F(x) 2.0 Beta 5 - Functions and Interfaces

Please note that function and interface list included in this document may be outdated. For the up-to-date list of functions and interfaces please refer to http://developer.symbianfx.com web site.

1 Parameter and return types

Basic parameter and return type notations used in function and interface descriptions:

<variant> - any data type (including arrays)
<array> - multidimensional array of variants
<matrix> - two dimensional array
<number> - floating point or an integer number
<integer> - integer number
<float> - real floating point number
<real> - real floating point number
<complex> - complex number
<boolean> - 0 or 1 integer
<position> - an integer indicating position value
<string> - string value
<timestamp> - 64 bit integer encoded date and time value
<handle> - internal system integer value
<resource> - array of internal system values

2 Interfaces

2.1 Communications

@send

Send value via device messaging system (SMS, E-Mail).
Type: Output

2.2 Date & Time

@date

Date display and selection.
Type: Input & Output
Returns: <timestamp>

@datetime

Date and time display and selection.
Type: Input & Output
Returns: <timestamp>

@duration

Duration display and selection.
Type: Input & Output
Returns: <seconds>

@time

Time display and selection.
Type: Input & Output
Returns: <timestamp>

2.3 Generic

@address

IPv4 formatted address display and entry.
Type: Input & Output
Returns: <integer (32 bit unsigned)>

@array

Array display.
Type: Output

@button

Button control.
Type: Input
Returns: <boolean>

@check

Checkbox control.
Type: Input & Output
Returns: <boolean>

@color

Solid color display.
Type: Output

@edit

Expression editor.
Type: Input & Output
Returns: <variant>

@enum <array of variants>

Value enumeration (Single choice selection).
Type: Input
Returns: <variant>

@slider [<min>] [<max>] [<step>]

Interactive slider control.
Type: Input & Output
Returns: <float>

@string

String editor.
Type: Input & Output
Returns: <string>

@text

Multi-line text box.
Type: Output

@unit <category> <default/target unit>

Unit value input.
Type: Input & Output
Returns: <value in default units>

2.4 Graph & Chart

@bar <maximum value>

Bar chart representation of array data.
Type: Output

@graph [<min x>] [<max x>] [<min y>] [<max y>]

Graph representation of one dimensional array.
Type: Output

@graph2d [<min x>] [<max x>] [<min y>] [<max y>]

Graph representation of two dimensional array.
Type: Output

@histogram <maximum value>

Histogram of array values.
Type: Output

@pie

Pie chart representation of array data.
Type: Output

@plot [<min x>] [<max x>] [<min y>] [<max y>]

Plot of values in one dimensional array.
Type: Output

@plot2d [<min x>] [<max x>] [<min y>] [<max y>]

Plot of values in two dimensional array.
Type: Output

@stem [<min x>] [<max x>] [<min y>] [<max y>]

Stem graph representation of one dimensional array.
Type: Output

2.5 Tables

@table

Table row selector.
Type: Input
Returns: <table resource>

@tselect [<string folder>]

Creates popup selector that lists all tables in the same folder as the applet.
Type: Input
Returns: <string table name>

3 Functions

3.1 Arrays

append(<array>, <variant to append>)

Returns: <array>
Append a variant to an array.

Variant and array must be of the same type.

count(<array>)

Returns: <number>
Number of elements in one dimentional array.

fill(<array>, <variant>)

Returns: <array>
Fill array with provided value.

Supplied variant must match the array type.

insert(<array>, <position>, <variant to set>)

Returns: <array>
Insert an element in the array.

Variant and array must be of the same type.

insert(<array>, <position>, <variant to insert>, <number of elements>)

Returns: <array>
Insert the same element in the array multiple times.

max(<array>)

Returns: <number>
Find the largest element of the array.

min(<array>)

Returns: <number>
Find the smallest element of the array.

nearest(<array>, <number>)

Returns: <position>
Find the array element closest to the provided value.

remove(<array>, <position>)

Returns: <array>
Remove an element from array.

remove(<array>, <position>, <number of elements>)

Returns: <array>
Remove a number of elements from the array.

set(<array>, <position>, <variant to set>)

Returns: <array>
Set array element to the supplied variant.

set(<array>, <position>, <variant to set>, <number of elements>)

Returns: <array>
Set a range of array elements to supplied value.

sort_asc(<argv>)

Returns: <array>
Sort array elements in ascending order.

sort_desc(<array>)

Returns: <array>
Sort array elements in descending order.

3.2 Date & Time

daydelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of days between two specified timestamps.

dayofmonth(<timestamp>)

Returns: <integer>
Retreive the day of month specified in the timestamp (0..N).

dayofweek(<timestamp>)

Returns: <integer>
Retreive day of the week specified in the timestamp (0..6).

dayofyear(<timestamp>)

Returns: <integer>
Retreive the day of the year specified in the timestamp (0..364).

daysinmonth(<timestamp>)

Returns: <integer>
Retreive amount of days in the month specified in the timestamp.

daysinmonth(<integer year>, <integer month>)

Returns: <integer>
Amount of days in the specified month.

getday(<timestamp>)

Returns: <integer>
Retreive day value from a timestamp.

gethour(<timestamp>)

Returns: <integer>
Retreive hour value from a timestamp.

getmicrosec(<timestamp>)

Returns: <integer>
Retreive microseconds value from a timestamp.

getminute(<timestamp>)

Returns: <integer>
Retreive minute value from a timestamp.

getmonth(<timestamp>)

Returns: <integer>
Retreive month value from a timestamp.

getsecond(<timestamp>)

Returns: <integer>
Retreive seconds value from a timestamp.

getyear(<timestamp>)

Returns: <integer>
Retreive year value from a timestamp.

hometime

Returns: <timestamp>
Obtain current home time.

hourdelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of hours between two specified timestamps.

hourdeltatest(<timestamp>, <timestamp>)

Returns: <boolean>
Test if amount of hours between two timestamps is too large for processing.

Returns TRUE if value is too large for processing, FALSE otherwise;

Hour distance processing is calculated from two supplied timestamps using 32bit integer processing. This function tests whether the different between two timestamps in hours is too great to fit into a 32 bit integer.

