '12.35'jst(12.35,'-10F-9') ## returns '1.23500000e+001'jst(12.35,'-10F-3U') ## returns '1.24E+001'
D	causes the data to be treated as a date. The default formatting string is #FD, but you can specify a formatting string as described later. Note - JavaScript client-executed methods do not support the D option.jst('26/11/97','DC') ## returns '26 Nov 97' if #FD = 'D m Y'
DT	causes the data to be treated as a long date and time. The default formatting string is #FDT but you can specify a formatting string using the : (colon) argument as described below. Note - JavaScript client-executed methods do not support the DT option. jst(#D,'DT') ## returns '26 Nov 97 15:30' if #FDT is 'D m Y H:N'
T	causes the data to be treated as a time using the formatting string #FT. You can include a format string using the : (colon) argument as described below. Note - JavaScript client-executed methods do not support the T option.jst('0620','T') ## returns '06:20' if #FT = 'H:N'
A	displays a null value as 'NULL'.jst(Field1,'A') ## returns 'NULL' when Field1 is null
B	causes the data to be treated as Boolean.jst(1,'LB') ## returns 'yes'
E	applies to numbers only and turns on the 'Zero shown empty' attribute.jst(0,'N2') ## returns '0.00'jst(0,'N2E') ## returns " , (comma) applies to numbers only and turns on the 'Shown like 1,234' attribute.jst(1234,'N2') ## returns '1234.00'jst(1234,'N2,') ## returns '1,234.00'
((open bracket) applies to numbers only and turns on the 'Shown like (1234)' attribute.jst(-1234,'N2') ## returns '-1234.00'jst(-1234,'N2(!)') ## returns '(1234.00)'jst(1234,'N2(!)') ## returns '1234.00'

-) (close bracket) applies to numbers only and turns on the 'Shown like 1234-' attribute. `jst(-1234,'N2')` ## returns '1234.00' `-jst(1234,'N2')` ## returns '1234.00'
 - +(plus sign) applies to numbers only and causes positive numbers to be shown with a "+" sign. `jst(1234,'N2+')` ## returns '+1234.00' `+jst(1234,'N2+')` ## returns '1234.00'
 - :(colon) characters following a colon are interpreted as a formatting string. This must be the last option since all characters following it become part of the formatting string. The meaning of the formatting string depends on the type of the data.
-

The formatting string has a similar format to #FDT if the data is a date/time value, using the following characters:

Y	Year (99)	H	Hour (0..23)
y	Year (1999)	h	Hour (1..12)
C	Century (19)	N	Minutes
M	Month (06)	S	Seconds
m	Month (JUN)	s	Hundredths
n	Month (June)	A	AM/PM
D	Day (12)	V	Short day of week (Fri)
d	Day (12th)	E	Day of year (1-366)
W	Day of week (5)	G	Week of year (1-52)
w	Day of week (Friday)	F	Week of month (1-6)

For example:

```
jst(#D,'D:w, d n CY')
# returns 'Saturday, 29th November 1997'
cap(jst(#D,'D:V d n Y'))
# returns 'Sat 29th Nov 97'
```

The formatting string has a similar format to #FT if the data is a time value. #FT is used as the formatting string if a formatting string is not specified for a time.

```
jst(#T,'T:H-N') ## returns '14-07'
```

If the data is neither a date nor a time value, and if the formatting string contains an X, the data value is inserted at the position of the X to produce the data value.

```
jst(0,'BC:The answer is X!')
# returns 'The answer is No!'
```

The formatting string is concatenated to the left of the data value if the formatting string does *not* contain an X. The data value is left unchanged if a formatting string is not specified.

```
jst(12,'-7N2:$') ## returns '$12.00'
jst(12,'-7N2:£') ## returns '£12.00'
jst(12,'-8N2:DM') ## returns 'DM12.00'
```

lday()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

lday(datepart,date)

Description

Returns the date of the last day of the year, month, week or quarter in which the specified date falls. `datepart` can be `kYear`, `kQuarter`, `kMonth`, or `kWeek`.

The period is specified using one of the following `datepart` constants: `kYear`, `kQuarter`, `kMonth`, `kWeek`.

Example

```
Calculate lDate as lday(kWeek,#D)
# returns May 11 2003, assume #D is May 8, 2003
```

```
Calculate lDate as lday(kMonth,#D)
# returns May 31 2003, assume #D is May 8, 2003
```

left()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
left(string,n)
```

Description

Returns the substring comprising the first *n* characters of the *string*.

Example

```
Calculate lResult as left('Raining',4)
# returns 'Rain'
```

len()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
len(string)
```

Description

Returns the length of a *string* (number of characters).

Example

```
Calculate lResult as len('Hello there!')
# returns 12
```

```
Calculate lResult as len(abs(-10.25))
# returns 5 as it is the same as len(10.25)
```

```
Calculate lResult as len('Omnis')+20
# returns 25
```

list()

Function group	Execute on client	Platform(s)
<i>List</i>	YES	All

Syntax

list(row1[,row2]...)

Description

Returns a list from a number of *row* variables of identical structure, that is, the data type of each column in each row variable should match.

In the following example, one row in the list is created for each row variable passed. If the type of a particular column in the list does not match the type of a row variable column, **list()** tries to convert the row variable column to that of the list column, from number to string. If the column type cannot be converted the column is left blank.

Example

```
Set current list lList
Define list {lCol1,lCol2,lCol3}
Calculate lList as list(lRow1,lRow2,lRow3)
# returns a list from row variables, lRow1, lRow2 and lRow3
```

listenv()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

listenv()

Description

Returns a 2 column list of the Omnis process environment variables. Column 1 contains the variable name, and column 2 the variable value. The list is sorted by the variable name column (case insensitive).

Example

```
# Get the list of environment variables
# There is no need to define the list first; the returned list has 2 character columns, name and value
Calculate lList as listenv()
```

ln()

Function group	Execute on client	Platform(s)
<i>Logarithmic</i>	YES	All

Syntax

ln(number)

Description

Returns the log to base e (the natural logarithm) of a *number*; or -1e100 if *number* <= 0.

Example

```
Calculate lResult as ln(exp(0.5))
# returns 0.5
```

locale()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	YES	All

Syntax

locale([locfile=kfalse])

Description

Returns either the locale string for the user locale in which the program is running, or in the fat client if *locfile* is kTrue, the current localisation data file locale.

loctoutc()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

loctoutc(*datetime*)

Description

Converts the *datetime* (or time) from the local timezone to UTC (Coordinated Universal Time), and returns the result.

log()

Function group	Execute on client	Platform(s)
<i>Logarithmic</i>	YES	All

Syntax

log(*number*)

Description

Returns the log to base 10 of a *number*; or -1e100 if *number* <= 0.

Example

```
Calculate lResult as log(100)
# returns 2
```

```
Calculate lResult as log(0.001)
# returns -3
```

lookup()

Function group	Execute on client	Platform(s)
<i>Lookup</i>	NO	All

Syntax

```
lookup([refname,]searchvalue[,fieldnumber])
```

Description

Returns a field value from another data file *refname* (opened as a lookup file) using a *searchvalue*.

The *fieldnumber* argument specifies the particular field in the lookup file to be returned by the function. Each lookup file is opened using the *Open lookup file* command at which time a reference label is assigned to that lookup.

If you omit the third argument, the value of the second field is returned by default. If one argument is given, the default lookup file is used, that is, the lookup file which was opened without a reference label, or the first lookup file opened if all have labels.

If no exact match is found, an empty value is returned. All field types are returned, including pictures and long text.

Example

```
Open lookup file {City/Cities.df1/FCITY/1}
# Reference name is City, file class FCITY
```

```
OK message {CNAME is [lookup('City','FOS',2)]}
# Uses first field in FCITY as the index
```

low()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
low(string)
```

Description

Returns the lower case representation of a *string*. Any non-alphabetic characters in the strings are unaffected by *low()*.

Example

```
Calculate lString as low('DAVID')
# returns 'david'
```

```
Calculate lString as low('OrAcLe8')
# returns 'oracle8'
```

```
Calculate lString as low('1017')
# returns '1017'
```

```
Calculate lString as mid(low('PERIPHERAL'),3,3)
# returns 'rip'
```

lst()

Function group	Execute on client	Platform(s)
List	NO	All

Syntax

```
lst([[listname,]linenumber,]fieldname)
```

Description

Returns the value of *fieldname* from a line specified by *linenumber* in a list specified by *listname*. For client-executed methods *listname* must be specified.

The *fieldname* argument must be a field stored in the list, not a constant or expression. If *listname* is not specified the current list is used. If only the *fieldname* argument is specified the current line of the current list is used.

Example

```
Calculate lResult as lst(lList,23,lCol1)
# returns lCol1 field value stored at line 23 of list lList
```

```
Calculate lResult as lst(23,lCol1)
# returns lCol1 field value stored in line 23 of the current list
```

```
Calculate lResult as lst(lCol1)
# returns lCol1 field value stored at current line of current list
```

```
Calculate lResult as lst(lList,0,#LM)
# returns the maximum number of lines in list lMyList
```

max()

Function group	Execute on client	Platform(s)
Lookup	YES	All

Syntax

```
max(value1[,value2]...)
```


Description

Returns the maximum value from a list of values. The values should all be numbers when numeric comparison is used or all strings when string comparison is used.

Example

```
Calculate lResult as max(3,6,2,7)
# returns 7
```

```
Calculate lResult as max('dagger','dog','dig')
# returns 'dog'
```

maxc()

Function group	Execute on client	Platform(s)
<i>List</i>	NO	All

Syntax

maxc(listname,column[,ignore-nulls])

Description

Returns the maximum value for a list column specified by *listname* and *column*. **maxc()** can only be used with columns defined using variables so it cannot be used with lists defined from a SQL class.

If you set *ignore_nulls* to 1, null values are ignored and not counted. If you omit this parameter or it evaluates to zero, nulls are treated as zero values and are counted.

Example

```
Calculate lMax as maxc(lInventory,lQuantity)
# returns the maximum value in the lQuantity column of lInventory
```

mid()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

mid(string,position,length)

Description

Returns a substring of a specified *length*, starting at a specified *position*, from a larger *string*.

If *position* is less than 1, or greater than the length of the *string*, the result is an empty string. If *length* is greater than the maximum length of any substring of *string* starting at *position*, the returned substring will be the remainder of *string* starting at position. If you omit *length*, the remainder of the *string* from *position* is returned.

Example

```
Calculate lString as mid('Information',6,3)
# returns 'mat'
```

```
Calculate lString as int(mid(12.45,2,3))
# returns 2 as it is the same as int('2.4')
```

```
Calculate lString as mid('interaction',6,24)
# returns 'action'
```

min()

Function group	Execute on client	Platform(s)
<i>Lookup</i>	YES	All

Syntax

```
min(value1[,value2]...)
```

Description

Returns the minimum value from a list of values. The values should all be numbers when numeric comparison is used or all strings when string comparison is used.

Example

```
Calculate lResult as min(3,6,2,7)
# returns 2
```

```
Calculate lResult as min('cat','dog','apple')
# returns 'apple'
```

minc()

Function group	Execute on client	Platform(s)
<i>List</i>	NO	All

Syntax

```
minc(listname,column[,ignore-nulls])
```

Description

Returns the minimum value for a list column specified by *listname* and *column*. *minc()* can only be used with columns defined using variables so it cannot be used with lists defined from a SQL class.

If you set *ignore_nulls* to 1, null values are ignored and not counted. If you omit this parameter or it evaluates to zero, nulls are treated as zero values and are counted.

Example

```
Calculate lMin as minc(lInventory,lQuantity)
# returns the minimum value in the lQuantity column of lInventory
```

mod()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

mod(number1,number2)

Description

Returns the remainder of a number division, that is, when *number1* is divided by *number2* to produce a remainder; it is a true modulus function.

mod() checks for integer arguments, and if so uses integer rather than floating point operations to determine its result. The result is still returned as a number rather than an integer.

Example

