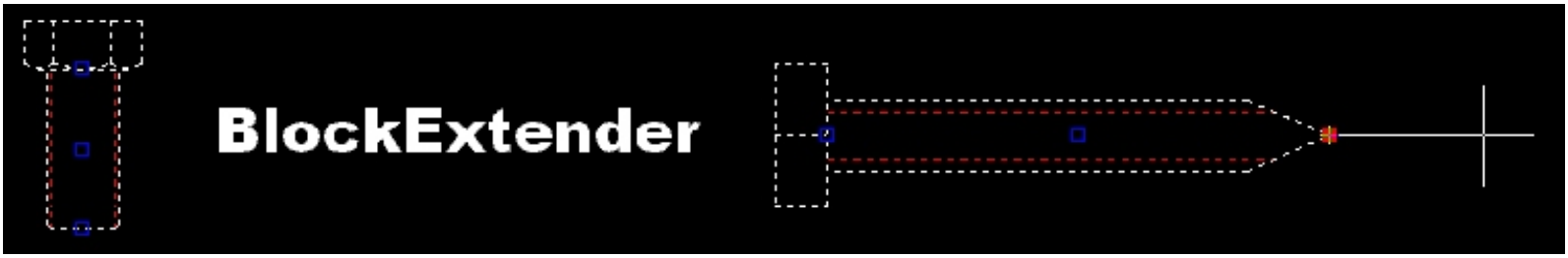
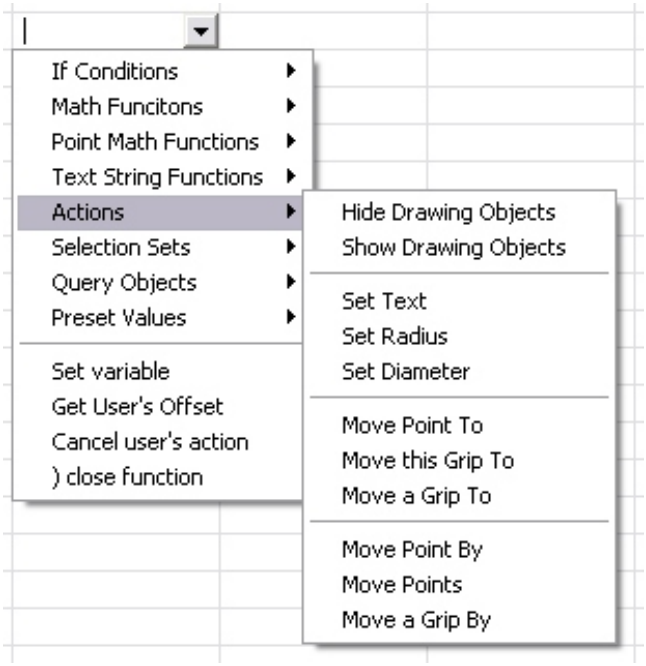


Functions List

BlockExtender provides the ability to add intelligence to any grip point in any block drawing using the command **iedit**. When editing a formula cell a drop down arrow is shown that allows you to easily get an organized list of all the functions available to you (as shown below). This manual is designed to allow you to easily lookup the exact descriptions and learn how each function is used. The list is present in groups exactly as shown in the drop down function menu. The three standalone functions are presented first because of their importance and frequency of use.



Each function will be titled by the name displayed in the drop down menu, followed by a description. Below that the syntax for writing the function and its input arguments is show below with descriptions of the input argument in square brackets [...].

Set variable

The setVar(...) function is used to set the value of a variable. The first input argument will be the name of the variable to be set. The second input argument will be the value to be placed/copied into the variable.

Syntax:

```
setVar(     [variable]     [value]     )
```

Example:

```
setVar(     minDistance    100    )
```

Get User's Offset

The getOffset() function is used to retrieve the distance that the user has tried to stretch this point by. There are no input arguments for this function

Syntax:

```
getOffset()
```

Example:

```
setVar(     offset     getOffset()    )
```

Cancel user's actions

The cancel(...) function is used if you decide you need to abort the user's stretch and return the grip's position to the previous position(s). Please note that this function will undo all grips stretched by one command. So for example if the user uses the stretch command and stretch more than one grip/stretch point and this function is called, all grips will retain their original positions.

The only input argument is a description you may wish to display to the user. This is optional.

Syntax:

```
setVar(     [description]     )
```

If Conditions - Menu Pull Out Items

if

The if function is used when you need to check a value or values in order to decide what course of action to take.

The input argument is a conditional testing function like (= ...), (<= ...), (> ...), etc. If the test is found to be true then the functions found inside the brackets { and } will be activated.

Syntax:

```
if      ([condition testing functions]          )
{



}
```

else do

The else do function is used in conjunction with the "if" statement above. When you test a value or values in order to decide what course of action to take and you want to take a course of action if the test in the "if" statement fails, you will use the else do function.