This function is intended for testing timestamps before supplying them to hourdelta() since if the distance is too great, hourdelta() will cause a generic "Value too Large" applet error.

isleapyear(<timestamp>)

Returns: <number>
Determines if specified year is a leap year.

leapyearsdelta(<timestamp>, <timestamp>)

Returns: <number>
Amount of leap years between two timestamps.

leapyearsupto(<timestamp>)

Returns: <number>
How many leap years have passed from 0 AD up to the supplied timestamp.

maxtime

Returns: <timestamp>
Obtains maximum supported time value (timestamp).

microsecdelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of microseconds between two specified timestamps.

mintime

Returns: <timestamp>
Obtains minimum supported time value (timestamp).

minutedelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of minutes between two specified timestamps.

minutedeltatest(<timestamp>, <timestamp>)

Returns: <boolean>
Test if amount of minutes between two timestamps is too large for processing.

Returns TRUE if value is too large for processing, FALSE otherwise;

Minute distance processing is calculated from two supplied timestamps using 32bit integer processing. This function tests whether the different between two timestamps in minutes is too great to fit into a 32 bit integer.

This function is intended for testing timestamps before supplying them to minutedelta() since if the distance is too great, minutedelta() will cause a generic "Value too Large" applet error.

monthdelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of months between two specified timestamps.

seconddelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of seconds between two specified timestamps.

seconddeltatest(<timestamp>, <timestamp>)

Returns: <boolean>
Test if amount of seconds between two timestamps is too large for processing.

Returns TRUE if value is too large for processing, FALSE otherwise;

Second distance processing is calculated from two supplied timestamps using 32bit integer processing. This function tests whether the different between two timestamps in seconds is too great to fit into a 32 bit integer.

This function is intended for testing timestamps before supplying them to seconddelta() since if the distance is too great, seconddelta() will cause a generic "Value too Large" applet error.

strtots(<string>)

Returns: <timestamp>
Convert readable string into timestamp.

Post 2.0 beta 5

The string may contain the date only, the time only, the date followed by the time, or the time followed by the date. When both the date and time are specified in the string, they should be separated using one or more space characters.

Leading zeros and spaces preceding any time or date components are discarded.

Dates may be specified either with all three components (day, month and year), or with just two components; for example month and day. The date suffix ("st" "nd" "rd" or "th") may not be included in the string.

The date and its components may take different forms:

• The month may be represented by text or by numbers.

• European (DD/MM/YYYY), American (MM/DD/YYYY) and Japanese (YYYY/MM/DD) date formats are supported. An exception to this ordering of date components occurs when European or American formatting is used and the month is represented by text. In this case, the month may be positioned in either the first or second field. When using Japanese date format, the month, whether text or numbers, must always be the second field.

• The year must be four digits.

• Any of the following characters may be used as the date separator: /(slash) – (dash) , (comma), spaces, or either of the date separator characters configured in the device locale settings. Other characters are illegal.

If a colon or a dot has been specified in the device locale settings as the date separator character, neither may be used as date separators in this function.

If specified, the time must include the hour, but both minutes and seconds, or seconds alone may be omitted.

The time and its components may take different forms:

• An am/pm time suffix may be appended to the time. If 24 hour clock format is in use, this text will be ignored.

• The am/pm suffix may be abbreviated to "a" or "p".

• Any of the following characters may be used as the time separator: :(colon) .(dot) or either of the time separator characters configured in the device locale settings. Other characters are illegal.

When a character can be interpreted as either a date or time separator character, this function will interpret it as a date separator.

Look out for cases in which wrongly interpreting the contents of a string, based on the interpretation of separator characters, causes an error. For example, trying to interpret "5.6.1996" as a time is invalid and will produce an error because 1,996 seconds is out of range.

Notes:

• The entire content of the descriptor must be valid and syntactically correct, or an error will be produced and invalid time will be returned. So, excepting whitespace, which is discarded, any trailing characters within the descriptor which do not form part of the date or time are illegal.

• If no time is specified in the string, the hours, minutes and seconds will be all set to zero, corresponding to midnight at the start of the day specified in the date. If no date is specified, each of date components are set to zero.

timestamp(<year>, <month>, <day>, <hour>, <minute>, <second>)

Returns: <timestamp>
Creates timestamp by specifying year, month, day, hour, minute, second values.

timestamp(<year>, <month>, <day>, <hour>, <minute>, <second>, <microsecond>)

Returns: <timestamp>
Creates timestamp by specifying year, month, day, hour, minute, second and microsecond values.

tstostr(<timestamp>, <format string>)

Returns: <string>
Format timestamp as readable string.

Post 2.0 beta 5

Puts the supplied timestamp into a string and formats it according to the format string specified in the second argument. Many of the formatting commands use the system's locale settings for the date and time, for instance the characters used to separate components of the date and time and the ordering of day, month and year. The list of formatting commands below is divided into two sections, the first of which lists the commands which operate without reference to the locale's date and time settings and the second table lists the commands which do use these settings.

The following formatting commands do not honour the locale-specific system settings:

• %% : Include a single '%' character in the string

• %* : Abbreviate following item (the following item should not be preceded by a '%' character).

• %C : Interpret the argument as the six digit microsecond component of the time. In its abbreviated form, ('%*C') this should be followed by an integer between zero and six — the integer indicates the number of digits to display.

• %D : Interpret the argument as the two digit day number in the month. Abbreviation suppresses leading zero.

• %E : Interpret the argument as the day name. Abbreviation is language-specific (e.g. English uses the first three letters).

• %F : Use this command for locale-independent ordering of date components. This orders the following day/month/year component(s) (%D, %M, %Y for example) according to the order in which they are specified in the string. This removes the need to use %1 to %5 (described below).

• %H : Interpret the argument as the one or two digit hour component of the time in 24 hour time format. Abbreviation suppresses leading zero. For locale-dependent hour formatting, use %J.

• %I : Interpret the argument as the one or two digit hour component of the time in 12 hour time format. The leading zero is automatically suppressed so that abbreviation has no effect. For locale-dependent hour formatting, use %J.

• %M : Interpret the argument as the one or two digit month number. Abbreviation suppresses leading zero.

• %N : Interpret the argument as the month name. Abbreviation is language specific, e.g. English uses the first three letters only. When using locale-dependent formatting, (that is, %F has not previously been specified), specifying %N causes any subsequent occurrence of a month specifier in the string to insert the month as text rather than in numeric form. When using locale-independent formatting, specifying %N causes the month to be inserted as text at that position, but any subsequent occurrence of %M will cause the month to be inserted in numeric form.

• %S : Interpret the argument as the one or two digit seconds component of the time. Abbreviation suppresses leading zero.

• %T : Interpret the argument as the one or two digit minutes component of the time. Abbreviation suppresses leading zero.

• %W : Interpret the argument as the one or two digit week number in year. Abbreviation suppresses leading zero.