```
Calculate lResult as mod(6,4)
```

```
# returns 2
```

```
Calculate lResult as mod(4,6)
```

```
# returns 4
```

```
Calculate lResult as mod(-5,-2)
```

```
# returns -1
```

mousedn()

Function group	Execute on client	Platform(s)
<i>Mouse</i>	NO	All

Syntax

mousedn()

Description

Returns true if the mouse button is held down, otherwise returns false (note no argument). This function returns a Boolean value describing the state of mouse button (the left-hand mouse button under Windows).

Note: This function is for desktop apps using Window classes only, not web or mobile apps.

mouseover()

Function group	Execute on client	Platform(s)
<i>Mouse</i>	NO	All

Syntax

mouseover(constant)

mouseover(constant[,tablefield])

Description

Note: This function is for desktop apps using Window classes only, not web or mobile apps.

Returns information about the mouse position defined by a predefined *constant* at the instant the function is evaluated.

Returns information about the mouse position defined by a mouseover *constant* (e.g. kMLine) when mouseover is called. If using kMItemref and *tablefield* is kTrue, enterable grid row fields are checked; in this case, the function will return a reference to the field under the mouse in a complex grid.

The function only works in an “open” user-defined window (not reports or searches). Moreover, it returns references only for fields and not background objects (text and graphic objects).

The mouse position is returned in a variety of ways depending on the constant you use. You can use the following constants:

Constant	Description
kMCharpos	returns the nth character in an edit field.
kMComplexGridRow	returns the row of a Complex grid the pointer is over
kMHorz	returns the horizontal position of the mouse relative to the topmost open user-defined window; if no user-defined window is open, returns the horizontal position of the mouse relative to the Omnis application window
kMItemref	returns a reference to the object under the mouse. The window instance itself can be returned as item 0 of the window.
kMLine	returns the line number for a list.
kMPLine	returns the line of the parent list the mouse is over.
kMScreenPos	returns a row with 2 columns, h and v, which are the horizontal and vertical position of the pointer (mouse) in screen coordinates, respectively
kMVert	returns the vertical position of the mouse relative to the topmost open user-defined window; if no user-defined window is open, returns the relative position to the Omnis application window

mouseup()

Function group	Execute on client	Platform(s)
<i>Mouse</i>	NO	All

Syntax

mouseup()

Description

Returns true if the mouse button is released after having been pressed, otherwise returns false (note no argument).

Note: This function is for desktop apps using Window classes only, not web or mobile apps.

msgcancelled()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

msgcancelled()

Description

Returns true if the user Cancel button is pressed on a message box.

For example, you can use this to distinguish between No and Cancel on a Yes/No message which both clear the flag.

Example

```
Yes/No message (Cancel button) {Do you want to proceed?}
If flag false
  If not(msgcancelled())
    # user chose No
  End If
Else
  # user chose Yes
End If
```

nam()

Function group	Execute on client	Platform(s)
<i>Field</i>	NO	All

Syntax

```
nam(fieldname[,doitems=ktrue])
```

Description

Returns the name of a field as a string; *fieldname* must be a field name or variable, not a constant or expression. If *doitems* is true, *nam()* follows item references.

Example

```
Calculate lFieldName as nam(CCODE)
# returns the string 'CCODE'
```

```
Calculate lFieldName as nam(#SUBFLD)
# returns the name of the subtotal field
```

natcmp()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

```
natcmp(value1,value2)
```

Description

Returns the result of comparing two values using the national sort ordering. Returns 0 if the strings are equal, 1 if *value1* > *value2*, and -1 if *value1* < *value2*.

Both values are converted to strings before the comparison is made. **natcmp()** uses the same rules for comparing the strings as it does for normal strings, except that it uses the national sort ordering.

Example

```
Calculate lResult as natcmp(lValue1,lValue2)
# returns 0 if values are equal
```

nday()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

nday(datepart,date)

Description

Returns the date of the day after the specified *date* when the *datepart* constant is set to *kDay*.

However, if one of the day of the week constants is used, this function returns the date of that day of the week following the specified *date*.

The datepart constants that you can use are: *kDay*, *kSunday*, *kMonday*, *kTuesday*, *kWednesday*, *kThursday*, *kFriday*, *kSaturday*.

Example

```
Calculate lDate as nday(kDay,#D)
# returns May 9 2003, assume #D is May 8, 2003
```

```
Calculate lDate as nday(kMonday,#D)
# returns May 12 2003, which is the next Monday after #D, assume #D is May 8, 2003
```

nfc()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	NO	All

Syntax

nfc(string)

Description

Carries out canonical decomposition followed by canonical composition on the *string* and returns the normalized string.

nfd()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	NO	All

Syntax

nfd(string)

Description

Carries out canonical decomposition on the *string* and returns the normalized string.

not()

Function group	Execute on client	Platform(s)
General	YES	All

Syntax

not(expression)

Description

Returns the logical Not of an *expression*.

All expressions in Omnis have a Boolean (truth) value. Firstly, non-zero numeric values (including negative values) are TRUE, zero values are FALSE. Secondly, string values are TRUE or FALSE depending on their numeric equivalent. String '1' has boolean value 1, therefore *not('1')* is 0. 'Bill' has boolean value 0, 'YES' has numeric value 0.

The numeric value of an expression that evaluates to true is 1, therefore *not(true)* is 0. Similarly, *not(false)* is 1.

You can also use **not()** to make method code more readable.

Example

```
Calculate lVar1 as not(2501)
# returns 0
```

```
Calculate lVar1 as not('Hello there!')
# is the same as not(0) which returns 1
```

```
Calculate lVar1 as not(31<45)
# is the same as not(true) which returns false
```

```
Do method ProcessList Returns Done
If not(Done)
    # do something
End If
# can be used to make code more readable
```

OIMAGE.\$getdimensions()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.\$getdimensions(*xImage*, &*iWidth*, &*iHeight* [,&*cErrorText*])

Description

Returns the dimensions in *iWidth* and *iHeight* of the image *xImage*. Returns Boolean true for success, or false and *cErrorText* if an error occurs.

xImage A binary variable containing the image.

iWidth and *iHeight* are 32 bit integer variables that receive the pixel Width and Height of the image.

cErrorText The error text returned from the function.

OIMAGE.\$makeqrcode()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.\$makeqrcode(vData, &xQrCode [,iFmt=kOIMAGEfmtPNG, iSizePNG=256, iECL=kOIMAGEerrHigh, iBorder=4, &cErrorText])

OIMAGE.\$makeqrcode(vData, &xQrCode [,iFmt=kOIMAGEfmtPNG, iSizePNG=256, iECL=kOIMAGEerrHigh, iBorder=4, &cErrorText, &wCenterImg])

Description

Makes a QR Code for the specified binary or character vData. Returns Boolean true and xImage for success, or false and cErrorText if an error occurs.

vData: Binary or character data to be represented by the QR Code. The method converts character data to UTF-8 before encoding it. The amount of data that can be encoded is limited, and varies according to iECL. See the QR Code standard.

xQrCode: Binary variable that receives the generated QR Code

iFmt: A kOIMAGEfmt... constant that specifies the format of the generated QR Code. kOIMAGEfmtPNG or kOIMAGEfmtSVG

iSizePNG: The width and height in pixels of the QR Code, when generating kOIMAGEfmtPNG. Must be between 32 and 262144 inclusive

iECL: A kOIMAGEerr... constant that specifies the error correction level supported by the generated QR Code:

1. **kOIMAGEerrLow**
The QR Code can tolerate about 7% erroneous codewords
2. **kOIMAGEerrMedium**
The QR Code can tolerate about 15% erroneous codewords
3. **kOIMAGEerrQuartile**
The QR Code can tolerate about 25% erroneous codewords
4. **kOIMAGEHigh**
The QR Code can tolerate about 30% erroneous codewords

iBorder: The number of border modules to surround the QR Code.

cErrorText: Receives the error text if an error occurs.

wCenterImg: Adds an icon image to the center of the generated QR code: see below.

Adding an image

The **wCenterImg** parameter controls how an icon can be overlaid after a QR code has been created; this only works for kOIMAGEfmtPNG output QR images (not for kOIMAGEfmtSVG). The parameter is a row variable with the following parameters:

```
row(cIconID, [ixSize=15, ixBorder=2, ixForeColor=kColorBlack, ixBackColor=kColorWhite])
```

By default, the row requires a single parameter, **cIconID**, which is the icon to be overlaid in the center of the QR code (this can be an SVG or PNG icon image). The second parameter **ixSize** controls the size of the icon, which is a percentage of the output QR image size; this defaults to 15% with a maximum value of 30% (a value over 30% may result in an error).

A border can be added to the overlaid icon using a third parameter **ixBorder** which defaults to 2 pixels. In addition, you can control the center icon area color, and icon color using parameters **ixForeColor** and **ixBackColor**, which default to black and white (to match the resulting QR image which is black and white).

For example, to add the account_box icon to the center of a QR code using the default size and border, use the following command:


```
Do OIMAGE.$makeqrcode(
    lText,iPicture,kOIMAGEfmtPNG,256,
    iErrorLevel,4,errText,row('account_box'))
```

To set the size of the overlaid icon to 25% of the QR image size, add a 5 pixel border, and to set the colors of the overlaid icon and border color, use the following command:

```
Do OIMAGE.$makeqrcode(
    lText,iPicture,kOIMAGEfmtPNG,256,
    iErrorLevel,4,errText,row('account_box',25,5,kWhite,kRed))
```

Note the account_box icon is a themed icon so you can set the icon color (but remember the output QR image must be a PNG, i.e. kOIMAGEfmtPNG).

OIMAGE.\$resize()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.\$resize(*xImage*, *iWidth*, *iHeight*, &*xNewImage* [,*wParams*=#NULL, &*cErrorText*])

Description

Resizes the image *xImage* to the dimensions supplied in *iWidth* by *iHeight*. Returns Boolean true and *xNewImage* for success, or false and *cErrorText* if an error occurs.

xImage and *xNewImage* are both binary variables. *xImage* must be a JPEG or PNG image. *xNewImage* has the same type as *xImage*. You can use the same binary variable for both parameters if you want to replace the original image.

iWidth and *iHeight* can both be greater than zero and less than or equal to 32000, meaning resize to exactly that size. Alternatively, only one of the new dimensions can be zero, meaning calculate the dimension with value zero using the aspect ratio of the input image.

wParams is an optional row variable of parameters. The following columns can be specified in *wParams*:

1. **sampler**

The sampling method to be used when resizing. Either kOIMAGEsamplerBilinear or kOIMAGEsamplerNearestNeighbour. Defaults to kOIMAGEsamplerBilinear if this column is not present, or if *wParams* is omitted.

2. **gray**

A Boolean that indicates if the new image is to be a grayscale image. Defaults to kFalse if this column is not present, or if *wParams* is omitted.

3. **quality**

If the input image type is JPEG this column contains the JPEG image quality (1 to 100) of the new image. Defaults to 80 if this column is not present, or if *wParams* is omitted.

cErrorText The error text returned from the function.

OIMAGE.\$transform()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.\$transform(*xImage*,&*xNewImage*[,*wParams*=#NULL,&*cErrorText*])

Description

Perform a transformation on the image *xImage*. Returns Boolean true and *xNewImage* for success, or false and *cErrorText* if an error occurs.

xImage and *xNewImage* are both binary variables. *xImage* must be a JPEG or PNG image. *xNewImage* has the same type as *xImage*. You can use the same binary variable for both parameters if you want to replace the original image.

wParams is an optional row variable of parameters. The following columns can be specified in *wParams*:

1. transform

The transformation to be performed, a kOIMAGEtransform... constant, see below. Defaults to kOIMAGEtransformRotate90 if this column is not present, or if *wParams* is omitted.

2. quality

If the input image type is JPEG this column contains the JPEG image quality (1 to 100) of the new image. Defaults to 80 if this column is not present, or if *wParams* is omitted.

Valid transformations are:

Constant	Description
kOIMAGEtransformRotate90	Rotates the image 90 degrees clockwise; the default
kOIMAGEtransformRotate180	Rotates the image 180 degrees
kOIMAGEtransformRotate270	Rotates the image 270 degrees clockwise
kOIMAGEtransformFlipLeftRight	Flips the image on the left to right axis
kOIMAGEtransformFlipTopBottom	Flips the image on the top to bottom axis
kOIMAGEtransformTranspose	Transposes the image, i.e. flips the image along a 45 degree axis, from top-left corner to bottom-right corner

cErrorText The error text returned from the function.

OJSON.\$arrayarraytolist()

Function group	Execute on client	Platform(s)
OJSON	NO	All

Syntax

OJSON.\$arrayarraytolist(*vData*[,&*cErrorText*])

Description

Parses *vData* (binary (UTF8/16/32) or character). *vData* must be a JSON array of arrays containing members with simple types. Returns a list representing the JSON. Returns NULL and *cErrorText* for an error.

This method only works with simple types as array members. *\$arrayarraytolist* requires that each JSON array must have the same number of elements, and each JSON array member at a particular index must be of the same data type.

\$arrayarraytolist allows integer and numeric values in the same column (resulting in a numeric column).

OJSON.\$couldbearray()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$couldbearray(vData)

Description

Returns true if vData (either binary (UTF8/16/32) or character) could possibly be a JSON array because its first character is [.

OJSON.\$couldbeobject()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$couldbeobject(vData)

Description

Returns true if vData (either binary (UTF8/16/32) or character) could possibly be a JSON object because its first character is {.

OJSON.\$formatjson()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$formatjson(vData)

Description

Parses the JSON in vData (either binary (UTF8/16/32) or character) and returns a formatted representation (or error message if parsing fails) suitable for use in a multi-line entry control.

OJSON.\$jsontolistorow()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$jsontolistorow(vData,[&cErrorText])

Description

Parses the JSON array or object in vData (either binary (UTF8/16/32) or character) and returns a row or a list representing the JSON.Returns NULL and cErrorText if an error occurs.

OJSON.\$listorrowtojson()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$listorrowtojson(vListOrRow,[iEncoding=kUniTypeUTF8,&cErrorText])

Description

Writes the list or row and returns JSON with the specified encoding (UTF8,UTF16BE/LE,UTF32BE/LE or Character).Returns NULL and cErrorText if an error occurs.

OJSON.\$listtoarrayarray()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$listtoarrayarray(lList[,iEncoding=kUniTypeUTF8,&cErrorText])

Description

Writes a list with simple columns to an array of arrays; returns the JSON with specified encoding (UTF8, UTF16BE/LE, UTF32BE/LE or Character). Returns NULL and cErrorText for an error.

This method only works with simple types as array members.

OJSON.\$listtoobjectarray()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$listtoobjectarray(lList[,iEncoding=kUniTypeUTF8,&cErrorText])

Description

Writes a list with simple columns to an array of objects; returns JSON with specified encoding (UTF8,UTF16BE/LE,UTF32BE/LE or Character). Returns NULL and cErrorText for an error.

OJSON.\$objectarraytolist()

Function group	Execute on client	Platform(s)
<i>OJSON</i>	NO	All

Syntax

OJSON.\$objectarraytolist(vData[,&cErrorText])

Description

Parses vData (binary (UTF8/16/32) or character). vData must be a JSON array of objects containing members with simple types. Returns a list representing JSON. Returns NULL and cErrorText for an error.

OmnisiCalendar.\$createcomponent()

Function group	Execute on client	Platform(s)
<i>OmnisiCalendar</i>	NO	All

Syntax

OmnisiCalendar.\$createcomponent(iType)

Description

Creates a new iCalendar Component object using one of the kICalendarComponent... constants. Returns an iCalendar component object.

iType: A kICalendarComponent... constant to specify the Component type.

OmnisiCalendar.\$createrow()

Function group	Execute on client	Platform(s)
<i>OmnisiCalendar</i>	NO	All

Syntax

OmnisiCalendar.\$createrow(iType)

Description

Returns a row which can be used to add or update certain Properties. iType can be one of the kICalendarRowType... constants.

iType: A kICalendarRowType... constant to specify the type of row.

OmnisIcn Library.\$getmask()

Function group	Execute on client	Platform(s)
<i>OmnisIcn Library</i>	NO	All

Syntax

OmnisIcn Library.\$getmask(iconid[,noscale])

Description

Returns a picture variable containing the mask for the icon identified by *iconid*.

When noscale is kTrue, a standard resolution image is returned. The default is kFalse, when the parameter is omitted, which will return an image scaled to match the resolution of the current or main display.

Example