There are no input arguments. The else do follows the last } of the if statement. If the test is found to be false then the functions found inside the else do brackets { and } will be activated.

Syntax:

```
}
else do
{



}
```

=

The (= ...) function is used in conjunction with the if function. It is used to test to values to see if they are **the same**.

All input arguments are to pass in a value to be tested. A minimum of two arguments are required.

Syntax:

```
(=      [value 1]       [value 2]       [etc.]          )
```

Example:

```
if      (=      dist    100)
```

/=

The (/= ...) function is used in conjunction with the if function. It is used to test to values to see if they are **not the same**.

All input arguments are to pass in a value to be tested. A minimum of two arguments are required.

Syntax:

```
(/=     [value 1]     [value 2]     [etc.]          )
```

Example:

```
if     (/=     dist     100)
```

<

The (< ...) function is used in conjunction with the if function. It is used to test to two values to see first is **less than** the second.

Both input arguments are to pass in a value to be tested. Two arguments are required.

Syntax:

```
(<     [value 1]     [value 2]     )
```

Example:

```
if     (<     dist     100)
```

>

The (> ...) function is used in conjunction with the if function. It is used to test to two values to see first is **greater than** the second.

Both input arguments are to pass in a value to be tested. Two arguments are required.

Syntax:

```
(>     [value 1]     [value 2]     )
```

Example:

```
if     (>     dist     100)
```

<=

The (<= ...) function is used in conjunction with the if function. It is used to test to two values to see first is **less than or equal to** the second.

Both input arguments are to pass in a value to be tested. Two arguments are required.

Syntax:

```
(<=     [value 1]       [value 2]       )
```

Example:

```
if      (<=     dist    100)
```

>=

The (<= ...) function is used in conjunction with the if function. It is used to test to two values to see first is **greater than or equal to** the second.

Both input arguments are to pass in a value to be tested. Two arguments are required.

Syntax:

```
(>=     [value 1]       [value 2]       )
```

Example:

```
if      (>=     dist    100)
```

and

The (and ...) function is used in conjunction with the if function. It is used to test to two or more values or functions to see they are all true.

All input arguments are to pass in a value to be tested. A minimum of two arguments is required.

Syntax:

```
(and    [value 1]       [value 2]       [etc.]  )
```

Example:

```
if      (and    (<=     dist    100)    (>      dist    0)      )
```

or

The (or ...) function is used in conjunction with the if function. It is used to test to two or more values or functions to see they are not the same.

All input arguments are to pass in a value to be tested. A minimum of two arguments is required.

Syntax:

```
(or  [value 1]  [value 2]  [etc.]  )
```

Example:

```
if  (or  (>=  dist  100)  (<  dist  0)  )
```

Math Functions - Menu Pull Out Items

+

The (+ ...) function is used to add two or more values together.

All input arguments are to pass in a value to be calculated. A minimum of two arguments are required.

Syntax:

(+ [value 1] [value 2] [etc.])

Example:

setVar(length (+ dist 100) **)**

-

The (- ...) function is used to subtract two or more values from the first input argument.

All input arguments are to pass in a value to be calculated. A minimum of two arguments are required.

Syntax:

(- [value 1] [value 2] [etc.])

Example:

setVar(length (- dist 100) **)**

*

The (* ...) function is used to multiply two or more values together.

All input arguments are to pass in a value to be calculated. A minimum of two arguments are required.

Syntax:

(* [value 1] [value 2] [etc.])

Example:

setVar(length (* dist 100) **)**

/

The (/ ...) function is used to divide two or more values.

All input arguments are to pass in a value to be calculated. A minimum of two arguments are required.

Syntax:

(/ [value 1] [value 2] [etc.])

Example:

setVar(length (/ dist 100) **)**

exp

The exp(...) function returns the constant e raised to a specified power (the natural antilog).

The only input argument is required.

Syntax:

exp([value])

Examples:

exp(1.0)
Returns: 2.71828

exp(2.2)
Returns: 9.02501

exp(-0.4)
Returns: 0.67032

expt

The exp(...) function returns a number raised to a specified power.

The first input argument is the number to be modified, the second is the modifier (power).

Syntax:

expt([value] [power])

Example:

exp(10.0 2)

expt

The expt(...) function returns a number raised to a specified power.

The first input argument is the number to be modified, the second is the modifier (power).

Syntax:

expt([value] [power])

Example:

expt(10.0 2)

gcd

The gcd(...) function returns the greatest common denominator of two values.

The first input argument is the number to be modified, the second is the modifier (power).

Syntax:

gcd([value1] [value2])

Example:

gcd(12 20)

Returns: 4

make negative

The neg(...) function returns a converted value as a negative. If the value is already negative, it will remain negative.

The only input argument is the number to be modified.

Syntax:

neg([value])

Example:

neg(20)

Returns: -20

make positive

The pos(...) function returns a converted value as a positive value. If the value is already positive, it will remain positive.

