

## CONTENTS

### GETTING STARTED

[Sources of help](#)

[Scope of app](#)

[Guide to controls](#)

---

[New in this version](#)

[Contact](#)

### REFERENCE

[Constructions](#)

[Expression syntax](#)

[Expression reference](#)

---

[Tracing errors](#)

[FAQ](#)

Tap on the Utilities button at the bottom-right to return to the Utilities screen.

### EXAMPLES AND TUTORIALS

For the purposes of learning the program there are example files and “replay tutorials” available. Example files are simply pre-made models. Replay tutorials are annotated re-enactments of using the app. It is recommended that these be used to explore the use of Algernon before progressing onto creating a new model.

To load and view the examples and tutorials, follow these steps:

1. Tap on the Utilities tab at the bottom of the screen.
2. Tap on the **Demo** button. A list will appear containing the names of examples and tutorials.
3. Scrolling down if necessary, tap on one of the names to load.

### DOCUMENTATION

While working with the app it is not possible to have this documentation visible. If you have Airprint available then these pages may be printed out via the button at the top of the screen.

### BEGIN EXPLORING

Tap on the Geometry tab at the bottom of the screen. A view of the structure should be seen.

Read the [Scope of app](#) page for further information.

## SCOPE OF APP

Algernon is an associatively parametric geometry tool. What this means is that geometry is created in terms of the relationships between points, lines, and circles (“entities”) and those relationships persist when any entity is changed, causing an automatic recalculation of the entire geometry. Aspects of the geometry such as distances and angles, can be expressed in terms of variables and mathematical formulae.

One of its primary purposes is as a geometry calculator and experimenting tool. Complex geometry can be constructed and dimensions taken from the results. If the geometry is constructed in terms of variables then it is easy to change the parameters and immediately observe the effect.

For draughting, Algernon supports inking of lines in various styles and the addition of text.

### BASIC PRINCIPLES OF USE

Geometry is created with **constructional entities**. These are infinite lines, full circles, and points. Entities are typically defined in terms of others, e.g. lines tangent to circles or running through intersection points.

To create a drawing suitable for printing, **ink** is added. Ink can be drawn with various line patterns, pen thicknesses, and colours. Straight lines can be created from point to point and arcs can be drawn around circles.

**Dimensions** can be added. Linear from point to point, radius or diameter of circles, and angles of arcs.

**Text** in a choice of fonts and sizes can be added, anchored to a point and optionally oriented parallel to a line.

### CREATING GEOMETRY

An important aspect of Algernon is the way in which geometry is defined. Unlike most CAD tools which have a proliferation of tools for all of the different constructions, Algernon needs only one tool for lines, one tool for circles, and one tool for points. For example, to draw a line tangential to two circles, you choose the line tool, tap on each circle approximately where you expect the tangents to be, and then tap the **apply** button (✓). Alternatively, to draw a line parallel to another and passing through an intersection of two others, you choose the line tool, tap on the line which defines the parallel, tap on the intersection, and then tap on the **apply** button.

When creating a construction, a preview will be drawn. It will be committed to the model when **apply** (✓) is tapped, or discarded if **cancel** (✕) is tapped.

A list of construction idioms are listed [here](#)

### EDITING (CHANGING) GEOMETRY

Algernon uses a noun-verb scheme for editing. Firstly, the select tool is chosen. Then the entity to be edited is selected by tapping on them. Then the **edit** button (⇨) is tapped. The entity then appears in its constructed form as had created it. The construction can then be changed as if in the process of creating it. The changes are then either accepted or cancelled.

### SELECTING OBSCURED ENTITIES

Sometimes two or more entities may overlap or obscure each other, such as ink overlying a circle, or text placed at a point. In this instance, tapping on them will select one of the entities. Tapping a second time will deselect that and select the next entity. This can be repeated as many times as necessary.

### PARAMETERS AND VARIABLES

Geometrical entities will often have a parameter, e.g. a circle might require a radius to be specified. The parameter can be typed in during the creation or editing of the entity. It can take the form of a simple number, a reference to a variable, or a mathematical expression calculating the value.

Variables are helpful in constructing geometry, particularly when a value is to be used in many places. Any quantity of variables can be defined and given convenient names, and they can be inter-dependent and relate to properties of geometrical entities.