```
Calculate lPicture as OmnisIcn Library.$getmask(2003)
```

OmnisIcn Library.\$getpict()

Function group	Execute on client	Platform(s)
<i>OmnisIcn Library</i>	NO	All

Syntax

OmnisIcn Library.\$getpict(*iconid*,*backgroundcolor*[,*noscale*])

Description

Returns a picture variable containing the icon identified by *iconid*, drawn on the specified *backgroundcolor*.

When *noscale* is *kTrue*, a standard resolution image is returned. The default is *kFalse*, when the parameter is omitted, which will return an image scaled to match the resolution of the current or main display.

Example

```
Calculate lPicture as OmnisIcn Library.$getpict(2003,kRed)
```

Omnis PDF Device.\$encrypt()

Function group	Execute on client	Platform(s)
<i>Omnis PDF Device</i>	NO	All

Syntax

Omnis PDF Device.\$encrypt(*cUserPwdASCII*[,*cOwnerPwdASCII*=",*bCanPrint*=*kTrue*,*bCanModify*=*kFalse*,*bCanCopy*=*kTrue*,*bCanAnnotate*=*kFalse*])

Description

Specifies how PDF documents created by the current task will be encrypted. Empty user password disables all encryption options.

Omnis PDF Device.\$setdocinfo()

Function group	Execute on client	Platform(s)
<i>Omnis PDF Device</i>	NO	All

Syntax

Omnis PDF Device.\$setdocinfo(*cAuthor*[,*cTitle*=",*cSubject*="])

Description

Sets the document information properties for PDF documents created by the current task.

Omnis PDF Device.\$settemp()

Function group	Execute on client	Platform(s)
<i>Omnis PDF Device</i>	NO	All

Syntax

Omnis PDF Device.\$settemp(*bTemp*[,*iTimeout*])

Description

bTemp false means use \$prefs.\$reportfile.*bTemp* true means next PDF is output to temp storage and is valid for *iTimeout* minutes; returns id to use with PDF \$clientcommands.\$settemp can be used for each task instance.

omnistoiso8601()

Function group	Execute on client	Platform(s)
upto 35924 Date and Time	NO	All
asof 35925 Date and Time	YES	All

Syntax

omnistoiso8601(*domnisdatetime*,*bneedtime*[,*cerrtext*, *bMillis=kFalse*])

Description

Converts dOmnisDateTime (in UTC) to ISO8601 date/date-time (depends on bNeedTime) and returns it. Returns NULL and cErrText on error. Rounds down hundredths if bMillis is false.

Hundredths are always rounded down to the nearest tenth. omnistoiso8601() always outputs the timezone using the "Z" UTC time indicator.

ONOTIFY.\$removebadge()

Function group	Execute on client	Platform(s)
<i>ONOTIFY</i>	NO	All

Syntax

ONOTIFY.\$removebadge([,&cErrorText])

Description

Removes the badge from the application icon. Returns Boolean true for success, or false (and sets *cErrorText*) if an error occurs.

ONOTIFY.\$removelocal()

Function group	Execute on client	Platform(s)
<i>ONOTIFY</i>	NO	All

Syntax

ONOTIFY.\$removelocal([*VIDs*,&cErrorText])

Description

Removes local notifications with id(s) specified by *vIDs* which is a single character id, or a single column list of ids (remove all local notifications if *vIDs* is empty or omitted). Returns Boolean true for success, or false (and *cErrorText*) if an error occurs.

ONOTIFY.\$sendlocal()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.\$sendlocal(*cTitle,cMessage,vImage,iAction,wUserInfo,[iDelay=0,&cErrorText]*)

Description

Sends a local operating system notification. Returns character notification id for success or returns #NULL (and *cErrorText*) if an error occurs.

If the call to *\$sendlocal*() succeeds, it returns a character string. This is a string that uniquely identifies the notification. You can use this string to remove the notification from the system Notification Center using *\$removelocal*, if for example the notification is no longer relevant.

The parameters are as follows:

Parameter	Description
<i>cTitle</i>	The title of the notification. Some text, displayed in bold font above the main notification text. The operating system will truncate this if it is too long. Windows allows this to occupy two lines, if you separate the lines using either <i>kCr, lLf</i> or <i>kCr kLf</i> . macOS only allows a single line
<i>cMessage</i>	The text of the notification. This is the main notification message, displayed in a plain font. The operating system will truncate this if it occupies more than 4 lines, either due to word wrapping, or the presence of newline characters (<i>kCr, lLf</i> or <i>kCr kLf</i>)
<i>vImage</i>	Image(s) to be displayed with the notification. See the 'Specifying Images' section
<i>iAction</i>	A value that specifies up to 2 optional actions that are to be included in the notification; on Windows, this is via one or two buttons; on macOS, this is either via a button for a single action, or via an options popup for two actions. A 'Specifying Actions' section
<i>wUserInfo</i>	A row containing user information that is passed to the <i>\$localnotify</i> () method when the user clicks on the notification or a notification action. It must be possible to convert <i>\$userinfo</i> to JSON. See section 'Handling Notification Clicks'
<i>iDelay</i>	The delay in seconds between the call to <i>\$sendlocal</i> () and the notification being delivered (optional). Omnis can quit before the notification is delivered, as the operating system takes care of deferred delivery
<i>cErrorText</i>	A character variable that receives text describing an error if <i>\$sendlocal</i> () fails

ONOTIFY.\$setbadgecount()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.\$setbadgecount(*iCount*[,&*cErrorText*,*iBadgeColor*,*iBadgeTextColor*])

Description

Sets the badge on the application icon to have the value *iCount*. Returns Boolean true for success, or false (and *cErrorText*) if an error occurs.

The parameters are as follows:

Parameter	Description
<i>iCount</i>	The count to display as the badge. Must be greater than zero. When running on Windows, a value greater than 99 is displayed as 99+.
<i>cErrorText</i>	A character variable that receives text describing an error, if \$setbadgecount() fails
<i>iBadgeColor</i>	Windows only. The background color of the count badge. Defaults to styledbadgebackgroundcolor in the "system" section of appearance.json.
<i>iBadgeTextColor</i>	Windows only. The text color of the count badge. Defaults to styledbadgetextcolor in the "system" section of appearance.json.

ONOTIFY.\$setbadgeicon()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.\$setbadgeicon(*vIconId*[,&*cErrorText*,*iBadgeColor*])

Description

Note this is available on Windows only. Sets the badge on the application icon to be the specified icon *vIconId*. Returns Boolean true for success, or false (and sets *cErrorText*) if an error occurs.

The parameters are as follows:

Parameter	Description
<i>vIconId</i>	The icon id of the icon to display as the badge. The size component is ignored, as badges are always 16x16.
<i>cErrorText</i>	A character variable that receives text describing an error, if \$setbadgeicon() fails
<i>iBadgeColor</i>	The color to be applied to the themed SVG; only applies if the icon is a themed SVG. Default is kColorHighlight.

ord()

Function group	Execute on client	Platform(s)
Unicode	NO	All

Syntax

`ord(integer[,locale,thousands=kfalse])`

Description

Returns string comprising positive *integer* with ordinal suffix for supplied *locale*, or current language if *locale* parameter is omitted or empty. Optionally includes ICU *thousands* separators.

OW3.\$computename()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

`OW3.$computename(bFormat=kFalse)`

Description

Returns the name of the current computer or an empty string if the name could not be obtained. If *bFormat* is true, `$computename` attempts to format the string to make it suitable for an end-user, by removing a '.local' suffix, replacing all - characters with space, and capitalizing each word.

OW3.\$escapeuritext()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

`OW3.$escapeuritext(cTextToEscape)`

Description

Returns the escaped form of *cTextToEscape*. All characters except a-z, A-Z, 0-9, - (hyphen), . (period), _ (underscore), and ~ (tilde) are escaped as %hh. Unicode characters are the escaped form of their UTF-8 representation.

OW3.\$gethardwareid()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

`OW3.$gethardwareid()`

Description

Returns a string ID that identifies the hardware on which Omnis is currently running. This function replaces sys(227) which has been removed.

OW3.\$makeuri()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$makeuri(cURIBase,IParameters)

Description

Returns the URI formed by adding the parameters in IParameters to cURIBase, escaping parameter names and values as necessary. IParameters is a list: column 1 is the parameter name and column 2 the parameter value.

OW3.\$makeuuid()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$makeuuid([bIncludeMinusSeparators=kTrue])

Description

Returns a new UUID as a string. bIncludeMinusSeparators affects the format of the returned string: if true, the returned UUID string is 36 characters long and includes – (hyphen) separators (so it has the standard 8-4-4-4-12 format); if false, the returned UUID string is 32 characters long and has no separators.

OW3.\$parserfc3339()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$parserfc3339(cRfc3339[,bUTC=kTrue,&iOffset,&cErrorText])

Description

Parses a date and time value conforming to RFC3339 and returns an Omnis date-time value and optionally the time zone offset in minutes. Returns #NULL if the string cannot be parsed.

The parameters are:

Parameter	Description
cRfc3339	a date and time string conforming to RFC3339
bUTC	If true, the returned date-time value is in UTC rather than the local time zone of the RFC3339 date-time value

Parameter	Description
<i>iOffset</i>	If the RFC3339 date and time string is parsed successfully this receives the time zone offset in minutes
<i>cErrorText</i>	If supplied, receives text describing the error that caused \$parserfc3339 to return #NULL

OW3.\$splitmultipart()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$splitmultipart(*cContentType*, *xContent*, &*MIMEList* [*iDefCharSet*=kUniTypeUTF8, &*cErrorText*])

Description

Splits MIME-encoded multi-part *xContent* into *MIMEList*. *cContentType* must include a boundary parameter. Returns true if successful. The parameters are:

Parameter	Description
<i>cContentType</i>	The content type header (must contain a boundary parameter)
<i>xContent</i>	The binary content to split
<i>MIMEList</i>	Receives the MIME list created by splitting the MIME content. See the documentation for the MailSplit command to see how a MIME list is structured; however note that the charset in the OW3 MIME list is a kUniType... constant
<i>iDefCharSet</i>	The default character set used to convert character data when there is no charset specified for a MIME text body part. A kUniType... constant (not Character/Auto/Binary)
& <i>cErrorText</i>	If supplied, receives text describing the error that caused \$splitmultipart to return false

The MIME list (for this call and for the other OW3 calls that generate a MIME list) now contains an additional column named bodypart-headers. This is a row containing a column for each non-empty header present for the body part. In addition, it has a column named "name" which contains the content-disposition header name parameter. All header names are normalized in the same way as those passed to RESTful services, that is, lower-case with any - characters removed.

OW3.\$totpgenerate()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$totpgenerate(*xSharedSecretKey*, *iTimeStep*, *iDigits*, &*iTOTP* [, &*cErrorText*, *iHashType*=kOW3hashSHA1])

Description

Generates a Time-based One-Time Password in iTOTP using the TOTP algorithm. Returns true if successful.

xSharedSecretKey (Binary) The shared secret key, length must be between 16 and 256 bytes inclusive.

iTimeStep (Integer 32 bit) The time step in seconds, must be between 1 and 3600 inclusive.

iDigits (Integer 32 bit) The number of digits in the TOTP, must be between 6 and 8 inclusive.

iTOTP (Integer 32 bit) Receives the generated TOTP.

cErrorText (Character 100000000) If supplied, receives text describing the error that caused \$totpgenerate or \$totpvalidate to return false.

iHashType (Integer 32 bit, default is kOW3hashSHA1) The type of hash to use, a kOW3hash... constant.

See OW3.\$totpvalidate().

OW3.\$totpvalidate()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$totpvalidate(*xSharedSecretKey*,*iTimeStep*,*iDigits*,*iTOTP*[,&*cErrorText*,*iHashType*=kOW3hashSHA1,*iStepsBefore*=2,*iStepsAfter*=1])

Description

Validates the Time-based One-Time Password in iTOTP using the TOTP algorithm. Returns true if iTOTP is a valid TOTP.

xSharedSecretKey (Binary) The shared secret key, length must be between 16 and 256 bytes inclusive.

iTimeStep (Integer 32 bit) The time step in seconds, must be between 1 and 3600 inclusive.

iDigits (Integer 32 bit) The number of digits in the TOTP, must be between 6 and 8 inclusive.

iTOTP (Integer 32 bit) The TOTP that is to be validated.

cErrorText (Character 100000000) If supplied, receives text describing the error that caused \$totpgenerate or \$totpvalidate to return false.

iHashType (Integer 32 bit, default is kOW3hashSHA1) The type of hash to use, a kOW3hash... constant.

iStepsBefore (Integer 32 bit, default is 2) The number of time steps before the step for the current time that can be checked when validating the supplied TOTP, must be between 1 and 20 inclusive.

iStepsAfter (Integer 32 bit, default is 1) The number of time steps after the step for the current time that can be checked when validating the supplied TOTP, must be between 1 and 20 inclusive.

See OW3.\$totpgenerate().

OW3.\$unescapeuritext()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.\$unescapeuritext(*cTextToUnescape*)

Description

Returns the unescaped form of `cTextToUnescape` by unescaping `%hh` sequences and converting the resulting UTF-8 string to character. Returns NULL if the string is not a valid escaped value.

OXML.\$base64decode()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$base64decode(*cData*,&*cErrorText*)

Description

Decodes the BASE64 encoded Binary data in *cData* and returns the result, or NULL and *cErrorText* if an error occurs.

OXML.\$base64encode()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$base64encode(*xData*,&*cErrorText*)

Description

Encodes the data using BASE64 and returns the result as a Binary, or NULL and *cErrorText* if an error occurs.

OXML.\$formatbinaschar()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$formatbinaschar(*xData*[,*iBytesPerRow*=16,*iMaxLength*=0])

Description

Formats at most the first *iMaxLength* bytes of binary data *xData* into character data suitable for display in a multi-line entry field.

OXML.\$maybexml()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$maybeXML(*xData*)

Description

Returns true if the binary data is likely to be XML (it starts with <?xml, in one of the encodings supported by the parser).

OXML.\$md5hexdigest()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$md5hexdigest(*cData*)

Description

Generates an MD5 hex digest based on the supplied character data.

OXML.\$striphttpheader()

Function group	Execute on client	Platform(s)
<i>OXML</i>	NO	All

Syntax

OXML.\$striphttpheader(&*cData*)

Description

Strips the HTTP header from the argument.

parsehttpauth()

Function group	Execute on client	Platform(s)
<i>RESTful</i>	NO	All

Syntax

parsehttpauth(*auth*)

Description

Parses the HTTP Authorization header value *auth* and returns a row variable containing the extracted information. Column 1 of the returned row (named *scheme*) is the scheme (e.g. basic). Other columns are scheme dependent.

parsehttpdate()

Function group	Execute on client	Platform(s)
<i>RESTful</i>	NO	All

Syntax

parsehttpdate(*httpdate*)

Description

Parses date value *httpDate* in HTTP header format (e.g. Sun, 06 Nov 1994 08:49:37 GMT) and returns an Omnis date-time value (in UTC) or NULL if the value cannot be parsed successfully.

pathsep()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

pathsep()

Description

Returns the pathname separator for the current OS.

Example