The only input argument is the number to be modified.

Syntax:

pos([value])

Example:

neg(-20)

Returns: 20

square root

The sqrt(...) function returns the square root of the input argument value.

The only input argument is the number to be modified.

Syntax:

sqrt([value])

Example:

sqrt(20)

squared

The sqrd(...) function returns the squared value of the input argument value.

The only input argument is the number to be modified.

Syntax:

sqrd([value])

Example:

sqrd(20)

remainder

The rem(...) function divides the first input argument by the second, and returns the remainder

Both input arguments are to pass in a value.

Syntax:

rem([value1] [value2])

Example:

neg(42 12)

Returns: 6

round down

The rndDown(...) function will round the input argument's value down to the nearest whole number.

The only input argument is to pass in a value.

Syntax:

rndDown([value])

Example:

rndDown(2.8)
Returns: 2.0

rndDown(-2.8)
Returns: -3.0

round up

The rndUp(...) function will round the input argument's value up to the nearest whole number.

The only input argument is to pass in a value.

Syntax:

rndUp([value])

Example:

rndUp(2.8)
Returns: 3.0

rndUp(-2.8)
Returns: -2.0

cos

The cos(...) function will return the cosine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

cos([value])

sin

The sin(...) function will return the sine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

sin([value])

tan

The tan(...) function will return the tangent of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

tan([value])

cosh

The cosh(...) function will return the hyperbolic cosine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

cosh([value])

sinh

The sinh(...) function will return the hyperbolic sine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

sinh([value])

tanh

The tanh(...) function will return the hyperbolic tangent of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

tanh([value])

acos

The acos(...) function will return the arccosine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

acos([value])

asin

The asin(...) function will return the arcsine of an angle expressed in radians.

The only input argument is to pass in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

asin([value])

atan

The atan(...) function will return the tangent of an angle expressed in radians.

Up to two input arguments may be passed in as an angle value in radians.
(Use degToRad to convert degree angle values into radian angle values)

Syntax:

atan([value1])

Or:

atan([value1] [value2])

log

The log(...) function will return the natural log of a input argument number.

Only one input argument may be passed in.

Syntax:

log([value])

log10

The log10(...) function returns the natural logarithm for the given float expression. Natural logarithms are calculated by using the base-2 system. However, the log10 function returns the base-10 logarithm.

Only one input argument may be passed in.

Syntax:

log10([value])

logand

The logand(...) function returns the result of the logical bitwise AND of a list of integers.

Two or three input arguments may be passed in.

Syntax:

logand([value1] [value2])

Or:

logand([value1] [value2] [value3])

logior

The logior(...) function returns the result of the logical bitwise inclusive OR of a list of integers

Two or three input arguments may be passed in.

Syntax:

logior([value1] [value2])

Or:

logior([value1] [value2] [value3])

lsh

The lsh(...) function returns the logical bitwise shift of an integer by a specified number of bits.

The first input argument passed in is to be a whole number. If it's not, it will be converted for you using rndDown.

The second input arguement passed in is the number of bits to shift the first input argument. If this value is positive, then the first input argument is shifted to the left; if numbits is negative, then the first input argument is shifted to the right. In either case, zero bits are shifted in, and the bits shifted out are discarded. If the second input argument is not specified, no shift occurs.

Syntax:

lsh([value1] [value2])

radian to degrees

The radToDeg(...) function converts radian angles into angles in degrees.

Only one input argument is required.

Syntax:

radToDeg([value])

degrees to radian

The degToRad(...) function converts angles in degrees into radian angles.

Only one input argument is required.

Syntax:

degToRad([value])

match calculated increment

The incCalc(...) function calculates incremental distances and returns the closest increment to the distance passed in.

Four input arguments are required. The first is the start or minimum of the incremental range, the second is the increment itself, the third is the end or maximum of the incremental range. The fourth is the value that is to be compared against the incremental distances.

Syntax:

incCalc([start] [increment] [end] [test value])

Example:

incCalc(5 5 50 29.5)

Returns: 30

match defined increment

The incDef(...) function finds the nearest incremental distances to the defined incremental values.

Any number of input arguments are allowed. The first is the value that is to be compared to the defined incremental distances. The following values all represent different possible incremental values/distances. Any number of defined increments are allowed up to about 95.

Syntax:

incDef([test value] [inc value1] [inc value2] [etc.])