An explanation of expression functions and operators are listed [here](#)

## TABS

Operation is divided over a set of tabs:

-  The geometry screen. All geometry and draughting is performed here.
-  The styles manager. This is for creating, editing, and deleting draughting styles.
-  The variables editor.
-  The dependency navigator. This is useful for tracing errors in constructions or calculations.
-  The Utilities screen. From here models can be loaded, saved, and printed.

## GEOMETRY TAB

### TOOL BAR

A tool must be chosen before creating a geometrical entity of the corresponding type (points, lines, circles, etc.). The select tool allows pre-existing entities of any type to be chosen by tapping on them and then subsequently edited.

The tool buttons are:

-  Select existing entities
-  Create Points
-  Create Lines
-  Create Circles
-  Create Dimensions
-  Create Text
-  Create Ink

### CONTROL PANEL

The control panel buttons are context-sensitive and only appear as necessary.

They act either generally or upon the entities that are selected or being edited.

-  Edit: Begins editing of the selected entity.
-  Clear: If editing an entity then it cancels the changes made, otherwise it clears the selection.
-  Apply: Sets the changes made to the currently edited entity.
-  Deletes the selected entity.
-  Swaps the definitions of two selected entities of the same type.
-  Undo previous edit.

 Redo an undone edit.

 Opens the styles picker.

If in doubt, use  to abandon edits and selections, and return the interface to a passive state.

## KEYBOARD ACCESSORIES

When editing a formula there are multiple keyboards available and some additional accessory buttons.

 Move cursor left.

 Move cursor right.

 Switch to the standard system keyboard.

 Switch to the numeric keyboard.

 Switch to the Variables list.

 Switch to the History list.

## CONSTRUCTIONS

Below are tables describing the sequences of taps for constructing entities in various ways.

<u>Points</u>	<u>Lines</u>	<u>Circles</u>
<u>Dimensions</u>	<u>Ink</u>	<u>Text</u>

### Key:

P	Point	X	Intersection	F	Freespace
L	Line	C	Circle	#	Expression
PLC	Any one of point, line, or circle				
F/#	Freespace and/or expression				

### PROMOTIONS AND EXPRESSIONS

When creating entities, certain taps will be promoted into points which will be automatically produced and the created entities will be dependent upon them. In particular:

- An intersection X can be used in all constructions whenever a point is required.
- A tap in freespace can be used whenever a point is required except when it would conflict with a tap sequence which defines an interpretation for freespace (as listed in these tables).

Taps in freespace can have certain interpretations for some constructions. In these instances, an expression may be entered by the user to provide the information mathematically. In the tables this is indicated by F/#. For example:

- Creating a point in free space can be set to the coordinates (x,y) by typing [  $x, y$  ] into the formula input field.
- Taps in freespace which are interpreted as distances or angles can be replaced by typing the distance or angle directly into the formula input field.

### POINTS

#### Taps Creates point:

F	at arbitrary location in space
F #	at [ $x, y$ ] in space
X	at intersection
P P	midway between points
P F/#	at vector offset [ $\Delta x, \Delta y$ ] from point
L	at arbitrary position on line
C	at arbitrary position on circle

### LINES

#### Taps Creates line:

P F/#	at angle through point
P P	through two points
P L	through point, perpendicular to line

P L F/# through point at angle to line  
 P C through point, tangential to circle  
 L F parallel to line at arbitrary distance  
 L F # parallel to line at specified distance  
 L P parallel to line, through point  
 L L bisect two lines  
 L C parallel to line, tangential to circle  
 C F/# tangential to circle, at angle  
 C P tangential to circle, through point  
 C L tangential to circle, perpendicular to line  
 C C tangential to two circles

## CIRCLES

### Taps

### Creates circle:

P	F/#	centred on point with radius
P	P	centred on first point, edge through second point
P	L	centred on point, tangential to line
P	C	centred on point, tangential to circle
C	F/#	concentric to circle at distance
C	P	concentric to circle through point
C	L	concentric to circle, tangential to line
C	C	concentric to first circle, tangential to second circle
PLC	PLC	F tangential to first two, with arbitrary radius
PLC	PLC	F # tangential to first two, with specified radius
PLC	PLC	PLC tangential to all three