```
Calculate lPathSep as pathsep()  
# returns '\' on Windows  
# returns '/' on macOS  
# returns '/' on Linux
```

pday()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

pday(datepart,date)

Description

Returns the date of the day before the specified *date* when the *datepart* constant is set to *kDay*.

However, if one of the day of the week constants is used, this function returns the date of that day of the week preceding the specified *date*.

The datepart constants that you can use are: *kDay*, *kSunday*, *kMonday*, *kTuesday*, *kWednesday*, *kThursday*, *kFriday*, *kSaturday*.

Example

```
Calculate lDate as pday(kDay,#D)  
# returns May 7 2003, assume #D is May 8, 2003
```

```
Calculate lDate as pday(kThursday,#D)  
# returns May 1 2003, which is the Thursday before #D, assume #D is May 8, 2003
```

pick()

Function group	Execute on client	Platform(s)
<i>Lookup</i>	YES	All

Syntax

pick(number,value0,value1[,value2]...)

Description

Selects an item from a list of *values* (strings or numbers) using *number* as a zero-based index and returns the result.

The *number* argument is rounded to an integer and used to select the item. *value0* is returned if the result is 0, *value1* if the result is 1, *value2* if the result is 2, and so on. If the *number* is less than zero or greater than the number of values in the list, an empty value is returned. Note the list of values can be a mixture of string and numeric values.

Example

```
Calculate lResult as pick(lVar1,123,'ABC','abc')
# returns 123 if lVar1 evaluates to 0
```

```
Calculate lResult as pick(lVar2,2200,4500,6800)
# returns 6800 if lVar2 evaluates to 2
```

```
Calculate lResult as pick(lVar3,'one',2,'three')
# returns empty if lVar3 evaluates to 5
```

pictconvfrom()

Function group	Execute on client	Platform(s)
<i>Picture</i>	NO	All

Syntax

pictconvfrom(format,binary-raw-data)

Description

pictconvfrom() takes the *binary-raw-data* in the specified *format* and returns a value suitable for use in relevant Omnis fields. Typically this means that pictconvfrom() adds header information to raw data.

Valid values for the format are as follows:

- **CS24**
Omnis color shared picture format (16 million colors), including the internal Omnis header.
- **PNG**
PNG (Portable Network Graphics) format (Raw, as written on disk)
- **JPEG**
JPEG format (Raw, as written on disk)

In addition, it is possible to add further formats using the external component interface.

The following example reads a JPEG file from disk and displays it.

Example

```
ReadBinFile ("c:\myfile.jpg",lJpegData)
Do pictconvfrom("jpeg",lJpegData) Returns lJpegData
# converts the JPEG data lJpegData into a suitable value for Omnis fields
```

pictconvto()

Function group	Execute on client	Platform(s)
<i>Picture</i>	NO	All

Syntax

pictconvto(source-format,source-data,dest-format)

Description

Converts the binary *source-data* (with or without our internal header) from the *source-format* to the *dest-format*,and returns the resulting binary value.

On macOS, conversion supports all bitmap data formats, including PNG, TIFF, BMP, JPEG, and GIF. Where a source data format is empty, an attempt will be made to convert from any supported bitmap image representation. In this case, the data must be in raw format with no Omnis header.

See *pictconvfrom()* for details of possible formats.

The following example converts the PNG data in lPngData to JPEG.

Example

```
Do pictconvto("png",lPngData,"jpeg") Returns lJpegData
# converts the PNG data in lPcxData to JPEG
```

pictconvtypes()

Function group	Execute on client	Platform(s)
<i>Picture</i>	NO	All

Syntax

pictconvtypes()

Description

Returns a single column list, which contains all the picture conversion format types registered with Omnis.

Example

```
Calculate lTypes as pictconvtypes()
# returns a list containing all the picture conversion types
```

pictformat()

Function group	Execute on client	Platform(s)
<i>Picture</i>	NO	All

Syntax

pictformat(*source-data*)

Description

Returns a character string which indicates the picture format of the binary *source-data*. See *pictconvfrom()* for details of possible formats.

Note that there are certain formats that **pictformat()** cannot recognize (for example, OLE data, GIF), and in these cases, **pictformat()** returns an empty string.

Example

```
Calculate lFormat as pictformat(lJpegData)
# returns "JPEG"
```

pictsize()

Function group	Execute on client	Platform(s)
<i>Picture</i>	NO	All

Syntax

pictsize(*source-data,width,height*)

Description

Sets *width* and *height* to the width and height of the picture stored in the supplied binary or picture *source-data*.

If the function fails, for example because it cannot recognize the format of the data, then width and height will be zero.

Example

```
Do pictsize(lJpegData,lWidth,lHeight)
# returns the width and height of lJpegData
```

platform()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

platform()

Description

Returns the OS code for the current Omnis executable.

Example

```
Calculate lPlatform as platform()
# returns 'N' on Windows NT/2000/2003/XP/Vista/Win7/8/10
# returns 'X' on macOS
# returns 'U' on Linux
# returns 'W' on Windows 95/98/ME ## deprecated
```

PortProfile.\$getportnames()

Function group	Execute on client	Platform(s)
<i>PortProfile</i>	NO	All

Syntax

PortProfile.\$getportnames()*(list-of-names)*

Description

Populates *list-of-names* with a list of all available ports on the current platform.

Example

```
Do PortProfile.$getportnames(lPorts)
```

PortProfile.\$getprofilenames()

Function group	Execute on client	Platform(s)
<i>PortProfile</i>	NO	All

Syntax

PortProfile.\$getprofilenames()*(list-of-names)*

Description

Populates *list-of-names* with a list of all profile names currently defined.

Example

```
Do PortProfile.$getprofilenames(lPorts)
```

pos()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

pos()*(substring,string)*

Description

Returns the position of *substring* in *string*, or zero if *substring* is not contained in *string*.

The *substring* must be contained within *string* in its entirety for the returned value to be non-zero. Also, the comparison is case sensitive and only the first occurrence of *substring* is returned (see third example).

Example

```
Calculate lResult as pos('Mouse','Mickey Mouse')
# returns 8
```

```
Calculate lResult as pos('mouse','Mickey Mouse')
# return 0, note case
```

```
Calculate lResult as pos(' ', 'R S W Smith')
# returns 2, that is the position of the first space character
```

```
# you can strip the extension from a filename using mid() and pos() combined
If pos('.',lFileName) ## if lFileName contains a dot
  Calculate lFileName as mid(lFileName,1,pos('.',lFileName)-1)
End If
```

printf()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	macOS and Linux

Syntax

```
printf(string[,newline=ktrue])
```

Description

Writes the string to standard output followed by a newline character if required (the function is ignored on Windows. Executes on macOS and Linux only).

putenv()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

```
putenv(name,value)
```

Description

Sets the Omnis process environment variable with the specified *name* to the specified *value*; creates a new environment variable if necessary; returns Boolean true for success, false for failure.

Example

```
# Set the environment variable MYVAR to the value MYVALUE
Do putenv("MYVAR","MYVALUE")
```

pwr()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

pwr(number,power)

Description

Returns the result of raising a *number* to a *power*.

Example

```
Calculate lResult as pwr(2,5)
# returns 32
```

```
Calculate lResult as pwr(21.37,0.831)
# returns 12.74 approx
```

```
Calculate lResult as int(pwr(1.105,5))
# is the same as int(1.65) which returns 1
```

rand()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

rand()

Description

Returns a random real number in the range $0.0 < \text{number} < 1.0$, inclusive (note no argument).

Example