Example:

incDef(29.5 5.0 17.5 25.0 32.5 50.0 75.0 100.0)

Returns: 32.5

Point Math Functions - Menu Pull Out Items

distance between

The dist(...) function returns the distance between two sets of points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

dist([point1] [point2])

Example:

dist(queryPt(100,100,0) queryPt(200,100,0))

Returns: 100

distance between X's

The distX(...) function returns the distance between two X coordinates of two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

distX([point1] [point2])

Example:

distX(queryPt(100,100,0) queryPt(200,200,0))

Returns: 100

distance between Y's

The distY(...) function returns the distance between two Y coordinates of two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

distY([point1] [point2])

Example:

distY(queryPt(100,100,0) queryPt(100,200,0))

Returns: 100

distance between Z's

The distZ(...) function returns the distance between two Z coordinates of two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

distZ([point1] [point2])

Example:

distZ(queryPt(0,0,0) queryPt(100,200,300))

Returns: 300

midpoint between

The mid(...) function returns the exact midpoint between two sets of points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

mid([point1] [point2])

Example:

dist(queryPt(100,100,0) queryPt(200,100,0))

Returns: 150,100,0

midpoint between X's

The midX(...) function returns the X coordinate of the midpoint between two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

midX([point1] [point2])

Example:

midX(queryPt(100,100,0) queryPt(200,200,0))

Returns: 150

midpoint between Y's

The midY(...) function returns the Y coordinate of the midpoint between two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

midY([point1] [point2])

Example:

midY(queryPt(0,100,0) queryPt(0,200,0))

Returns: 150

midpoint between Z's

The midZ(...) function returns the Z coordinate of the midpoint between two points. (Use the queryPt(function to define which two points to test.

Both input arguments passed in must be points.

Syntax:

midZ([point1] [point2])

Example:

midZ(queryPt(0,0,0) queryPt(0,0,300))

Returns: 150

get X

The getX(...) function is used to extract the X coordinate of a point.

The only input argument is a point.

Syntax:

getX([point])

Example:

getX(queryPt(10,20,30))

Returns: 10

get Y

The getY(...) function is used to extract the Y coordinate of a point.

The only input argument is a point.

Syntax:

getY([point])

Example:

getY(queryPt(10,20,30))

Returns: 20

get Z

The getZ(...) function is used to extract the Z coordinate of a point.

The only input argument is a point.

Syntax:

getZ([point])

Example:

getZ(queryPt(10,20,30))

Returns: 30

set X

The setX(...) function is used to set the X coordinate of a point.

The first input argument is a point. The second is the value to set the X coordinate.

Syntax:

```
setX(       [point]       [value]       )
```

Example:

```
setX(       queryPt(10,20,30)    25)
```

Returns: 25,20,30

Or in this real world example: setVar(is first used to set a variable to the value the user stretched the grip point retrieved by the function getOffset(). Then setX(is used to set the X direction to 0 so the point will only be moved in the Y direction.

```
setVar(     offset        getOffset()    )
setX(       offset        0       )
movePtBy(   10,10,0       offset         )
```

set Y

The setY(...) function is used to set the Y coordinate of a point.

The first input argument is a point. The second is the value to set the Y coordinate.

Syntax:

```
setY(       [point]       [value]       )
```

Examples:

```
setY(       queryPt(10,20,30)    25)
```

Returns: 10,25,30

Or in this real world example: setVar(is first used to set a variable to the value the user stretched the grip point retrieved by the function getOffset(). Then setY(is used to set the Y direction to 0 so the point will only be moved in the X direction.

```
setVar(     offset        getOffset()    )
setY(       offset        0       )
movePtBy(   10,10,0       offset         )
```

set Z

The setZ(...) function is used to set the Z coordinate of a point.

The first input argument is a point. The second is the value to set the Z coordinate.

Syntax:

setZ([point] [value])

Examples:

setZ(queryPt(10,20,30) 25)

Returns: 10,20,25

Or in this real world example: setVar(is first used to set a variable to the value the user stretched the grip point retrieved by the function getOffset(). Then setZ(is used to set the Z direction to 0 so the point cannot be moved in the Z direction.

setVar(offset getOffset())
setZ(offset 0)
movePtBy(10,10,0 offset)

set point

The setPt(...) function is used to set the X, Y and Z coordinates of a new point.

The first input argument is the X value. The second input argument is the Y value. The third and option input argument is the Z value. If the third is not passed in, 0 is assumed.

Syntax:

setPt([X value] [Y value])

Or:

setPt([X value] [Y value] [Z value])

Example:

setPt(10 25)

Returns: 10,25,0

polar from (@ __ < __)

The polar(...) function is equivalent to the command line syntax from drawing a line from a point @ dist < degrees.

The first input argument is a point that will be calculated from. The second input argument is the distance that you wish to calculate the offset by and the third input argument is the angle in radians. (use degToRad to convert an angle in degrees into radians)

Syntax:

polar([point] [distance] [angle in radians])

Command Line Example:

command: line Specify first point: 0,0

command: Specify next point or [Undo]: @100<90

Equivalent Example:

polar(queryPt(0,0,0) 100 degToRad(90))

Returns: 0,100,0

from, to: by X, Y & Z

The fromTo(...) function is equivalent to the command line syntax from drawing a line from a point @ X,Y,Z

The first input argument is a point that will be calculated from. The second input argument is the either a point or optionally three numbers can be passed in as the X offset, the Y offset and the Z offset.