• %X : Interpret the argument as the date suffix. Cannot be abbreviated. When using locale-dependent formatting (that is, %F has not previously been specified), %X causes all further occurrences of the day number to be displayed with the date suffix. When using locale-independent formatting, a date suffix will be inserted only after the occurrence of the day number which %X follows in the format string. Any further occurrence of %D without a following %X will insert the day number without a suffix.

• %Y : Interpret the argument as the four digit year number. Abbreviation suppresses the first two digits.

• %Z : Interpret the argument as the one, two or three digit day number in the year. Abbreviation suppresses leading zeros.

The following formatting commands do honour the locale-specific system settings:

• %. : Interpret the argument as the decimal separator character (as set in device locale settings). The decimal separator is used to separate seconds and microseconds, if present.

• %: : Interpret the argument as one of the four time separator characters (as set in device locale settings). Must be followed by an integer between zero and three inclusive to indicate which time separator character is being referred to.

• %/ : Interpret the argument as one of the four date separator characters (as set in device locale settings). Must be followed by an integer between zero and three inclusive to indicate which date separator character is being referred to.

• %1 : Interpret the argument as the first component of a three component date (i.e. day, month or year) where the order has been set in device locale settings. When the date format is European, this is the day, when American, the month, and when Japanese, the year. For more information on this and the following four formatting commands, see the Notes section immediately below.

• %2 : Interpret the argument as the second component of a three component date where the order has been set in device locale settings. When the date format is European, this is the month, when American, the day and when Japanese, the month.

• %3 : Interpret the argument as the third component of a three component date where the order has been set in device locale settings. When the date format is European, or American this is the year and when Japanese, the day.

• %4 : Interpret the argument as the first component of a two component date (day and month) where the order has been set in device locale settings. When the date format is European this is the day, and when American or Japanese, the month.

• %5 : Interpret the argument as the second component of a two component date (day and month) where the order has been set in device locale settings. When the date format is European this is the month, and when American or Japanese, the day.

• %A : Interpret the argument as "am" or "pm" text according to the current language and time of day. Unlike the %B formatting command (described below), %A disregards the locale's 12 or 24 hour clock setting, so that when used without an inserted + or – sign, am/pm text will always be displayed. Whether a space is inserted between the am/pm text and the time depends on the locale-specific settings. However, if abbreviated (%*A), no space is inserted, regardless of the locale's settings. The am/pm text appears before or after the time, according

to the position of the %A, regardless of the locale-specific settings. For example, the following ordering of formatting commands causes am/pm text to be printed after the time: %H %T %S %A. Optionally, a minus or plus sign may be inserted between the "%" and the "A". This operates as follows: —

• %-A causes am/pm text to be inserted into the descriptor only if the am/pm symbol position has been set in the locale to be displayed before. Cannot be abbreviated using asterisk.

• %+A causes am/pm text to be inserted into the descriptor only if the am/pm symbol position has been set in the locale to be displayed after. Cannot be abbreviated using asterisk. For example, the following formatting commands will cause am/pm text to be displayed after the time if the am/pm position has been set in the locale to be displayed after or before the time if locale has been set to display it before: %–A %H %T %S %+A.

• %B Interpret the argument as am or pm text according to the current language and time of day. Unlike the %A command, when using %B, am/pm text is displayed only if the clock setting in the locale is 12-hour. Whether a space is inserted between the am/pm text and the time depends on the locale-specific settings. However, if abbreviated (%*B), no space is inserted, regardless of the locale's settings. The am/pm text appears before or after the time, according to the location of the "%B", regardless of the locale-specific settings. For example, the following formatting commands cause am/pm text to be printed after the time: %H %T %S %B. Optionally, a minus or plus sign may be inserted between the "%" and the "B". This operates as follows: —

• %-B causes am/pm text to be inserted into the descriptor only if using a 12 hour clock and the am/pm symbol position has been set in the locale to be displayed before. Cannot be abbreviated using asterisk.

• %+B causes am/pm text to be inserted into the descriptor only if using a 12 hour clock and the am/pm symbol position has been set in the locale to be displayed after. Cannot be abbreviated using asterisk. For example, the following formatting commands cause am/pm text to be printed after the time if the am/pm position has been set in the locale to be displayed after or before the time if position in the locale is set to before: %–B %H %T %S %+B.

• %J Interpret the argument as the hour component of the time in either 12 or 24 hour clock format depending on the locale's clock format setting. When the clock format has been set to 12 hour, leading zeros are automatically suppressed so that abbreviation has no effect. Abbreviation suppresses leading zero only when using a 24 hour clock.

Notes:

The %1, %2, %3, %4 and %5 formatting commands are used in conjunction with %D, %M and %Y to format the date locale-dependently. When formatting the date locale-dependently, the order of the day, month and year components within the string is determined by the order of the %1 to %5 formatting commands, not that of %D, %M, %Y.

When formatting the date locale-independently (that is, %F has been specified in the format string), the %1 to %5 formatting commands are not required, and should be omitted. In this case, the order of the date components is determined by the order of the %D, %M, %Y format commands within the format string.

Up to four date separators and up to four time separators can be used to separate the components of a date or time. When formatting a numerical date consisting of the day, month and year or a time containing hours, minutes and seconds, all four separators should always be specified in the format command string. Usually, the leading and trailing separators should not be displayed. In this case, the first and fourth separators should still be specified, but should be represented by a null character.

The date format follows the pattern:

DateSeparator[0] DateComponent1 DateSeparator[1] DateComponent2 DateSeparator[2] DateComponent3 DateSeparator[3]

where the ordering of date components is determined by the locale's date format setting.

The time format follows the pattern:

TimeSeparator[0] Hours TimeSeparator[1] Minutes TimeSeparator[2] Seconds TimeSeparator[3]

If the time includes a microseconds component, the third separator should occur after the microseconds, and the seconds and microseconds should be separated by the decimal separator. When formatting a two component time, the following rules apply:

• if the time consists of hours and minutes, the third time delimiter should be omitted

• if the time consists of minutes and seconds, the second time delimiter should be omitted

universaltime

Returns: <timestamp>
Obtain current universal time.

weekofyear(<timestamp>)

Returns: <integer>
Obtain sequential week of the year number from timestamp (0..51)

yeardelta(<timestamp>, <timestamp>)

Returns: <integer>
Amount of years between two timestamps.

3.3 Execution Control

activate(<string applet name>)

Loads specified applet in the workspace.

error(<constant || string>)

Report error.

This function will terminate applet execution and report the error to the user.