```
Calculate lResult as rand()
```

randintrng()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

randintrng(number1,number2)

Description

Returns a random integer between *number1* and *number2* inclusive.

Example

Calculate `lResult` as `randintrng(25,50)`

randrealrng()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

`randrealrng(number1,number2)`

Description

Returns a random real number between *number1* and *number2* inclusive.

Example

Calculate `lResult` as `randrealrng(25,50)`

rcedit.\$setapplicationmanifest()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

`rcedit.$setApplicationManifest(cFileName, cManifestPath)`

Description

Sets manifest from file path in *cManifestPath* to exe/dll in *cFileName*. Returns `kTrue` if operation succeeded, else `kFalse`.

rcedit.\$setfileversion()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

`rcedit.$setFileVersion(cFileName, cFileVersion)`

Description

Sets file version to value from *cFileVersion* to exe/dll in *cFileName*. Returns `kTrue` if operation succeeded, else `kFalse`.

rcedit.\$setIcon()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

rcedit.\$setIcon(*cFileName*, *clcoPath*)

Description

Sets icon to exe/dll in *cFileName* using filepath to .ico in *clcoPath*. Returns kTrue if operation succeeded, else kFalse.

rcedit.\$setproductversion()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

rcedit.\$setProductVersion(*cFileName*, *cProductVersion*)

Description

Sets product version to value from *cProductVersion* to exe/dll in *cFileName*. Returns kTrue if operation succeeded, else kFalse.

rcedit.\$setresourcestring()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

rcedit.\$setResourceString(*cFileName*, *cKey*, *cValue*)

Description

Sets the resource string with key *cKey* to value *cValue* to exe/dll in *cFileName*. Returns kTrue if operation succeeded, else kFalse.

rcedit.\$setversionstring()

Function group	Execute on client	Platform(s)
rcedit	NO	All

Syntax

rcedit.\$setVersionString(*cFileName*, *cKey*, *cValue*)

Description

Sets version string with key *cKey* to value *cValue* to exe/dll in *cFileName*. Returns kTrue if operation succeeded, else kFalse.

replace()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

replace(source-string,target-string,replacement-string)

Description

Replaces the first occurrence of the *target-string*, within the *source-string*, with the *replacement-string*. Returns the resulting string.

Example

```
Calculate lString as replace('admInIstratIon',chr(73),'i')
# returns 'adminIstratIon'
```

replaceall()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

replaceall(source-string,target-string,replacement-string)

Description

Replaces all occurrences of the *target-string*, within the *source-string*, with the *replacement-string*. Returns the resulting string.

Example

```
Calculate lString as replaceall('admInIstratIon',chr(73),'i')
# returns 'administration'
```

rgb()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

rgb(red,green,blue)

Description

Returns the RGB value formed from the 3 supplied color components (each of which must be 0-255).

The three arguments *red*, *green*, *blue* correspond to the RGB value of the desired color; each must be an integer in the range 0-255. For example, yellow has an RGB value of 255,255,0.

See also rgba().

Example

```
Do $cinst.$objs.FIELD1.$forecolor.$assign(rgb(0,178,178))
# changes the object forecolor to green
```

rgba()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

```
rgba(red,green,blue,alpha)
or
rgba(constant,alpha)
```

Description

Returns the RGBA value formed from the 3 supplied color components (each of which must be 0-255) and an alpha component (0-255), where 255 is fully opaque. Alternatively, a color constant and an alpha value can be supplied.

The four arguments *red*, *green*, *blue*, *alpha* correspond to the RGBA value of the desired color; each must be an integer in the range 0-255.

The return value can be assigned to an RGBA color property, but only where a color *and* alpha value is required.

See also rgb().

Example

```
Do $cinst.$objs.FIELD1.$examplecolor.$assign(rgb(0,178,178,127))
# changes the object color to green with 50% opacity
Do $cinst.$objs.FIELD1.$examplecolor.$assign(rgb(kRed,192))
# changes the object color to red with 75% opacity
```

right()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

```
right(string,n)
```

Description

Returns the substring comprising the last *n* characters of the *string*.

Example

```
Calculate lResult as right('Elephant',3)
# returns 'ant'
```

rmousedn()

Function group	Execute on client	Platform(s)
<i>Mouse</i>	NO	All

Syntax

rmousedn()

Description

Returns true if the right mouse button is held down (under Windows and Linux), or the Ctrl key is held down while the mouse/pointer is clicked (under macOS). If `rmousedn()` is true, `mousedn()` is also true but not vice versa (note no argument).

Note: This function is for desktop apps using Window classes only, not web or mobile apps.

rmouseup()

Function group	Execute on client	Platform(s)
<i>Mouse</i>	NO	All

Syntax

rmouseup()

Description

Returns true if, after being pressed, the right mouse button is up (under Windows and Linux) or the mouse is up after the mouse/pointer has been pressed with the Ctrl key held down (under macOS) (note no argument required).

Note: This function is for desktop apps using Window classes only, not web or mobile apps.

rnd()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

rnd(number,dp)

rnd(number,dp[,asnumber=kfalse])

Description

Rounds a *number* to a number of decimal places specified in *dp* and returns the result.

Rounds a *number* to a number of decimal places specified in *dp* and returns the result as a string, or a number if *asnumber* is `kTrue`.

Note that the return value of **rnd()** is (by default) a character string. This is the only representation of the returned number guaranteed to hold the result correctly, due to potential floating point inaccuracies.

You need to be careful when comparing the return values of two calls to **rnd()**, since the return values are character strings, and will be compared as strings.

The optional parameter *asnumber* allows you to specify that the function should return a Number, rather than a Character value. Note that this may lose some precision if the data type cannot represent the full floating point value.

Example

```
Calculate lResult as rnd(2.105693,5)
# returns 2.10569
```

```
Calculate lResult as rnd(2.105693,3)
# returns 2.106
```

```
Calculate lResult as rnd(0.5,0)
# returns 1
```

rolldice()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

rolldice(number,faces)

Description

Returns the result of a die roll. You specify the *number* of dice to roll and the number of *faces* on each die.

Example

```
Calculate lResult as rolldice(2,6)
# "rolls" two normal, six-sided dice and puts the result in lResult
```

rollstring()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

rollstring(string-formula)

Description

Returns the result of a die roll from a *string-formula*.

The format of the *string-formula* is:

rollstring NdF [+ - * / offset]

where N is the number of dice, d is a delimiter, and F is the number of faces for each die. In addition, you can add, subtract, multiply, or divide by an offset.

Example

```
Calculate lResult as rollstring('2d6')
# returns the result of rolling two standard sixed-faced dice
```

```
Calculate lResult as rollstring('3d6+1')
```

```
Calculate lResult as rollstring('12d4+6')
```

row()

Function group	Execute on client	Platform(s)
<i>List</i>	YES	All

Syntax

row(expression1[,expression2]...)

Description

Creates and returns a row with a column for each parameter set to the value of the supplied *expression*. When a parameter is a variable, the column is named using the variable name: see the example.

Example

```
Calculate lRow1 as row(lVar1,lVar2,lVar3)
# creates a row variable with the column names lVar1, lVar2, lVar3
```

rpos()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

rpos(substring,string)

Description

Returns the position of the last occurrence of *substring* in *string*, or zero if *substring* is not contained in *string*.

The *substring* must be contained within *string* in its entirety for the returned value to be non-zero. Also, the comparison is case sensitive.

Example

```
Calculate lResult as rpos('Mouse','Mickey Mouse')
; returns 8
```

```
Calculate lResult as rpos('mouse','Mickey Mouse')
; return 0, note case
```

```
Calculate lResult as rpos(' ','R S W Smith')
; returns 6, that is the position of the last space character
```

rxpos()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

```
rxpos(rx,str,ignorecase,words,mchlen[,captureRow])
```

Description

Returns position of characters matching regular expression *rx* in *str*, or zero for no match; sets *mchlen* to length of match. *ignorecase* and *words* are Boolean, true for ignore case or only match whole words.

The optional parameter *captureRow* has been added to PCRE2 regular expressions (the default from Studio 10.2). You can pass this a row variable that returns the captured groups resulting from the regular expression match operation. This is a standard feature of regular expressions – groups correspond to the parts of the regular expression contained in parentheses (provided that the open parenthesis is not followed by *?*: indicating a non-capturing group). Groups can also optionally be named, and they are numbered 1 to *n*, with various rules regarding duplication when using the *|* operator. For example:

1. When *rxpos()* locates nothing, it sets *captureRow* to empty.
2. When *rxpos()* locates something, and the *captureRow* parameter is supplied, it adds a column to the *captureRow* for each captured group, where the column name is *G<n>* for group number *n* where the group is not named, or the group name.

The following examples will illustrate this:

```
Calculate cString as "2017-01-02"
```

```
Do rxpos("(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})$",cString,0,0,cLen,cRow) Returns cOffset
```

After executing the above line, *cRow* is a row with three columns named *year*, *month*, and *day*, with the values 2017, 01 and 02, respectively.

```
Calculate cString as "hey_test_ho"
```

```
Do rxpos("(hey|ho)_test_(ho|hey)",cString,0,0,cLen,cRow) Returns cOffset
```

After executing the above line, *cRow* is a row with two columns named *G1* and *G2*, with the values *hey* and *ho*.

You can mix named and unnamed capture groups.

setfye()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

```
setfye(date)
```

Description

Sets the *date* of the fiscal year end.

The *date* does not have to be in the current year, that is, the function ignores the year part of the date. The setting of the fiscal year end affects all other date functions that involve quarters. It returns the previous value so you can save it for later use.

Example

```
Calculate lDate as setfye('MAR 31')
```

```
# sets the fiscal year end to March 31st
```

```
Calculate lDate as setfye('12 31 03')
```

```
# sets the fiscal year end to December 31st and ignores the year
```

setseed()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

setseed(seed)

Description

Sets the random number *seed* for the random functions, such as *rand()*. *setseed()* converts *seed* into a Long number. It returns the previous seed as an integer number.

Example

```
Calculate lResult as setseed(10)
# sets seed to 10 and returns previous seed in lResult
```

setws()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

setws(datepart)

Description

Sets the beginning of the week using one of the day of the week *datepart* constants, e.g. *kSunday*. This affects the value returned by the functions *dpart()*, *fday()* and *lday()*.

It returns the previous value so you can save it for later use. If the *datepart* is invalid this function still returns the week start but does not change it.

The datepart constants that you can use are: *kSunday*, *kMonday*, *kTuesday*, *kWednesday*, *kThursday*, *kFriday*, *kSaturday*.

Example

```
Calculate lDate as setws(kMonday)
# sets Monday as the week start
```

sha256()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

sha256(binary)

Description

Returns the 32-byte binary SHA-256 hash of the supplied binary data. Returns #NULL if binary is null or empty.

sha512()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

sha512(binary)

Description

Returns the 64-byte binary SHA-512 hash of the supplied binary data. Returns #NULL if binary is null or empty.

shift()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

shift()

Description

Returns true if the Shift key is being pressed.

shufflelist()

Function group	Execute on client	Platform(s)
<i>Random</i>	NO	All

Syntax

shufflelist(sourcelist,targetlist,number)

Description

Shuffles the items in *sourcelist*, the specified *number* of times, and puts the results in *targetlist*.

A value of 2 or 3 for *number* provides a good shuffle. **shufflelist()** does not support Binary fields, List fields, and Picture fields stored in a list. An empty or null date converts to '31 DEC 00' in the *targetlist*.

Example

```
Do shufflelist(lSourceList,lTargetList,3)
# shuffles lSourceList three times and puts the result in lTargetList
```


sin()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

sin(*angle*)

Description

Returns the Sine of an *angle* where the *angle* is in degrees (or radians if #RAD is true).

Example

```
Calculate lResult as sin(30)
# returns 0.5
```

sleep()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

sleep(milliseconds)

Description

Suspends execution for the specified number of *milliseconds*; returns true if execution suspended successfully or false if an error occurred.

The actual delay can be longer in the multi-threaded server if another stack is executing its time slice when the sleep delay expires. As soon as you execute sleep() in the multi-threaded server, other waiting stacks can run.

split()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

split(*string*[,*delimiters*='',*stripwhitespace*=kfalse])

split(*string*[,*delimiters*=kComma,*stripwhitespace*=kfalse])

Description

Splits *string* at the character(s) in *delimiters* and returns a list of the resulting substrings. Strips leading and trailing whitespace from each substring if *stripWhitespace* is kTrue.

sqr()

Function group	Execute on client	Platform(s)
<i>Number</i>	YES	All

Syntax

sqr(number)

Description

Returns the square root of a *number*. Omnis defines the square root of a negative number X as *sqr(abs(X))*.

Example