Syntax:

fromTo([point] [point as vector])

Or:

fromTo([point] [X offset] [Y offset] [Z offset])

Command Line Example:

command: line Specify first point: 0,0

command: Specify next point or [Undo]: @100,0,0

Equivalent Example:

polar(queryPt(0,0,0) 100 0 0)

Returns: 100,0,0

angle between two points

The getAngle(...) function returns the angle in radians between two points. (Use radToDeg to convert the angle returned in radians to degrees)

Both input arguments passed in must be points.

Syntax:

getAngle([point1] [point2])

Example:

radToDeg(getAngle(queryPt(0,0,0) queryPt(0,300,0)))

Returns: 45

Actions - Menu Pull Out Items

Hide Drawing Object

The hide(...) function will hide all designated drawing entities. When this function is selected it will prompt you to select some drawing objects on the screen. Then it will insert the show(and one or more drawing object id's that look similar to this "id:1074240856".

Both input arguments passed in must be points.

Syntax:

hide([entities])

Example:

hide(id:1074240856)

Or

hide(id:1074240856 id:1074240864)

Show Drawing Object

The show(...) function will show all previously hidden designated drawing entities. When this function is selected it will prompt you to select some drawing objects on the screen. Then it will insert the show(and one or more drawing object id's that look similar to this "id:1074240856".

Both input arguments passed in must be points.

Syntax:

show([entities])

Example:

show(id:1074240856)

Or

show(id:1074240856 id:1074240864)

Set Text

The setText(...) function will change the text of a designated text, mtext, dimension or attribute drawing object. When this function is selected it will prompt you to select a drawing object on the screen. Then it will insert the setText(function and one drawing object id that looks similar to this “id:1074240856".

Both input arguments passed in must be points.

Syntax:

setText([entity] [text])

Example:

setText(id:1074240856 “new text”)

Set Radius

The setRadius(...) function will change the radius of a designated arc or circle drawing object. When this function is selected it will prompt you to select an arc or circle drawing object on the screen. Then it will insert the setRadius(function and one drawing object id that looks similar to this “id:1074240856".

Both input arguments passed in must be points.

Syntax:

setRadius([entity] [radius])

Example:

setRadius(id:1074240856 40)

Set Diameter

The setDia(...) function will change the diameter of a designated arc or circle drawing object. When this function is selected it will prompt you to select an arc or circle drawing object on the screen. Then it will insert the setRadius(function and one drawing object id that looks similar to this “id:1074240856".

Both input arguments passed in must be points.

Syntax:

setDia([entity] [radius])

Example:

setDia(id:1074240856 20)

Move Point To

The movePtTo(...) function will move any objects that have a point found in the exact location specified.

Both input arguments passed in must be points. The first being the current location of the point (this point can ***only*** be picked at drawing time) and the second is the new location point.

Syntax:

movePtTo([current position] [new position])

Example:

movePtTo(10,10,0 20,10,0)

Move this Grip To

The moveThisGripTo(...) function will move the grip point that this formula is being added to and move it to an location. **Warning:** the moveThisGripTo(function differs from the movePtTo(function because the grip is responsible for moving drawing objects and if the movePtTo(function was used to move the grip, the drawing objects at that point would be ***moved twice***.

Up to two input arguments are required. The optional first is the index identification of the grip point. This is required for grip points that like the line and rectangle style grip points. The possible index values are 1, 2 and 3. Do not pass in an index value if the grip style is not a line or rectangle. The second is the new location of the grip point. **Note:** The possible function moveThisGripBy(will intentionally not be provided.

Syntax:

moveThisGripTo([grip index id] [new location point])

Or:

moveThisGripTo([new location point])

Examples:

moveThisGripTo(1 50,50,0)

Or:

moveThisGripTo(50,50,0)

Move Grip Point To

The moveGripTo(...) function will move any grip point to an exact location. **Warning:** the moveGripTo(function differs from the movePtTo(function because the grip is responsible for moving drawing objects and if the movePtTo(function was used to move the grip, the drawing objects at that point would be ***moved twice***.

Three input arguments are required. The first is the id of the grip. When the function is selected using the drop down menu, you will be prompted to select the grip point on screen. The grip's id will look similar to this "id:1074240856". The optional second is the index identification of the grip point. This is required for grip points that like the line and rectangle style grip points. The possible index values are 1, 2 and 3. Do not pass in an index value if the grip style is not a line or rectangle. The third is the new location of the grip point.

Syntax:

moveGripTo([grip point id] [grip index id] [new location point])

Or:

moveGripTo([grip point id] [new location point])

Examples:

moveGripTo(id:1074240856 1 50,50,0)

Or:

moveGripTo(id:1074240856 50,50,0)

Move Point By

The movePtBy(...) function will move any objects that have a point found in the exact location specified by a specific distance. The movePtTo(function differs because it moves the point to a specific location.