User can supply an error string or one of the following system constants:

E_INPUT-Invalid User Input
E_RANGE-Out of Range
E_INSUFF-Insufficient Input
E_PARAMS-Too Many Parameters.

exec(<string applet name>, <array parameters>)

Returns: <variant>
Execute external applet and return calculated result.

This function executes specified external applet and returns the variant resulting from this applet calculation. This function provides an ability to invoke the external applet as a function.

notify(<string>)

Sends notification message to the user.

Notification message usually appears at the top of the screen.

range(<number>, <number min>, <number max>)

Check value range.

range(,,

This function will terminate formula execution with the “Out of Range” error if the supplied value is outside of the supplied minimum and maximum boundaries.

respawn( )

Schedules the applet to be executed after it exits.

Post 2.0 beta 4

This function schedules execution of the applet after it's current execution run has been finished. This is useful to continiously run an applet or have it reflect ui changes performed by current execution.

3.4 Generic

isarray(<variant>)

Returns: <boolean>
Test if the variable is an array.

isarray2d(<variant>)

Returns: <boolean>
Tests if the variable is a 2d array.

This function returns TRUE if supplied variant is a 2 dimensional array, FALSE otherwise.

isfi(<number>)

Returns: <boolean>
Test if the value is finite.

Parameters:

n - value to be tested.

isfloat(<variant>)

Returns: <boolean>
Test if the variable is a floating point variable.

isinf(<number>)

Returns: <boolean>
Test for infinite (Inf)

Test if the value is Inf.

Parameters:

n - value to be tested.

isinteger(<variant>)

Returns: <boolean>
Test if the variable is an integer.

isnan(<number>)

Returns: <boolean>
Test for NaN.

Test if the value is NaN.

Parameters:

n - value to tbe tested.

isnumeric(<variant>)

Returns: <boolean>
Test if the variable is numeric (integer or floating point).

isstring(<variant>)

Returns: <boolean>
Test if the variable is a string.

iszero(<number>)

Returns: <boolean>
Test for zero value.

Test if the value is equal to zero.

Parameters:

n - value to be tested.

This function returns TRUE if the value is 0.0 and FALSE otherwise.

3.5 Math - Generic

abs(<real or complex>)

Returns: <real or complex>
Absolute value function

The abs() function computes the absolute value of x.

Parameters:

x - real or complex.

The abs() function returns the absolute value of x.

acos(<real or complex>)

Returns: <real or complex>
Arc cosine function

The acos() function computes the principal value of the arc cosine of x in the range [0, PI]

Parameters:

x - Real or complex value.

if real |x| > 1, acos(x) will return NaN.

acosh(<real or complex>)

Returns: <real or complex>
Calculates inverse hyperbolic cosine of the argument.

angle(<complex>)

Returns: <real>
Calculates an angle from the complex number.

This function calculates an angle from the complex number by calling atan2(r,i).

asin(<real or complex>)

Returns: <real or complex>
Arc Sine function

The asin() function computes the principal value of the arc sine of x in the range [-1, +1].

Parameters:

x - Real or complex value

The asin() function may return NaN in case of an error.

asinh(<real or complex>)

Returns: <real or complex>
Calculates inverse hyperbolic arc sine of the argument.

atan(<real or complex>)

Returns: <real or complex>
Arc tangent function

The atan() function computes the principal value of the arc tangent of x in the range [-PI/2, +PI/2].

Parameters:

x - Is the real or complex value from which to derive the principle value of the arc tangent.

The atan() function may return NaN in case of an error.

atan2(<real>, <real>)

Returns: <real>
Arc tangent function of two variables.

The atan2() function computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

Parameters:

x - Is the numerator, y - Is the denominator

The atan2() function returns the arc tangent of y/x, in the range [-PI, +PI] radians. NaN may be returned in case of an error.

atanh(<real or complex>)

Returns: <real or complex>
Calculates inverse hyperbolic tangent of the argument.

ceil(<float>)

Returns: <float>
Round to smallest integral value greater than or equal to x

The ceil() function returns the smallest integral value greater than or equal to x

Return Value:nceil() returns the rounded up value based on x.

cinv(<complex>)

Returns: <complex>
Calculates inverse of a complex number.

This function calculates inverse of the supplied complex number: 1/a

clip(<number>, <number min>, <number max>)

Returns: <number>
Clamps a value within a given range.

Parameters:

x, y1, y2

This function will clamp the value of x to be within the provided values of y1 and y2. This is an equivalent of the following code:

```
if(x < y1) x = y1;
if(x > y2) x = y2;
```

conjg(<complex>)

Returns: <complex>
Returns conjugate of the complex number.

This function returns the conjugate of the supplied complex number.

The conjugate of the complex number (r, i) is (r, -i).

copysign(<number>, <number signed>)

Returns: <number>
Returns one value with the sign of another.

Parameters:

x - floating point value to be changed, y - floating point value the sign of which is to be taken.

copysign() returns its floating-point argument x with the same sign as its floating-point argument y.

cos(<real or complex>)

Returns: <real or complex>
Cosine function

The cos() function computes the cosine of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Parameters:

x - Is an angle in radians.

Return Value:

The cos() function returns the cosine value. If x is NaN, NaN is returned. If the return value would underflow, 0.0 is returned.

cosh(<real or complex>)

Returns: <real or complex>
Hyperbolic cosine function

The cosh() function computes the hyperbolic cosine of x.

Parameters:

x - Is the value to derive the hyperbolic cosine from.

Ths cosh() function may return NaN in case of an error.

deg(<float>)

Returns: <float>
Convert radians to degrees.

Parameters:

x - An angle in radians.

Returns an angle in degrees.

dist(<real or complex array>, <real or complex array>)

Returns: <real>
Calculates distance between two N element vectors.

This function calculates distance between two N element vectors.

This is equivalent to:

sqrt((X1-Y2)^2 + (X2-Y2)^2 ... + (Xn-Yn)^2);

dtor(<float>)
Returns: <float>
Convert degrees to radians.

exp(<real or complex>)
Returns: <real or complex>
Exponential functions

exp(), expm1()

The exp() function computes the exponent of x. The expm1() function computes the exponent of x - 1.0

Parameters:

x - floating point value.

The exp() and expm1() functions may return INF in case of an overflow.

expm1(<real or complex>)
Returns: <real or complex>
Exponential functions

exp(), expm1()

The exp() function computes the exponent of x. The expm1() function computes the exponent of x - 1.0

Parameters:

x - floating point value.

The exp() and expm1() functions may return INF in case of an overflow.

floor(<float>)
Returns: <float>
Round to the largest integral value not greater than x

The floor() function returns the largest integral value less than or equal to x.