```
Calculate lResult as sqr(100)
# returns 10
```

```
Calculate lResult as sqr('-301.56')
# returns 17.37 approx
```

```
Calculate lResult as mid('Omnis',sqr(16),2)
# returns 'is'
```

startswith()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

startswith(*string*,*start*[,*ignorecase=kFalse*])

Description

Returns true if the *string* starts with the non-empty string *start*. If *ignorecase* is *kTrue*, the function uses case-insensitive comparison.

Example

```
Calculate lString as 'Fast prototyping'
```

```
If startswith(lString,'Fast')
  # returned True
End If
```

```
Calculate lBoolean as startswith(lString,'FAST',kTrue)
# lBoolean = True
```

stddevc()

Function group	Execute on client	Platform(s)
<i>List</i>	NO	All

Syntax

stddevc(listname,column[,ignore-nulls])

Description

Returns the standard deviation for a list column specified by *listname* and *column*. `stddevc()` can only be used with columns defined using variables so it cannot be used with lists defined from a SQL class.

If you set *ignore_nulls* to 1, null values are ignored and not counted. If you omit this parameter or it evaluates to zero, nulls are treated as zero values and are counted.

Example

```
Calculate lResult as jst(stddevc(lList,lCol1),'N2')
# returns the standard deviation rounded to 2 decimal places
```

stgetcol()

Function group	Execute on client	Platform(s)
<i>Client String Table</i>	NO	All

Syntax

stgetcol(table)

Description

Returns the name of the current column for string table *table* (iOS plug-in client methods only).

stgettext()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	YES	All

Syntax

stgettext(id)

Description

Returns the string with the *id* from a string table, or empty if the lookup fails. (Supported in client methods; when using \$stringtable in the remote task supported in client and server methods for the task string table).

StringTable.\$colcnt()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$colcnt([*cTableName*])

Description

Returns the number of columns in string table *cTableName*, or an error code which is less than zero (see the *kStringTable...* constants).

Example

```
Calculate lResult as StringTable.$colcnt('MyStringTable')
# returns the number of columns in MyStringTable
```

StringTable.\$getcolumnname()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$getcolumnname([*cTableName*])

Description

Returns the current column name for the string table specified by *cTableName*, or an error code which is less than zero (see the *kStringTable...* constants).

Example

```
Calculate lResult as StringTable.$getcolumnname('MyStringTable')
# returns the name of the current column in MyStringTable
```

StringTable.\$getcolumnnumber()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$getcolumnnumber([*cTableName*])

Description

Returns the current column number for the string table specified by *cTableName*, or an error code which is less than zero (see the *kStringTable...* constants).

Example

```
Calculate lResult as StringTable.$getcolumnnumber('MyStringTable')
# returns the number of the current column in MyStringTable
```

StringTable.\$getlistfromfile()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

```
StringTable.$getlistfromfile(cPathname)
```

Description

Reads the string table file *cPathname*, and returns the string table list it contains.

StringTable.\$gettablelist()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

```
StringTable.$gettablelist(&lList)
```

Description

Populates a two column *lList* with the loaded string table names in column 1 and their pathnames in column 2. Define *lList* to have two character columns before calling this method.

Example

```
Do StringTable.$gettablelist(lTableList)
# returns a list of all loaded string tables
```

StringTable.\$gettext()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

```
StringTable.$gettext(cRowID)
```

Description

Returns the text for a cell, or an error code less than zero (a *kStringTable...* constant). *cRowID* is either the row name within the current column, or when using multiple tables it has the syntax *TABLENAME.ROWNAME[COLUMNNAME]*.

Example

```
Calculate lResult as StringTable.$gettext('A')
# returns the text from the cell with id 'A' in the current column
```

StringTable.\$loadcolumn()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$loadcolumn(cColumnNameOrName,cTableName,cPathname)

Description

Loads column *cColumnNameOrName* from string table *cPathname* into table *cTableName*. Returns kStringTableOK or an error code which is less than zero (see the kStringTable... constants).

Example

```
Do StringTable.$loadcolumn(lCol1,lTable,lPath)
# loads the column lCol1 from the table specified in lPath, into the table lTable
```

StringTable.\$loadexistingtablefromlist()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$loadexistingtablefromlist(cTableName,lList)

Description

Replaces an existing string table with the content of a list. Returns kStringTableOK or an error code which is less than zero (see the kStringTable... constants).

StringTable.\$loadlistfromtable()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$loadlistfromtable(cTableName)

Description

Copies string table *cTableName* to a list, and returns the list.

Example

```
Calculate lTableList as StringTable.$loadlistfromtable(lTable)
# copies the string table lTable to a list lTableList
```

StringTable.\$loadstringtable()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$loadstringtable(*cTableName,cPathname*)

Description

Loads string table from file *cPathname*, and gives it the name *cTableName*. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Do StringTable.$loadstringtable(lTable,lPath)
# loads the table specified in lPath into the table lTable
```

StringTable.\$loadtablefromlist()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$loadtablefromlist(*cTableName,cPathname,lList*)

Description

Creates a string table from a list. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Do StringTable.$loadtablefromlist(lTable,lPath,lTableList)
# creates a string table from lTableList
```

StringTable.\$redraw()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$redraw(*hwnd*)

Description

Redraws the window specified by *hwnd*. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

StringTable.\$removestringtable()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$removestringtable(*cPathname*)

Description

Deletes the string table file specified by *cPathname*. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Do StringTable.$removestringtable(lPath)
# removes the string table specified in lPath
```

StringTable.\$rowcnt()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$rowcnt([*cTableName*])

Description

Returns the number of rows in string table *cTableName*, or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Calculate lResult as StringTable.$rowcnt('MyStringTable')
# returns the number of row in MyStringTable
```

StringTable.\$savestringtable()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$savestringtable(*cTableName*)

Description

Saves the string table specified by *cTableName*. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Do StringTable.$savestringtable(lTable)
# saves the string table lTable
```

StringTable.\$setcolumn()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$setcolumn(*cColumnNameOrName*)

Description

Sets the current column (which can be prefixed by 'TABLENAME.'. Returns `kStringTableOK` or an error code which is less than zero (see the `kStringTable...` constants).

Example

```
Do StringTable.$setcolumn(lCol1)
# sets lCol1 as the current column
```

StringTable.\$unloadall()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$unloadall()

Description

Unloads all string tables from memory.

Example

```
Do StringTable.$unloadall()
# removes all string tables from memory
```

StringTable.\$unloadstringtable()

Function group	Execute on client	Platform(s)
<i>StringTable</i>	NO	All

Syntax

StringTable.\$unloadstringtable(*cTableName*)

Description

Unloads the string table *cTableName* from memory. Returns kStringTableOK or an error code which is less than zero (see the kStringTable... constants).

Example

```
Do StringTable.$unloadstringtable(lTable)
# removes the string table lTable from memory
```

Strip()

Function group	Execute on client	Platform(s)
<i>External functions</i>	NO	All

Syntax

Strip(string,character)

Description

Removes all instances of *character* from *string*, and returns the resulting string.

Example

```
Calculate #S1 as Strip("Sunshine",'n')
# returns "Sushie"
```

strpbrk()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

strpbrk(string1,string2)

Description

Returns the substring of *string1* from the point where any of the characters in *string2* match *string1*.

Example

```
Calculate lString as "this is a test"
Calculate lNewString as strpbrk(lString,"absj")
# returns the string "s is a test"
```

strspn()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

strspn(string1,string2)

Description

Returns the index of the first character in *string1* that does not match any of the characters in *string2*.

Example

```
Calculate lResult as strspn(lString1,lString2)
# lResult is a 1-based index, or len(lString1)+1 if all characters are in lString2
```

strtok()

Function group	Execute on client	Platform(s)
<i>String</i>	NO	All

Syntax

strtok('string1',string2)

Description

Tokenises *string1*, using *string2* as the delimiter with which to tokenize.

This function returns tokens which are a substring of *string1* until any character in *string2* matches a character in *string1*. When **strtok()** is called, the token found in *string1* is removed, so that the function looks for the next token the next time it is called.

Note: **strtok()** does not support null characters in *string1* or *string2*.

Example

```
Calculate lString1 as "The quick brown fox, jumped over the lazy dog"
Repeat Calculate lString2 as strtok('lString1'," ")
  OK message {Token = [lString2]}
Until lString2=''
# returns each word in lString1 in an OK Message
```

stsetcol()

Function group	Execute on client	Platform(s)
<i>Client String Table</i>	NO	All

Syntax

stsetcol(table,col)

Description

Sets the current column for lookups from string table *table* to the column with name *col* and returns Boolean true for success (iOS plug-in client methods only).

style()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

style(textescape[,value])

Description

Returns a style sequence suitable for embedding in styled text. *textEscape* is a kEsc... constant; *value* is a suitable value for the particular *textEscape*.

Valid constants for the *style-character* are listed under 'Text escapes' in the Constants pane of the IDE Catalog window (there is also a group called 'JavaScript text escapes' for the JavaScript client). Depending on the style character, you may also need to specify a *value*, which itself can be a constant. You can use this function to format the columns in a headed list box field; in this case, you can insert an icon by specifying its ID, change the text color, or set a text style such as italic.

Note that the alignment escapes (kEscCTab, kEscLTab and kEscRTab) are not designed to work with the headed list box. The headed list box method \$setcolumnalign() can be used to set the alignment of a column.

The JavaScript client supports a subset of the standard text escapes, and some additional values specific to the JavaScript client, as follows:

Constant	Description
kEscColor	In a client-executed method, the color parameter must be a numeric literal or constant.
kEscStyle	In a client-executed method, the style parameter must be a numeric literal or constant.
kEscBmp	In a client-executed method, the icon id parameter must be either a numeric literal, or the sum of a numeric literal and an icon size constant e.g. 1710+k48x48
kEscJsNewline	No additional parameters are required. Inserts a line break tag. There is also a new function br() which can be used as short-hand to do this.
kEscJsClose	No additional parameters are required. Closes the current open style information (inserted by kEscColor or kEscStyle) in the styled text and reverts to the original color and text style. Note -kEscColor and kEscStyle insert a tag to style the text. kEscJsClose closes the tag.
kEscJsHtml	Inserts raw HTML (the second parameter to style()).

Example

```
# to format the columns in a headed list box you could use the following calculation
Do con(1Col1,style(kEscBmp,1756),kTab,1Col2,style(kEscColor,kRed),kTab,1Col3,style(kEscStyle,kItalic))
# give 1Col1 a blue spot icon, 1Col2 is red and 1Col3 italic
```

styledtohtml()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

styledtohtml(*styledtext*)

Description

Converts *styledtext* (a character string with styles embedded using the *style()* function) into HTML (JavaScript client only, execute on client only; this function cannot be used in a server method).

sys()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

sys(*number*)

Description

Returns information about the current system depending on a *number* argument.

Using the **sys()** function, you can obtain system information such as the current printer name, the pathname of the current library, the screen width or height in pixels, and so on.

You can use the following *number* values with the *sys()* function.

Number	Description
1	returns the Omnis version number.
2	returns the Omnis program type byte: bit 0 = full program (value 1), bit 1 = runtime (value 2), bit 2 = evaluation (value 4), bit 4 = Unicode (value 16) - always set. Note that the current version of Omnis does not support the use of "integrated" versions.
3	returns your company name entered on installation.
4	returns your name entered on installation.
5	returns your serial number entered on installation.
6	returns the platform code of the current executable: 'N' = Windows NT, 2000, 2003, XP, Vista, Windows 7/8/10. 'U' = Linux. 'X' = macOS. ('W' = Windows 95, 98 or ME is no longer used)
7	returns a string containing the version number of the current OS. For example, returns "6.2" under Windows 10, and "10.4.8" under macOS 10.4.8
8	returns the platform type of the current Omnis program as a string: 'MAC64', 'MACARM' (Apple M1), 'WIN32', 'WIN64', 'UNIX'
9	returns the pathname separator for the current platform: '\ ' (back slash) for Windows, '/' (forward slash) for Unix and 64-bit macOS (32-bit Macs will return :)
10	returns the pathname of the current open library file.
11,...,20	returns the pathname(s) of the current open data file segment(s) (empty if none are open).
21	returns the pathname of the current print file name (empty if not open).
22	returns the pathname of the current import file name (empty if not open).
23	returns the current port name (empty if no port open).