The first input argument is the current location of the point (this point can ***only*** be picked at drawing time) and the second is a vector that specifies the offset distance and direction that the point is to be moved. **Note:** the function getOffset() is most often used to get the offset vector for the second argument.

Syntax:

movePtBy([current position] [vector offset])

Examples: (to move by a specific offset)

movePtBy(10,10,0 10,0,0)

Or: (to move by a the offset the user indicated during the stretch)

movePtBy(10,10,0 getOffset())

Or: (to move by a the offset the user indicated during the stretch, but setting the Y offset to 0 so the point will only be moved in the X direction)

setVar(offset getOffset())

setY(offset 0)

movePtBy(10,10,0 offset)

Important Note: If a point is moved by this function and also by a grip point, than this point will be moved twice and probably too far. It is better to use movePtTo(...) if this point will also be moved by a grip point.

Move Grip Point By

The moveGripBy(...) function will move any grip that has a point found in the exact location specified by a specific distance. The moveGripTo(function differs because it moves the point to a specific location. **Warning:** the moveGripBy(function differs from the movePtBy(function because the grip is responsible for moving drawing objects and if the movePtBy(function was used to move the grip, the drawing objects at that point would be ***moved twice***.

Three input arguments are required. The first is the id of the grip. When the function is selected using the drop down menu, you will be prompted to select the grip point on screen. The grip's id will look similar to this "id:1074240856". The optional second is the index identification of the grip point. This is required for grip points that like the line and rectangle style grip points. The possible index values are 1, 2 and 3. Do not pass in an index value if the grip style is not a line or rectangle. The third is a vector that specifies the offset distance and direction that the grip point is to be moved.

Syntax:

moveGripBy([grip point id] [grip index id] [vector offset])

Or:

moveGripBy([grip point id] [vector offset])

Examples:

moveGripBy(id:1074240856 1 50,00,0)

Or:

moveGripBy(id:1074240856 50,00,0)

Move Points

The movePts(...) function will move any drawing entities with points that cross into the selection fence (crossing window or polygon). This function is used in conjunction with the getFence(...) To setup and retrieve the selection fence. **Warning:** If the movePts(function is used to move any grips, the drawing objects controlled by those grips may be ***moved twice***.

Two input arguments are required. The first is the fence selection argument. When the function is selected using the drop down menu, you will be prompted to select the select fence on screen. The select fence will then be inserted and will look something like this: getFence(0,0,0 10,0,0 10,10,0 0,10,0). The optional second input argument is a vector that specifies the offset distance and direction that the grip point is to be moved.

Syntax:

movePts([select fence] [vector offset])

Examples:

movePts(getFence(0,0,0 10,0,0 10,10,0 0,10,0) getOffset())

Or:

setVar(fence getFence(0,0,0 10,0,0 10,10,0 0,10,0))
movePts(fence getOffset())

Selection Sets - Menu Items

Selection Fence

The getFence(...) function is used to setup and retrieve a selection fence (crossing window or polygon) in conjunction with the function movePts(...) which will move any drawing entities with points that cross into that selection fence.

Several input arguments are required. Each is a point of the selection fence.

Syntax:

```
getFence(   [point 1]   [point 2]   [point 3]   [point 4]   [etc.]   )
```

Examples:

```
getFence(   0,0,0   10,0,0  10,10,0   0,10,0   )
```

Or:

```
setVar(   fence   getFence(   0,0,0   10,0,0  10,10,0   0,10,0   )   )
movePts(   fence   getOffset()   )
```

Query Objects - Menu Items

Query Point

The queryPt(...) function is used to find the current location of the point selected on screen. When the drop down menu is used to insert this function the dialog box will hidden and you will be prompted to select the point on screen. Because these points can change at any time in intelligent blocks the select point may be anywhere in the drawing.

One input arguments as a point is required.

Syntax:

queryPt([point])

Example:

setVar(placement queryPt(10,10,0))

Query Text

The queryText(...) function is used to extract the text value of the indicated text, mtext, dimension or attribute drawing object. When the drop down menu is used to insert this function the dialog box will hidden and you will be prompted to select the text drawing object on screen. Then it will insert the drawing object id's that look similar to this "id:1074240856".

Note: If the indicated object is a dimension and the dimension is displaying its measurement and not that of text that of overriding text, the numeric measurement value will be returned.

One input arguments as a drawing object id is required.

Syntax:

queryText([drawing object id])

Example:

setVar(text queryText(id:1074240856))

Query Radius

The queryRadius(...) function is used to extract the radius value of the indicated arc, circle or ellipse drawing object. When the drop down menu is used to insert this function the dialog box will hidden and you will be prompted to select the drawing object on screen. Then it will insert the drawing object id's that look similar to this "id:1074240856". **Note:** If an ellipse is found the radius ratio will be returned.