Parameters:

x - Is the number whose floor to compute.

Return Values:

The floor() function returns the largest integral value not greater than x, expressed as a double.

fmod(<float>, <float>)
Returns: <float>
Floating point remainder.

fmod() returns the floating point remainder of x / y.

Parameters:

x, y - floating point values.

fmod() function calculates the floating-point remainder f of x / y such that x = i * y + f, where i is an integer, f has the same sign as x, and the absolute value of f is less than the absolute value of y.

fmod() may return NaN if x is 0.0 or an error has occured.

frac(<float>)
Returns: <float>
Calculates the fractional part of a number.

The fractional part is that after a decimal point. Truncation is toward zero, so that frac(2.4) == 0.4, frac(2) == 0, frac(-1) == 0, frac(-1.4) == 0.4.

Parameters:

n - The number whose fractional part is required.

imag(<complex>)

Returns: <float>
Returns imaginary part of the supplied complex number.

Post 2.0 beta 5

length(<real or complex array>)

Returns: <real>
Calculates length of the N element vector.

This function calculates the length of the N element vector.

This is equivalent to:

sqrt(X1*X1+X2*X2...+Xn*Xn);

length2(<float>, <float>)

Returns: <float>
2d Length

Parameters: x, y

Calculates the length of 2d vector.

This calculation is equivalent to: sqrt(x*x+y*y).

length3(<float>, <float>, <float>)

Returns: <float>
3d Length

Parameters: x, y, z

Calculates the length of 3d vector.

This calculation is equivalent to: sqrt(x*x+y*y+z*z).

lerp(<number x1>, <number x2>, <float factor n 0.0..1.0>)

Returns: <number>
Interpolation function: r = x1+n*(x2-x1);

Parameters:

x1, x2, n

This function interpolates between x1 and x2 based on the n value which should range from 0.0 to 1.0 (values above 1 or below 0 will mean extrapolation). This function is equivalent to: x1+n*(x2-x1)

log(<real or complex>)

Returns: <real or complex>
Compute natural logarithm

The log() function computes natural logarithm of argument x.

Parameters:

x - Is the number whose logarithm is to be computed.

NaN may be returned if x is NaN or in case of an error.

log10(<real or complex>)

Returns: <real or complex>
Compute natural logarithm

The log10() function cmputes natural logarithm of argument x to base 10.

Parameters:

x - Is the number whose logarithm is to be computed.

NaN may be returned if x is NaN or in case of an error.

max(<number>, <number>)

Returns: <number>
Maximum of two values.

Parameters:

n1, n2.

min(<number>, <number>)

Returns: <number>
Minimum of two values.

Parameters:

n1, n2.

norm(<float>, <float>, <float>)

Returns: <float>
Normalize value.

Parameters:

x, y1, y2

This function will normalize value x to the y2-y1 range as follows: x = (x - y1) / (y2-y1).

pow(<real or complex x>, <real or complex y>)

Returns: <real or complex>
Returns x raised to the power of y.

Parameters:

x - Base, y - Exponent.

pow10(<real or complex>)

Returns: <real or complex>
10 to the power of x

Parameters:

exp - The power to which 10 is raised.

rad(<float>)

Returns: <float>
Convert degrees to radians.

Parameters:

x - An angle in degrees.

Returns an angle in radians.

rand

Returns: <float>
Generate random number between 0.0 and 1.0.

The rand() function generates pseudo-random numbers in the range from 0.0 to 1.0. The function uses a seed value to generate the pseudo-random number. The seed has an initial value of 1 but it can be changed by calling srand().

real(<complex>)

Returns: <float>
Returns real part of the supplied complex number.

Post 2.0 beta 5

round(<number || number array>)

Returns: <integer>
Round the value to the nearest integer.

rounds to the nearest integer or to the nearest specified decimal place.

Example:

round(1.6) results in 2
round(1.4) results in 1
round(1.26,1) results in 1.3
round(1.24,1) results in 1.2.

round(<number || number array>, <number of decimal places>)

Returns: <float>
Rount the value to the specified decimal point.

rtod(<float>)

Returns: <float>
Convert radians to degrees.

sample(<expression string>, <start>, <end>, <number of samples>)

Returns: <array>
Samples expression N times within X1..X2 range.

sample() function allows sampling of a given expression for a set of values. On input this function takes an expression to be evaluated, the starting value of the evaluation, the ending value and the amount of samples. On output this function returns an array with evaluated samples.

Example:

sample("sin(x)*0.5",-PI,PI,25)

The above function call will return an array of 25 elements that will represent the sampling of the supplied expression between –PI and PI values.

The variable X is automatically created in the applet that uses sample() function. If user creates variable X in the applet, it will be used as a sampling range counter during sampling.

The outside environment of the compiler is visible to the sampled expression, thus all variables that are currently declared in the applet can be used within the specified expression:

in float MyValue;

out float graph[] = sample(“cos(x)*MyValue”,-5,5,10);

scalbn(<number>, <integer>)

Returns: <float>
Multiply by a power of machine's radix.

The scalbn() function computes x * r^n, where r is the radix of the machine's floating point arithmetic.

Parameters:

x - Is the value of the scale, y - Is the value of exponent.

Return Values: On success, scalbn() returns x * r^n; scalbn() may return NaN or +/- Inf in case of an error.

sin(<real or complex>)

Returns: <real or complex>
Sine function

The sin() function computes the sine of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Parameters:

x - Is the number whose sine is to be computed.

Return Values:

On success, sin() returns the sine of x. If x is NaN or ±Inf, NaN is returned. If it underflows, 0.0 is returned.

sinh(<real or complex>)

Returns: <real or complex>
Hyperbolic sine function

The sinh() function cmputes the hyperbolic sine of x.

Parameters:

x - Is the number whose hyperbolic sine is to be computed.

sinh() may return NaN in case of an error.

sqr(<real or complex>)

Returns: <real or complex>
Square of the value

Returns the value squared (x*x).

sqrt(<real or complex>)

Returns: <real or complex>
Square root of the number

Returns a square root of the number.Parameters:

x - nonnegative integer or floating point number.

srand(<integer>)

Set random seed.

srand() set the seed for a new sequence of pseudo-random numbers.

Parameters:

x - The random seed value.

You can use rand() function to obtain a random value based on the set seed value.

To acheive a random seed for each formula evaluation, you can use the time() function to generate random seed values: srand(time());

tan(<real or complex>)

Returns: <real or complex>
Tangent function

The tan() function computes the tangent of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Parameters:

x - Is the value whose tangent to be computed.