Number	Description
24	returns the current report device, for example, Printer, Screen, Preview, File (Preview is the default).
30,,,49	returns the name of the installed user-defined fat client menu(s) starting from the left-most menu (empty if none are installed).
50,,,79	returns the name of the open user-defined fat client window(s) starting with the top window (empty if none are open).
80	returns the current report name (empty if no report set).
81	returns the current search name (empty if no search set).
82	returns the main file name (empty if no main file set).
83	returns the number of records in main file.
85	returns the name of currently executing method in the form class name/method number.
86	returns a list of event parameters for the current event. The first parameter is always pEventCode containing an event code representing the event, for example, evClick for a click on a button: a second or third event parameter may be supplied which tells you more about the event
87	returns horizontal screen resolution in pixels per inch
88	returns vertical screen resolution in pixels per inch
89	returns the text for the current search calculation, or empty if no calculation is set.
90	returns the number of methods on the method stack. This does not work for client requests running in a thread of the Multi-threaded Server.
91	returns the decimal separator
92	returns the thousand separator
93	returns the parameter separator for calculations
94	returns the file class field name separator
101	returns the current printer name, and network path (empty if not connected).
104	returns the screen width in pixels.
105	returns the screen height in pixels.
106	macOS: not relevant - returns a fixed large value.Other platforms: returns the size in bytes of available physical memory.
107	macOS: not relevant - returns a fixed large value.Other platforms: returns the total size in bytes of physical memory.
108	macOS: not relevant - returns a fixed large value.Other platforms: returns a number between 0 and 100 that gives a general idea of current memory utilization, in which 0 indicates no memory use and 100 indicates full memory use.
109	macOS: not relevant - returns zero.On Windows and Unix returns the unused memory in bytes. This is the number of bytes available in the paging file, added to the number of bytes of available physical memory.
110	Deprecated. Does not return a value using the 64-bit versions of Omnis, and can no longer be relied on as an accurate indicator of the processor type or computer in use.Returns the CPU type.For Intel Processors:3 = 803864 = 804865 = PentiumFor PowerMac Processors:257 = PowerPC 601259 = PowerPC 603260 = PowerPC 604262 = PowerPC 603e263 = PowerPC 603ev264 = PowerPC 750 & G3265 = PowerPC 604e266 = PowerPC 604ev268 = G4313 = G5
111	Deprecated. (macOS only) returns the Apple ROM version: 121 = Si, 124 = IIsi, CI & FX.
114	Deprecated. (macOS only) returns 1 (true) if Apple events are available, 0 otherwise.
115	Returns the pathname of the folder containing the installed writeable files, including the terminating path separator. To get the pathname of the folder containing the Omnis executable, use sys(215).
116	(Unix only) returns 1 (true) if Omnis has been configured to force all file names to be lower case, 0 otherwise.

Number	Description
118	(Windows only) returns additional information about the operating system version. The information includes major and minor version numbers, a build number, a platform identifier, and information about product suites and the latest Service Pack installed on the system. This function returns the operating system version information in the Windows OSVERSIONINFOEX structure, so you should obtain information about this structure for full details about the information returned (see the Microsoft web site).
120	(Windows only) returns the width of the current dialog base-width unit based on the current system font; this differs for Small and Large font mode.
121	(Windows only) returns the height of the current dialog base-width unit based on the current system font; this differs for Small and Large font mode.
122	returns true if the Web Client server communications listener started successfully. It returns false if listener startup failed, or has not yet occurred.
123	returns the build number of the Omnis executable. The Omnis About box shows the same build number in the version string
130-138	Not supported in Omnis Studio 5.0 and later.
185	returns the current executing method in the format:Libraryname.Classname/Methodname/LinenumberThis is in addition to sys(85) which returns the current executing method in the form Classname/Methodname.
190	returns the number of times Omnis has loaded a class from disk and added it to the memory class cache.
191	returns the number of times Omnis deleted a memory class cache entry, when it added a class to the cache, meaning that the class deleted from the cache may need to be reloadedsys(190) and sys(191) provide information you can use to monitor the impact of changing the preference <i>root.prefs.\$maxcachedclasses</i> . Too low a value for this preference may result in a performance hit, due to too many classes being repeatedly loaded from disk.
192	sys(192) caters for both error handlers and other situations where information about the method stack might be useful. It returns a list representing the current method call stack (with a line for each line that would appear on the debugger stack menu). The first line in the list corresponds to the call to sys(192), and subsequent lines corresponding to how the method running the previous line was called.The list has the following columns:classitem - item reference to the class containing the method.object - the name of the object containing the method in the class; empty if it is a class method.method - the name of the executing method.line - the line number of the executing method line.linertext - the text for the executing method line.params - the parameters passed to the method. This is a two-column list, with columns name and value. The value is the value that would be displayed as a variable tooltip for the parameter.methoditem - item reference to the method.inst - item reference to the instance executing the method.The "sys192excludesIDEmethods" item in the "defaults" section of config.json, specifies whether or not to include IDE method calls in the list returned from sys(192) (the default is true). This will exclude an IDE method if the library containing the method is marked as always private.The "sys192ListRowLimit": N item (with N added) in the "defaults" section of config.json allows lists (and rows) with up to N rows to be included as a third column in the output parameter data. If N <= 0 (the default) then no third column is added. If N > 0, then each parameter in the parameter list stored in each line of the sys(192/292) list has a third column, which for lists and rows contains the actual list (or row) data, if the list or row has less than or equal to N lines. In all other cases (not a list or row, or line limit N exceeded) column 3 is empty.
193	returns a list which is a copy of the trace log.

Number	Description
194	returns the list of open windows in the IDE. The list has the following columns: windowname - the name of the window, as it would appear in the windows menuclassitem - if the window is a class or method editor, the item reference to the class; otherwise nullismethodeditor - boolean, true if and only if the window is a method editor windowThe current line is non-zero if and only if the window is the top design window, i.e. the top class editor or method editor window. When deciding the top design window, the catalog, component store, and property inspector are ignored.
195	returns the list of selected objects, if there is a top design window (as defined by sys(194) above), and if the top design window is a window, report, toolbar, menu, or remote form editor (NOT a method editor). The list has one column: objectitem - the item reference of the selected object
196	returns the list of all open libraries (including private libraries) and their VCS build properties. The list has the following columns: name - the internal name of the librarypathname - the pathname of the library filevcsbuildersname - the \$vcsbuildersname property of the libraryvcsbuildnotes - the \$vcsbuildnotes property of the libraryvcsbuilddate - the \$vcsbuilddate property of the library
197	Deprecated. Returns true (1) if running on Windows XP with themes enabled, false (0) if not.
198	Returns the Bind status: an integer which indicates the status of binding the web client server port number to the socket on which the server receives incoming connections. The value is -1 if bind has not been attempted yet, 0 if bind was successful, or a positive value (a socket layer error code) if the bind failed
199	returns a list with a row for each method command. The list has two columns: column 1 is the command group, column 2 is the command name
202	returns the command line parameters passed to Omnis.If the first argument passed via the command line is not a library, you can prepend it with a - (hyphen) character. So when using sys(202), you will get all arguments passed to Omnis, including the first argument, whether it starts with a hyphen or not
203	(Win32/Win64 only) returns 1 (true) if Omnis is running as a service
204	returns a list with a row for each function. The list has three columns: column 1 is the function group, column 2 is the function name and column 3 is the function description
205	returns the byte ordering of the hardware. True (1) for big-endian machines, false (0) for little-endian machines.
209	for macOS, returns the pathname of the folder containing the Omnis.app bundle, including the terminating path separator; this may be different to the pathname of the folder containing the Omnis executable (returned by sys(115)) if you are using Omnis as a bundle. For other platforms, returns the same pathname as sys(115).
210	returns the number of processors for the current machine (depending on the information reported by the operating system, the value may not reflect logical processor cores).
212	returns the list of sort fields for the executing method stack. The list has the following columns:name - the name of the sort fielddescending - boolean, true if this field is sorted in descending orderupper - boolean, true if sorting of this field is case-insensitivesubtotals - boolean, true if subtotals are printed when the sort field changesnewpage - boolean, true if a new page is started when the sort field changes
215	Returns the pathname of the folder containing the Omnis executable, including the terminating path separator.
216	Returns the pathname of the currently executing Omnis program.
218	Modifies the OEM character conversion table to leave CR and LF unchanged.
219	Restores the OEM character conversion table to the original mappings for CR and LF.
228	Modifies the OEM character conversion table to leave TAB unchanged.
229	Restores the OEM character conversion table to the original mapping for TAB.

Number	Description
232	(Win32/Win64 only) returns 1 (true) if Omnis is running as an elevated user or administrator
233	(Windows and Linux only) Returns the title of the main Omnis window
234	Returns a row containing statistics about RESTful requests to the Omnis Server. The row has three columns: column 1 is the count of successful calls, column 2 is the count of calls resulting in an error, column 3 is the count of calls internally requeued because there was not a free user
235	Returns the value of the server configuration item <code>timeOffsetMinutes</code> in <code>config.json</code> . Omnis adds this to the current system date-time when generating the value for #D and #T. <code>timeOffsetMinutes</code> defaults to zero if it is not present in <code>config.json</code>
236	Returns true if VCS branching is enabled.
237	Returns an item reference to the method currently being edited, if the method editor is the top window (and only one method is selected).
238	Returns 1 (true) if the remote debug server is running
239	Returns true if the Startup method in the library has finished, or false if not
240	Returns a string that identifies the current monitor configuration. You could use this in your code to branch according to the monitor configuration of end users
241	Returns a list containing the find and replace log list from <code>\$findandreplace</code> . Where possible, the 6th column named 'item' contains an item reference either directly to the found data, such as a method line, or to the item containing the found data, such as a property
242	Returns a string which uniquely identifies the current child process when running in the Multi Process Server. If the current process is not a child process, returns 0.
243	Returns true if Omnis is currently running as a Multi Process Server.
250	Returns a list of files which were used to open Omnis, e.g. double-clicked from the Finder, or passed on the command line. This is empty if Omnis was opened directly by double-clicking
251	On Windows, returns the <i>width</i> of the available area of the main application window; the same information is returned in the <code>\$mainresized</code> task message. On macOS, returns the overall <i>width</i> of all screens
252	On Windows, returns the <i>height</i> of the available area of the main application window; the same information is returned in the <code>\$mainresized</code> task message. On macOS, return the overall <i>height</i> of all screens
253	Returns true if Omnis is using dark mode.
254	Returns the pathname of the folder where Omnis writes scripts and other files used to generate PDF documents.
255	Returns the pathname of the folder where Omnis writes temporary PDF files.
256	Returns the maximum server license count.
257	Returns the current number of server licenses in use.
290	Returns the number of methods on the method stack excluding any that are to be cleared. This does not work for client requests running in a thread of the Multi-threaded Server.
292	Returns an empty or single line list containing the same columns as <code>sys(192)</code> where the line represents the calling method; see <code>sys(192)</code>
3000	Turn on logging of method command execution.
3001	Turn off logging of method command execution.
3003	Include method commands executed by IDE tasks in method command logging turned on using <code>sys(3000)</code> .
3004	Exclude method commands executed by IDE tasks from method command logging turned on using <code>sys(3000)</code> .

Example

The following example uses `sys(6)` to test the current OS and branches accordingly.

```
Calculate lPlatform as sys(6)
```

```

If lPlatform='X' ## on macOS
  Do FileOps.$createdir('/Omnis/Test')
Else If lPlatform='U' ## on Linux
  Do FileOps.$createdir('/home/omnis/test')
Else ## on Windows platforms
  Do FileOps.$createdir('c:\Omnis\test')
End If

```

systemversion()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

systemversion()

Description

Returns a row of version information about the current operating system, with columns platform, major, minor, build, and server. The platform parameter is the platform code; major, minor, and build identify the system version (not for Linux); server is true for Window server systems.

Note this function replaces all the “is...()” functions (e.g. iswindows10()) for determining the current operating system version.

tan()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

tan(number)

Description

Returns the Tangent of an *angle* where the *angle* is in degrees (or radians if #RAD is true).

Example

```

Calculate lResult as tan(45)
# returns 1

```

textsize()

Function group	Execute on client	Platform(s)
<i>Font handling</i>	NO	Windows, Linux

Syntax

textsize(string, fontname, pointsize, style, 'width', 'depth')

Description

Returns the width and depth in pixels of the specified text or string.

Note that `textsize()` is deprecated. Use `FontOps.$reptextheight()`, `FontOps.$reptextwidth()`, `FontOps.$wintextheight()` or `FontOps.$wintextwidth()` instead.

The *string* parameter can be a literal string or character variable with a maximum length of 255; *fontname* is the name of the font; *pointsize* the point size of the font; *style* is represented by an integer, that is, 0 = Normal, 1 = Bold, 2 = Italic, 4 = Underline (or a combination of these: 3 would be bold-italic, for example). The *width* and *depth* parameters hold the returned values in pixels. The *width* and *depth* parameters are integer variables to hold the width and depth values returned; these must be in quotes. The function also returns a value of 0 to indicate the font does not exist or an error occurred, otherwise 1 is returned indicating the text was found.

Example

```
Calculate lFontStyle as 6 ;; Italic Underline
Calculate lFontName as "Arial"
Calculate lFontSize as 16
If textsize(lString,lFontName,lFontSize,lFontStyle,'lWidth','lDepth')<>0
  # do something depending on lWidth and lDepth
Else
  # font didn't exist or error occurred
End If
```

tim()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

tim(number[,timeformat])

Description

Converts the specified *number* value to a time determined by the *timeformat* argument and returns the result; #FT is used as the time formatting string if the second argument is not specified.