One input arguments as a drawing object id is required.

Syntax:

queryRadius([drawing object id])

Example:

setVar(radius queryRadius(id:1074240856))

Query Diameter

The queryDia(...) function is used to extract the diameter value of the indicated arc or circle drawing object. When the drop down menu is used to insert this function the dialog box will hidden and you will be prompted to select the drawing object on screen. Then it will insert the drawing object id's that look similar to this "id:1074240856".

One input arguments as a drawing object id is required.

Syntax:

queryDia([drawing object id])

Example:

setVar(diameter queryDia(id:1074240856))

Query Dimension Measurement

The queryDim(...) function is used to extract the measurement value of the indicated dimensional drawing object. When the drop down menu is used to insert this function the dialog box will hidden and you will be prompted to select the drawing object on screen. Then it will insert the drawing object id's that look similar to this "id:1074240856".

One input arguments as a drawing object id is required.

Syntax:

queryText([drawing object id])

Example:

setVar(length queryDim(id:1074240856))

Query this Grip's Point

The queryThisPt() function is used to find the current location of the grip point this formula is being applied to.

No input arguments are required.

Syntax:

queryThisPt()

Example:

setVar(placement queryThisPt())

Preset Values - Menu Items

true

The “true” value is used to test a variable or result if it is true or to set a variable as true.

Syntax:

```
true
```

Example:

```
setVar(  test  true  )
```

false

The “false” value is used to test a variable or result if it is false or to set a variable as false.

Syntax:

```
false
```

Example:

```
setVar(  test  false  )
```

pi

The “pi” value is a constant π. It evaluates to approximately 3.14159.

Syntax:

```
pi
```

Example:

```
setVar(  value  (*  pi  180  )  )
```

90 degrees

The “90deg” value is a constant. It is the value for the angle in radians of the angle of 90 degrees.

Syntax:

90deg

Example:

```
setVar(  angle  (+  angle  90deg  )  )
```

180 degrees

The “180deg” value is a constant. It is the value for the angle in radians of the angle of 180 degrees.

Syntax:

180deg

Example:

```
setVar(  angle  (+  angle  180deg  )  )
```

270 degrees

The “270deg” value is a constant. It is the value for the angle in radians of the angle of 270 degrees.

Syntax:

270deg

Example:

```
setVar(  angle  (+  angle  270deg  )  )
```

Text String Functions - Menu Items

Left

The left(...) function is used to extract a certain length of text from the left side of large text string.

The first input argument is the text string to search. The second is a number indicating how many letters to extract.

Syntax:

left([search text] [number of letters])

Example:

left(“some text” 4)

Returns: “some”

Middle

The middle(...) function is used to extract a certain length of text from the middle side of large text string.

The first input argument is the text string to search. The second is a number indicating where to start extracting the text. The third is an optional number indicating how many letters to extract. If the third is missing, the function will extract all the text from the indicated starting point to the end of the string.

Syntax:

middle([search text] [starting index] [number of letters])

Or:

middle([search text] [starting index])

Examples:

middle(“some text here” 5 4)

Returns: “text”

Or:

middle(“some text here” 5)

Returns: “text here”

Right

The right(...) function is used to extract a certain length of text from the right side of large text string.

The first input argument is the text string to search. The second is a number indicating how many letters to extract.

Syntax:

right([search text] [number of letters])

Example:

right(“some text” 4)

Returns: “text”

Find

The find(...) function is used to find where a section of text starts that may be found in a larger text string. If not found -1 is returned.

The first input argument is the text string to search. The second is the text string used to search the first. The third option input argument is a number indicating where to start searching

Syntax:

find([main text] [search text] [starting index])

Or:

find([main text] [search text])

Examples:

find(“some text here” “text”)

Returns: 5

Or:

find(“some text here” “not”)

Returns: -1

Or:

find(“some text here” “e” 5)

Returns: 7

Find One Of

The findOneOf(...) function is used to search a text string for the first letter found in the search text string. If not found -1 is returned.

The first input argument is the text string to search. The second is the text string used to search the first.

Syntax:

findOneOf([main text] [search text])

Examples:

findOneOf(“some text here” “abcde”)

Returns: 4

Insert

The insert(...) function is used to insert a text string into another text string.

The first input argument is the main text string. The second is the index indicating where to insert the text. The third is the text string that is to be inserted.

Syntax:

insert([main text] [insert position] [replacement text])

Examples:

insert(“some text here” 5 “new “)

Returns: “some new text here”

Make Lower

The makeLower(...) function is used return the input string as all lowercase letters.

The first input argument is the text string.

Syntax:

makeLower([text])

Examples:

makeLower(“SOME TEXT HERE”)

Returns: “some new here”

Make Upper

The makeUpper(...) function is used return the input string as all upper-case letters.