The tan() function returns the tangent value.

tanh(<real or complex>)

Returns: <real or complex>
Hyperbolic tangent function

The tanh() function computes the hyperbolic tangent of x.

Parameters:

x - Is the value whose hyperbolic tangent is to be computed.

The tanh() function may return NaN in case of an error.

3.6 Math - Matrices

maddcols(<matrix>, <source col>, <dest col>, <float scale factor>)

Returns: <matrix>
Adds source column to the destination column scaled by the factor.

Post 2.0 beta 5

Dn = Dn + Sn*F

maddrows(<matrix>, <source row>, <dest row>, <float scale factor>)

Returns: <matrix>
Adds source row to the destination row scaled by the factor.

Post 2.0 beta 5

Dn = Dn + Sn*F

mclr(<matrix>)

Returns: <matrix>
Clears the matrix.

Post 2.0 beta 5

Resets all the values of the matrix to 0.0.

mclrcol(<matrix>, <integer col number>)

Returns: <matrix>
Clears the matrix column.

Post 2.0 beta 5

Resets the column values of the supplied matrix. Returns the resulting matrix.

mclrrow(<matrix>, <integer row number>)

Returns: <matrix>
Clears the matrix row.

Post 2.0 beta 5

Resets all the values in the given row of the matrix. Returns the resulting matrix.

mcmac(<matrix A>, <matrix B>)

Returns: <matrix>
Concatenate matrix as columns.

Post 2.0 beta 5

Concatenates matrix B on the bottom of the matrix A.

mcmar(<matrix A>, <matrix B>)

Returns: <matrix>
Concatenate matrix as rows.

Post 2.0 beta 5

Concatenates matrix B on the right of the matrix A.

mcovar(<matrix>)

Returns: <matrix>
Create covariant of the matrix.

Post 2.0 beta 5

mdet(<matrix>)

Returns: <float>
Finds determinant of a matrix.

Post 2.0 beta 5

mdivcol(<matrix>, <integer column>, <float>)

Returns: <matrix>
Divide all elements of the column by the supplied value.

Post 2.0 beta 5

mdivrow(<matrix>, <integer row>, <float>)

Returns: <matrix>
Divide all elements of the row by the supplied value.

Post 2.0 beta 5

mfill(<matrix>, <float value>)

Returns: <matrix>
Initializes all matrix elements to the supplied value.

Post 2.0 beta 5

Same functionality as fill() array function.

mfillcol(<matrix>, <integer col number>, <float init value>)

Returns: <matrix>
Sets all values in the column of the matrix to the supplied value.

Post 2.0 beta 5

mfillrow(<matrix>, <integer row number>, <float init value>)

Returns: <matrix>
Sets all values in the row of the matrix to the supplied value.

Post 2.0 beta 5

mgetcolmax(<matrix>, <integer col number>)

Returns: <float>
Finds the largest value in the column of the matrix.

Post 2.0 beta 5

mgetcolmin(<matrix>, <integer col number>)

Returns: <float>
Finds the smallest value in the column of the matrix.

Post 2.0 beta 5

mgetcolrange(<matrix>, <integer col number>)

Returns: <float>
Returns the difference between the smallest and the largest value in the supplied column of the matrix.

Post 2.0 beta 5

mgetmax(<matrix>)

Returns: <float>
Finds the largest value in the matrix.

Post 2.0 beta 5

Same functionality as max() operating on array.

mgetmin(<matrix>)

Returns: <float>
Finds the smallest value in the matrix.

Post 2.0 beta 5

Same functionality as min() operating on array.

mgetrange(<matrix>)

Returns: <float>
Get the difference between the largest and the smallest values in the matrix.

Post 2.0 beta 5

mgetrowmax(<matrix>, <integer row number>)

Returns: <float>
Finds the largest value in the row of the matrix.

Post 2.0 beta 5

mgetrowmin(<matrix>, <integer row number>)

Returns: <float>
Finds the smallest value in the row of the matrix.

Post 2.0 beta 5

mgetrowrange(<matrix>, <integer row number>)

Returns: <float>
Returns the difference between the smallest and the largest values if the given row of the matrix.

Post 2.0 beta 5

mgetsubm(<matrix>, <row start>, <column start>, <row length>, <column length>)

Returns: <matrix>
Get submatrix bounded by the supplied values.

Post 2.0 beta 5

mident(<integer size>)

Returns: <array>
Create an identity matrix of specified size.

minverse(<matrix>)

Returns: <matrix>
Returns the inverse of the supplied matrix.

Post 2.0 beta 5

mminor(<matrix>, <int row pos>, <int column pos>)

Returns: <matrix>
Returns minor around row,column element.

Post 2.0 beta 5

mmul(<matrix>, <matrix>)

Returns: <matrix>
Multiplies two supplied matrices.

mmulcol(<matrix>, <integer column>, <float>)

Returns: <matrix>
Multiplies all values in the column by the supplied value.

```
$category = $_POST["category"];
```

mmulrow(<matrix>, <integer row>, <float>)

Returns: <matrix>
Multiplies all values in the row by the supplied value.

Post 2.0 beta 5

mnorm(<matrix>, <float min>, <float max>)

Returns: <matrix>
Normalize matrix between the supplied min and max values.

Post 2.0 beta 5

mrcol(<matrix>, <integer column number>)

Returns: <matrix>
Removes the column from the matrix.

Post 2.0 beta 5

mref(<matrix>)

Returns: <matrix>
Convert the matrix into row-echelon form.

Post 2.0 beta 5

mrref(<matrix>)

Returns: <matrix>
Convert the matrix into reduced row-echelon form.

Post 2.0 beta 5

mrrow(<matrix>, <integer row number>)

Returns: <matrix>
Removes the row from the matrix.

Post 2.0 beta 5

msortasc(<matrix>)

Returns: <matrix>
Sort matrix in the ascending order.

Post 2.0 beta 5

msortdesc(<matrix>)

Returns: <matrix>
Sort matrix in the descending order.

Post 2.0 beta 5

msum(<matrix>)

Returns: <float>
Sums all the values in the matrix.

Post 2.0 beta 5

msumcol(<matrix>, <integer col number>)

Returns: <float>
Sums all the values in the column of the matrix.

Post 2.0 beta 5

msumcolsq(<matrix>, <integer col number>)

Returns: <float>
Sums all the values in the column of the matrix and returns the square of that value.

Post 2.0 beta 5

msumrow(<matrix>, <integer row number>)

Returns: <float>
Sums all the values in the row of the matrix.