If the first argument is already a date/time, its format is changed to the format given by the second argument.

The #FT string defaults to 'H:N' and so *tim(number)* takes *number* to be a number of minutes. If you supply a format string such as 'N.S', *number* will be taken as a number of seconds.

With a second argument, **tim()** is equivalent to `dat()` with two arguments. The format string determines how the conversion is carried out.

Example

```
Calculate lTime as tim(1)
# returns 00:01 when #FT is H:N (the default)
```

```
Calculate lTime as tim(950)
# returns 15:50
```

```
Calculate lTime as tim(950,'H:N.S')
# returns 00:15.50, that is 950 seconds
```

tot()

Function group	Execute on client	Platform(s)
List	NO	All

Syntax

tot([listname,]fieldname)

Description

Returns the total of the stored values of *fieldname* in a list specified by *listname*.

The *fieldname* argument must be a field stored in the list, not a constant or expression. If the *listname* argument is not specified the current list is used. This function does not work for table based lists. If *fieldname* is not a numeric field, all values are converted to their numeric equivalents before being accumulated.

Example

```
Calculate lResult as tot(lList,lCol1)
# returns total of all lCol1 field values stored in the list lList

Calculate lResult as tot(lCol1)
# returns total of all lCol1 field values stored in current list

Calculate lResult as tot(#LSEL)
# returns total number of selected lines in the current list
```

totc()

Function group	Execute on client	Platform(s)
List	NO	All

Syntax

totc([listname,]expression)

Description

Returns the total of an *expression* evaluated for a list specified by *listname*.

If the *listname* argument is not specified, the current list is used. This function does not work for table based lists. The *expression* is totaled for the lines in the specified list.

This is a more general version of the *tot()* function.

Example

```
# If list lList contains field lCol1, the sum of the
# squares of all values of lCol1 in the list is:
# note that totc(lList,lCol1) is the same as tot(lList1,lCol1)
Calculate lResult as totc(lList,lCol1*lCol1)
```

tracelog()

Function group	Execute on client	Platform(s)
<i>General</i>	YES	All

Syntax

tracelog(*string*)

Description

Writes the string to the trace log (or JavaScript console for client-executed code). Does nothing if debugging is disabled using the library property \$nodebug. Returns the string if the string was written, or #NULL if it was not written.

trim()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

trim(*string*[,*leading*=ktrue,*trailing*=ktrue,*character*=space_char])

Description

Removes the specified *leading* and/or *trailing* character from the *string*.

You specify the character to be removed in the *character* argument. If this is omitted the space character is removed from the string by default.

Example

```
Calculate lString as trim(' ABCDE')  
# returns 'ABCDE'
```

```
Calculate lString as trim('*****ABCDE*****',kTrue,kFalse,'*')  
# returns 'ABCDE*****'
```

truergb()

Function group	Execute on client	Platform(s)
<i>General</i>	NO	All

Syntax

truergb(*color*)

Description

Converts the specified *color* into its true RGB value and returns the result.

Example

```
Calculate lValue as truergb(kRed)
# returns RGB value for red
```

```
Calculate lValue as truergb(rgb(255,0,0))
# returns RGB value for red
```

tzcurrent()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

```
tzcurrent()
```

Description

Returns the character string identifying the time zone of the current date and time. The result can be mapped to an alternative using the Built-in strings.

tzdaylight()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

```
tzdaylight()
```

Description

Returns the character string identifying the daylight saving time zone. The result can be mapped to an alternative using the built-in strings.

tzoffset()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

```
tzoffset()
```

Description

Returns the system time zone offset from UTC time in minutes.

tzstandard()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

tzstandard()

Description

Returns the character string identifying the standard time zone. The result can be mapped to an alternative using the built-in strings.

uncompress()

Function group	Execute on client	Platform(s)
<i>Compression</i>	NO	All

Syntax

uncompress(binary,uncompressed-length)

Description

Uncompresses the *binary* variable (containing the result of the previous call to *compress()*) and returns the binary uncompressed data.

Note you need to supply the length of the original uncompressed data.

Example

Calculate `lOriginal` as `uncompress(lCompressed,50)`

unichr()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	YES	All

Syntax

unichr(code1[,code2]...)

Description

Returns a string formed by concatenating the supplied Unicode character codes. Each code is either a number or a string of the form 'U+h', where h is 1-6 characters representing a hexadecimal value.

unicode()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	YES	All

Syntax

unicode(string,position[,hex])

Description

Returns the Unicode value of the character at the *position* in the *string*. The first *position* in the *string* is 1. If Boolean *hex* is true (default false) it returns a hex string 'U+h'.

unicnv()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	NO	All

Syntax

unicnv(srctype,src,dsttype,dst,bom,errtxt)

Description

Converts *src* to *dst*. Returns zero if ok, or errcode and *errtxt*. *Types* are kUniType... constants (not binary). If Boolean *Bom* is true, *dst* has a Unicode BOM if it can.

You can handle CR, LF and TAB characters using the following `sys()` functions:

sys()	Description
218	Modifies the OEM character conversion table to leave CR and LF unchanged.
219	Restores the OEM character conversion table to the original mappings for CR and LF.
228	Modifies the OEM character conversion table to leave TAB unchanged.
229	Restores the OEM character conversion table to the original mapping for TAB.

upp()

Function group	Execute on client	Platform(s)
<i>String</i>	YES	All

Syntax

upp(string)

Description

Returns the upper case representation of a *string*. Any non-alphabetic characters in the *string* are ignored.

Example

```
Calculate lString as upp('Author')
# returns 'AUTHOR'
```

```
Calculate lString as upp('oMnIs')
# returns 'OMNIS'
```

```
Calculate lString as upp(mid('peripheral',3,3))
# returns 'RIP'
```


useradians()

Function group	Execute on client	Platform(s)
<i>Trigonometric</i>	YES	All

Syntax

useradians(*[useradians]*)

Description

Sets or returns #RAD. Either sets #RAD and returns the previous value, or if no parameter is supplied returns the current value of #RAD.

utctoloc()

Function group	Execute on client	Platform(s)
<i>Date and Time</i>	NO	All

Syntax

utctoloc(*datetime*)

Description

Converts the *datetime* (or time) from UTC (Coordinated Universal Time) to the local timezone, and returns the result.

utf8tochar()

Function group	Execute on client	Platform(s)
<i>Unicode</i>	NO	All

Syntax

utf8tochar(*utf8*)

Description

Returns the character string which is represented by the UTF-8 encoded parameter.

Binary Field

Functions

bdif() *binchecksum()* *bincompare()* *binfrombase32()*
binfrombase64() *binfromhex()* *binfromint32()* *binfromint64()*
binlength() *binsearch()* *binbase32()* *binbase64()*
binohex() *binoint32()* *binoint64()* *bitand()*

bitclear() *bitfirst()* *bitmid()* *bitnot()*
bitor() *bitrotatel()* *bitrotater()* *bitset()*
bitshiftl() *bitshiftr()* *bittest()* *bitxor()*
bundif() *bytecon()* *byteget()* *bytemid()*
byteset() *decxtea()* *encxtea()* *sha256()*
sha512()

Class

Functions

cdif() *cundif()*

Client String Table

Functions

stgetcol() *stsetcol()*

Compression

Functions

compress() *uncompress()*

Date and Time

Functions

dadd() *dat()* *ddiff()* *dim()*
dname() *dpart()* *dscy()* *dtd()*
dtm() *dtw()* *dty()* *fday()*
fmtdateime() *fmtshortdate()* *fmtshorttime()* *getdatetime()*
getfye() *getticks()* *getws()* *iso8601toomis()*
isoweek() *isoweekstart()* *lday()* *loctoutc()*
nday() *omnistoiso8601day()* *setfye()*
setws() *tim()* *tzcurrent()* *tzdaylight()*
tzoffset() *tzstandard()* *utctoloc()*

DNet

Functions

DNet.\$addchildren(\$basefolder,\$name,\$getenum())

External Functions

Strip()

Field

Functions

fld() *nam()*

FileOps

Static Functions

FileOps.\$changepermissions(\$filepath,\$permissions)

FileOps.\$createfile(\$filepath,\$mode,\$permissions)

FileOps.\$deletefile(\$filepath,\$permissions)

FileOps.\$getfile(\$filepath,\$permissions)

FileOps.\$getunixpermissions(\$filepath,\$permissions)

FileOps.\$renamefile(\$filepath,\$newfilepath,\$permissions)

FileOps.\$selectfilesinsystemviewer(\$filepath,\$permissions)

Note: *FileOps.\$writeentirefile()* and *FileOps.\$readentirefile()* are obsolete and have been removed from the Catalog; use their equivalent functions *FileOps.\$writefile()* and *FileOps.\$readfile()* as above, or the dynamic functions below.

FileOps External Object

Dynamic Functions (Methods)

The FileOps dynamic functions allow you to manipulate files. To use these methods, you must declare an object variable, and set its subtype to the FileOps external object. You then call the method using the syntax *object.\$methodname()*. You can use the Interface Manager to view the methods of the object.

The FileOps external object has the following methods.

\$closefile() *\$createfile()* *\$createtmpfile()* *\$getfileinfo()*
\$getlasterror() *\$getposition()* *\$isopen()* *\$openfile()*
\$readcharacterfile() *\$setfileinfo()* *\$setfilesize()*
\$writecharacterfile()

FileOps Worker Objects

The FileOps Worker Objects, **FileOpsCopyWorker**, **FileOpsMoveWorker**, and **FileOpsDeleteWorker** allow you to copy, move or delete files asynchronously, which may be useful when operating on a large number of files.

To use the FileOps workers, you need to create an Object variable and set its subtype to one of the FileOps worker objects via the Select Object dialog. Alternatively, you can create an Object class and set its superclass to one of the FileOps worker objects, then create an Object variable or Object reference variable and set its subtype to the object class name. Having created the variable you can call its methods using OBJECTVAR.\$methodname.

The FileOps workers work exactly like the other workers, including the ability to run the \$init, \$start or \$run methods. They have the property \$progressinterval which sets an interval in seconds, at which progress notifications are sent to the \$progress method. \$progressinterval defaults to the minimum value of 1 second.

For the \$init method, the parameters are the same as used in the static methods of the FileOps object to copyfile, movefile or deletefile.

When \$completed is called, a row is returned with the errorCode and errorText of the action; errorCode will be 0 if there is no error, otherwise a FileOps Worker error is returned, plus the error description in the errorText column.

When the \$progress method is called (only if \$callprogress is true), a row is returned with a column name 'progress' containing an estimated number between 0 and 100 indicating the percentage progress of the file operation. Note: the Delete worker never calls \$progress, therefore progress notifications are not sent from the delete worker.

Note that if you try to cancel a FileOps worker while it is running, the main thread will be blocked until the worker finishes its job.

Financial

Functions

ann() *anna()* *cmp()*

Font Handling

Functions

fontlist() *isfontinstalled()* *textsize()*

FontOps

Functions

FontOps.\$createFontOps() *FontOps.\$deleteFontOps()* *FontOps.\$listfonts()*
FontOps.\$widthFontOps() *FontOps.\$textwidth()*

General

Functions

alt() *basecolor()* *callprivate()* *cmd()*

complementary() *color()* *errcode()* *errtext()*

eval() *evalf()* *flag()* *format()*

getenv() *hexcolor()* *hsla()* *iconidwithbadge()*

iconurl() *isclear()* *isclient()* *isheadless()*

isleopard() *islion()* *ismountainlion()* *isll()*

issnowleopard() *issta()* *iswindows10()* *iswindows11()*

iswindows7() *iswindows8()* *iswindows81()* *iswindowsserver()*

listenv() *msgcancelled()* *hot()* *pathsep()*

platform() *printf()* *putenv()* *rgb()*

rgba() *shift()* *sleep()* *sys()*

systemversion() *trunclog()* *truergb()*

JavaObjs Library

Functions

Support for the JavaObjs Library functions `$addclass()` and `$resetclasscache()` has been removed in Studio 10, but the library is available from support if required.

JavaObjs *JavaObjs*
Li- *Li-*
brary.\$addclass() *brary.\$resetclasscache()*

List

Functions

avgc() *cnubintolist()* *list()* *lst()*

maxc() *minc()* *row()* *stddevc()*

tot() *totc()*

Logarithmic

Functions

exp() *ln()* *log()*

Lookup

Functions

lookup() *max()* *min()* *pick()*

Mouse

Functions

These functions are for desktop apps using Window classes only, not web or mobile apps.

mousedn() *mouseover()* *mouseup()* *rmousedn()*

rmouseup()