The first input argument is the text string.

Syntax:

makeUpper([text])

Examples:

makeUpper(“some new here”)

Returns: “SOME TEXT HERE”

Length

The length(...) function is used return the length of the text.

The first input argument is the text string.

Syntax:

length([text])

Put Strings Together

The putTogether(...) function is used make one large text string out of two or more input strings.

All input arguments are as text strings.

Syntax:

putTogether([text 1] [text 2] [text 3] [etc.])

Examples:

putTogether(“some “ ”new “ ”here”)

Returns: “some new here”

Replace

The replace(...) function returns a new string based on the first input string but with text replaced.

The first input arguments is the main text strings. The second is the old text and the third is the replacement text.

Syntax:

replace([original text] [old text] [new text])

Examples:

replace(“everyone likes hockey “ ”hockey “ ”football”)

Returns: “everyone likes football”

Reverse Find

The revFind(...) function is used to find where a single letter starts that may be found in a larger text string. This function differs from the find(...) function in that it searches only for one letter and also starts the search from the end of the text not the beginning. If not found -1 is returned.

The first input argument is the text string to search. The second is the text string used to search the first. The third option input argument is a number indicating where to start searching

Syntax:

find([main text] [search letter])

Or:

find([main text] [search letter])

Examples:

find(“some text here” “t”)

Returns: 5

Or:

find(“some text here” “n”)

Returns: -1

Or:

find(“some text here” “e”)

Returns: 14

Trim Left

The trimLeft(...) function will return the main input text with a specified letter removed from the left side.

The first input arguments is the main text strings. The second is the letter to be removed.

Syntax:

trimLeft([original text] [letter])

Examples:

trimLeft(“ddadd“ ”d “)

Returns: “add”

And:

trimLeft(“dadd“ ”d “)

Returns: “add”

Trim Right

The trimRight(...) function will return the main input text with a specified letter removed from the rightside.

The first input arguments is the main text strings. The second is the letter to be removed.

Syntax:

trimRight([original text] [letter])

Examples:

trimRight(“ddadd“ ”d “)

Returns: “dda”

And:

setVar(text trimRight(“100.0000“ ”0“))

setVar(text trimRight(text ”.“))

Returns: “100”

Angle To String

The angToStr(...) function will convert an angular value in radians into a text string.

The first input arguments is the angle number.

The optional second is argument is a number that specifies the angular units style. If the unit is omitted, the current value of the AutoCAD system variable AUNITS. The following units may be specified:

-1 Use the drawings current settings.

0 Degrees

1 Degrees/minutes/seconds

2 Grads

3 Radians

4 Surveyor's units

The optional third argument is a number indicating the number of decimal placements for the precision of the number. A value of 0 - 8 is permitted.

Syntax:

angToStr([angle number] [units] [precision])

Examples:

angToStr(1.57079632 0 4)
Returns: “90.0000"

Number To String

The numToStr(...) function will convert numeric value into a text string.

The first input arguments is the number.

The optional second is argument is a number that specifies the linear units style. If the unit is omitted, the current value of the AutoCAD system variable LUNITS. The following units may be specified:

-1 Use the drawings current settings.
1 Scientific

2 Decimal

3 Engineering (feet and decimal inches)

4 Architectural

5 Fractional

The optional third argument is a number indicating the number of decimal placements for the precision of the number. A value of 0 - 8 is permitted.

Syntax:

numToStr([number] [units] [precision])

Examples:

numToStr(12 4 4)
Returns: “1'-0""

String To Angle

The strToAng(...) function will convert a text string to an angular value in radians. **Note**: if the text cannot be converted a value of 0 will be returned.

The first input arguments is the text string.

The optional second is argument is a number that specifies the angular units style which the text is formatted in.. If the unit is omitted, the current value of the AutoCAD system variable AUNITS. The following units may be specified:

- **-1** Use the drawings current settings.
- **0** Degrees
- **1** Degrees/minutes/seconds
- **2** Grads
- **3** Radians
- **4** Surveyor's units

Syntax:

strToAng([text] [units])

Examples:

strToAng(“90.0000" 0)
Returns: 1.57079632

String To Number

The strToNum(...) function will convert a text string into a numeric value.

The first input arguments is the text string.

The optional second is argument is a number that specifies the linear units style. If the unit is omitted, the current value of the AutoCAD system variable LUNITS. The following units may be specified:

-1	Use the drawings current settings.
1	Scientific
2	Decimal
3	Engineering (feet and decimal inches)
4	Architectural
5	Fractional

Syntax:

numToStr([text] [units])

Examples:

strToNum(“1'-0"" 4)
Returns: 12

3rd Day Software
P.O. Box #808
Grande Cache, Alberta, Canada
T0E 0Y0
www.objectdcl.com/BlockExtender.html
sales@objectdcl.com