Post 2.0 beta 5

msumrowsq(<matrix>, <integer row number>)

Returns: <float>
Sums all the values in the row of the matrix and returns the square of that value.

Post 2.0 beta 5

msumsq(<matrix>)

Returns: <float>
Sums all the values in the matrix and returns square of that value.

Post 2.0 beta 5

mswapcols(<matrix>, <integer column>, <integer column>)

Returns: <matrix>
Swaps two columns in the matrix.

Post 2.0 beta 5

mswaprows(<matrix>, <integer row number>, <integer row number>)

Returns: <matrix>
Swaps two rows in the matrix.

Post 2.0 beta 5

mtranspose(<matrix>)

Returns: <matrix>
Returns transpose of the matrix.

Post 2.0 beta 5

3.7 Stacks

pop(<stack handle>)

Returns: <variant>
Pop the variant from the stack.

push(<stack handle>, <variant>)

Push a variant on the stack.

stackcount(<stack handle>)

Returns: <number>
Count of elements on the stack.

stackreset(<stack handle>)

Reset all data on the stack specified by stack handle.

stacktoarray(<stack handle>)

Returns: <array>
Convert stack specified by stack handle to an array.

3.8 Strings

atof(<string>)

Returns: <float>
Convert string to a floating point value.

atoi(<string>)

Returns: <integer>
Convert string to an integer value.

currencysymbol( )

Returns: <string>
Returns current currency symbol as set in the device locale settings.

Post 2.0 beta 4

Please see tofunds() function for formatting values as currency.

explode(<string text>, <string separator>)

Returns: <string array>
Break a string into an array of strings separating the string on each occurence of the supplied separator.

Post 2.0 beta 4

Example:

str arr[] = explode("AxBxC","x");
results in:
["A" "B" "C"]

ftoa(<number>)

Returns: <string>
Convert floating point value to a string.

ftoa(<number>, <number of digits>, <number of decimal places>)

Returns: <string>
Convert formatted floating point value to a string.

implode(<array>, <string separator>)

Returns: <string>
Merges all array elements into a string separating them with the supplied string separator.

Post 2.0 beta 4

Example:
str s = implode(["A" "B" "C"],"-");
results in:
"A-B-C"

itoa(<number>)

Returns: <string>
Convert integer value to a string.

strcaps(<string>)

Returns: <string>
Capitalizes supplied string.

Post 2.0 beta 4

strlen(<string>)

Returns: <integer>
String length.

strlwr(<string>)

Returns: <string>
Convert supplied string to lower case.

strstr(<string>, <substring to search for>)

Returns: <position> or -1
Get position of the first occurence of a substring in a string or -1 if not found.

strupr(<string>)

Returns: <string>
Convert supplied string to upper case.

substr(<string>, <position>)

Returns: <string>
Get substring from a string starting from the provided position until the end of the string.

substr(<string>, <position>, <length>)

Returns: <string>
Get substring from a string starting from the provided position for the supplied length.

tofunds(<variant>)

Returns: <string>
Formats supplied variant to be represented as currency.

Post 2.0 beta 4

Supplied value (if string, converted to numeric) is formatted to be represented as currency (as done by the funds datatype.

Formatting and addition of a currency symbol is performed in accordance to the locale settings of the device.

Example:

str value = tofunds(99);

value results in "$99.00".

Currency formatting settings can be changed in the control panel application on host device. Some Series 60 devices obtain their locale settings from current language settings.

trim(<string>)

Returns: <string>
Removes leading and trailing space characters from the string.

Post 2.0 beta 4

Removes leading and trailing characters from the supplied string.

Example:

str result = trim(" hello world ");

the resulting string is "hello world"

3.9 Tables

tappend(<table handle>)

Returns: <position of new record>
Append new record to the table and obtain it's position.

tchanged(<table resource>)

Returns: <boolean>
Checks if current record in the table record selector interface has been changed because of user interaction.

tcolcount(<table handle>)

Returns: <number>
Retreive numboer of columns in the table.

Retreives number of columns in the table specified by the handle.

tcolget(<table handle>, <column position>)

Returns: <array>
Retreive a column of values as array.

Retreives a column of values as an array.
For example, if a table "tid" contains

1,a
2,b
3,c

tcolget(tid,0) will return [1 2 3]
and
tcolget(tid,1) will return ["a" "b" "c"]

tcreate(<string filename >)

Returns: <integer code>
Creates a table with supplied filename.

Post 2.0 beta 4 release.

This function creates a table with a supplied filename. tcreate() returns 1 in case of success, 0 in case of a general error and -1 if file already exists.

Example:

```
tcreate("my table");
tcreate("folder\\my table");
```

Please note that because of real-time interpretation of applets this function may be invoked on each interaction with the applet. In order to avoid this, call to this function must be enclosed in a condition.

Example:

```
bool button_create;
@button_create button "Create Table";
if(button_create)
{
tcreate("my table");
}
```

tenum([<optional subfolder>])

Returns: <array of strings>
Enumerate all table files in the applet folder of subfolder.

Post 2.0 beta 4

This function allows enumeration of tables in the current applet folder or an optionally supplied subfolder. This function returns a string array containing table names that can be used with topen() function.

If no tables have been found, an empty array is returned. Use count(array) function to determine number of tables found.

Example:

```
str tarr[] = tenum();
or
str tarr[] = tenum("subfolder");

for(int i = 0; i < count(tarr); i++)
{
// table processing goes here
}
```

texists(<string table file name>)

Returns: <boolean>
Tests for existance of a table file.

Post 2.0 beta 4

Tests is supplied table filename exists. Returns TRUE if it does and FALSE if file was not found.

Example:

```
str table_name = "data\\mytable";
if(!texists(table_name);
tcreate(table_name);
int tid = topen(table_name);
```

tname(<table handle>)

Returns: <array of strings>
Retreive array of captions for each column in the table.

tname(<table handle>, <column position>)

Returns: <string>
Return the string containing table column caption.

topen(<string table file name>)

Returns: <handle>
Open the table with specified filename and obtain the handle to it.

If table could not be opened, system reports an error and applet execution halts.

tremove(<table handle>)

Remove currently selected row.

trowcount(<table handle>)

Returns: <number>
Retreive number of rows in the table.

Retreives number of rows in the table specified by the handle.

trowget(<table resource>)

Returns: <array>
Obtain table row from the supplied table resource.

trowget(<table handle>, <row position>)

Returns: <array>
Obtain table row from the supplied table handle and row position.

trowset(<table handle>, <array>)

Set data from the array into the currently selected row.

tselect(<table resource>)

Select current row provided in the supplied table resource.

tselect(<table handle>, <row position>)

Select current row from the supplied position in the table indicated by the supplied handle.

tset(<table handle>, <string column name>, <variant>)

Set variant into the cell indicated by currently selected row and supplied column name.

3.10 Variables & UI

isinit( )

Returns: <boolean>
Tests if the applet is being executed first time after compilation.

Post 2.0 beta 4

This function returns TRUE if the applet is being executed first time after compilation. This occures when the applet loads in the view or when the user modifies the source code.

This function is particularly important when dynamically creating user interface and interface controls have to be set in the initial state.

setiv(<string || string array>, <variant || variant array>)

Set dynamic variable input value content.

This function sets the text of the dynamic input variable.

The following example resets the edit field of the input variable 's' to "Default Value" when user clicks on button 'b':

```
in str s;
in bool b;
@b button "Click here to set default value";

if(b)
{
setiv(s,"Default Value");
}
```

uihideresult( )

Hides result output area.

Post 2.0 beta 4

F(x) user interface consits of 4 areas: Variable Input, Variable Output, Result Output and Editor.

F(x) can be switched (via view menu or using Lock button on UIQ/7710) into I/O Only view, Edit Only view and I/O and Edit combined view.

The Result Output area is present when F(x) is set to Edit Only view or or there is no variable output (thus no Output view).

This function can be used to hide Result Output view when there is no Variable Output view. Doing this will leave the user with only Variable Input view.

This function can be useful when dynamically reconfiguring input controls to reflect the current state of the applet, thus presenting the user with data using Input View.

For example: An applet can track on/off state by storing data in the table. The on/off state can be switched using a trigger button. When the applet runs, it can read the current status from the table and use vsettitle() function to change the button title to "TURN ON" or "TURN OFF" according to the current state. This method effectively offers input-only user interface.

uireset( )

Destroys all dynamically created variables.

This function destroys all variables created using vcreate(), vcreatein() and vcreateout() functions.

The applet is executed on each user interaction. If the applet builds dynamic user interface the interface changed can be accumulative, i.e. each dynamically created variable will reside in the user interface until it is destroyed or the applet is closed.

To avoid the cumulative effect when creating dynamic variables user must call uireset() at the applet start to

cleanup all previously created variables.

Example:

```
uireset(); // reset user interface environment

vcreatein("i1","First Value");
vcreatein("i2","Second Value");

uiupdate(); // synchronize user interface
```

---
uiupdate() function must be called to update user interface after using this function..

uiupdate( )

Applies user interface modifications.

uiupdate() synchronizes the F(x) variable environment (new or deleted variables, their titles and interfaces) with the user interface layout.

Example:

```
vcreatein("i1","First Value");
vcreatein("i2,"Second Value");

// at this point variables exist but
// are not visible to the user.

// make them visible to the user
// by synchronizing user interface:

uiupdate();
```

vcreate(<string typename>, <string name>)

Dynamically create a variable.

Allows dynamic creation of variables in the applet environment.

Dynamically created variables can not be used by their name in the applet. However, they can be accessed via variable management functions such as vvalue() and vset().

vcreate() is typically useful only for dynamic creation of variables for temporary storage.

Example:

```
vcreate("int","my_var");
```

vcreatein(<string typename>, <string name>, <string title>)

Dynamically create input variable.

vcreatein() allows dynamic creation of the input variable. Input variables are visible in the input pane of the user interface.

```
vcreatein("int","var","Var Title");
```

is an equivalent of:

```
in int var;
@var "Var Title";
```

Dynamic variables can not be used in the applet source code by their names. They can only be accessed via variable management functions such as vvalue(), vtitle() etc.

---
uiupdate() function must be called after using vcreate() to update the user interface changes.

vcreateout(<string type>, <string name>, <string title>)

Dynamically create output variable.

vcreatein() allows dynamic creation of the output variable. Output variables are visible in the output pane of the user interface.

vcreateout("int","var","Var Title");

is an equivalent of:

out int var;
@var "Var Title";

Dynamic variables can not be used in the applet source code by their names. They can only be accessed via variable management functions such as vvalue(), vtitle() etc.

---
uiupdate() function must be called after using vcreate() to update the user interface changes.

vexists(<string variable name>)

Returns: <boolean>
Test if a variable exists in the applet environment.

Renamed from vpresent() in 2.0 post beta 4.

Allows to test for existance of a variable by its name supplied as a string.

This function can be used to test existance of dynamically created as well as local (in applet) variables.

Example:

```
int q1,q2,q3;

for(int i = 0; vexists("q"+itoa(i)); i++)
{
// this loop will run for all qN variables
}
```

viface(<string name>, <string iface> [, <param 0..5>])

Set the variable interface.

viface() allows to set variable input/output interface dynamically.

optional parameters 3-9 allow setting of interface parameters.

Example:

```
int a;

@a check;
or
iface("a","check");

@a enum [1 4 8];
or
iface("a","enum",[1 4 8]);

float g[];
...
@g graph 0 10 0 5;
or
iface("g","graph",0,10,0,5);
```

---
uiupdate() function must be called to update user interface after using this function.

vset(<string>, <variant>)

Set variable value by specifying variable name as a string.

Allows setting of a variable value by specifying its name as a text string.

Example:

```
int q1,q2,q3 ... q10;

...

int total = 0;
for(int I = 1; I <= 10; I++)
{
vset("q"+itoa(I),I);
}
```

Above example will cycle through variables q1..q10 and set them to the corresponding value of I.

vsettitle(<string>, <string>)

Set variable title.

This function sets the variable title as it should be displayed in the Input or Output panes of the user interface.

This functionality is useful if the the variable meaning changes due to the user input.

vtitle(<string>)

Returns: <string>
Return the title of the variable.

Returns the title of the variable. The title of the variable is displayed in the Input / Output areas of the user interface infront of the variable interface.

Variable titles are defined as follows:

```
@variable_name [ parameters ] "";
```

vvalue(<string>)

Returns: <variant>
Return current value of a variable identified by the string name.

Allows to obtain current value of a variable by specifying the variable name as a string.

This function is helpful when the variable name must be dynamically assembled from different components.

Example:

If an applet has variables q1..q10 it is possible to sum values of all variables using the following loop:

```
int q1,q2,q3 ... q10;

...

int total = 0;
for(int I = 1; I <= 10; I++)
{
total += vvalue("q"+itoa(I));
}
```

For each iteration of the loop the above example will convert variable I to string and append it to the letter "q" thus creating "q1", "q2" etc. Values are then retreived from these variables and added to the total.