## DIMENSIONS

### Taps

### Creates linear dimension:

P	P	F	between points at arbitrary distance
P	P	P	between first two points, through third point
P	P	L	between points, projected onto line

### Taps

### Creates arc-length dimension:

P	P	C	between points, concentric with circle
L	L	P	between lines, through point <sup>(*)</sup>
L	L	C	between lines, around circle

### Taps

### Creates angular dimension:

P	C	P	between points, concentric with circle
L	P	L	between lines, through point <sup>(*)</sup>

L C L between lines, around circle

*(\*) the measurement is around a circle with its centre at the intersection of the lines and its edge passing through the point.*

All angular and arc-length dimensions are constructed anti-clockwise, with the positions of the tap indicating where the dimension text is displayed (internal/external).

## INK

**Taps**                      **Creates inked line:**

P P straight segment between points

P C P arc around circle between points

L along entire line

C around entire circle

Segments and arcs can be concatenated in a chain from point to point.

## TEXT

**Taps**    **Creates text:**

P T anchored to point, printed horizontal

P P T anchored to first point, scaled to second point, printed horizontal

L P T anchored to point, printed parallel to line

L P P T anchored to first point, scaled to second point, printed parallel to line

The text is typed into the formula input field.

When scaling text, the two points describe the size of an imaginary rectangle parallel to the angle of the text. The text is made as large as possible to fit within this size. The text is printed relative to the first point and does not necessarily lie within the imaginary rectangle. All text in the same style is set to the same size; being the smallest for which all scaled texts will remain within their prescribed sizes.

## EXPRESSION SYNTAX

All of the geometry in Algernon is represented by mathematical expressions. When constructing geometry graphically, the system defines entities (points, lines, circles, etc.) in mathematical terms of their parameters and their relationships with other entities. This creates a parametrically associative geometry.

The user can define the parameters of entities in terms of numbers, variables, and functions by giving an expression to the relevant construction. Typically this may be a radius of a circle or the distance between parallel lines.

Variables form an important basis of the system. The user can create a variable and provide an expression to define it as either a simple number, a function of other variables, or a function of geometrical entity.

### SYNTAX

#### NUMBERS

Numbers are in scientific form, with a dot (.) used to separate decimals. An exponent can optionally follow the number using the letter e to separate it.

Examples: 500 500.0 +500.0 5e2

The last example means  $5 \times 10^2 \Rightarrow 500$ .

#### VARIABLES

Variable names must begin with a letter (A~Z, or foreign characters) and can contain letters, numbers (0~9), and the underscore (\_). They must not include punctuation, symbols, or brackets. Foreign-language letters and numerals may be used so long as they do not conflict with the syntax of the expression.

Examples: x dy radius angle\_2

Names are not case-sensitive. This means that DY is identical to dy.

#### OPERATORS

Operators are primarily symbolic, such as + for addition. A full list appears in the [reference tables](#).

Standard precedence rules apply, and round brackets can be used when required.

Examples:  $2+3*4 \Rightarrow 14$ ,  $(2+3)*4 \Rightarrow 20$

#### ARRAYS

An array is represented by a list of sub-expressions separated by commas and contained within square brackets.

Example:  $[1, 2*3, (4+5)/3] \Rightarrow$  array of 1, 6, 3

Geometrical coordinates and vectors are formed as 2-element arrays:  $[x,y]$  or  $[\Delta x, \Delta y]$

Most operators can work with arrays as well as scalars.

Example:  $[5, 6] + [1, -2] \Rightarrow [6, 4]$

When mixing arrays and scalars, the array must come first.

Example:  $[5, 6] * 2 \Rightarrow [10, 12]$

Example:  $[3, 4, 5]^2 \Rightarrow [9, 16, 25]$

For convenience, square brackets are not required around an *entire expression* that yields an array. So the following are equivalent:

Example:  $5, 6 \Rightarrow [5, 6]$

This convenience will be mostly experienced when specifying coordinates for points.

## ARRAY INDEXING

Elements from an array can be picked out according to their position in the array using the [...] operator. The first element in the array is at index 0.

Example: Let variable  $V = [10, 11, 12, 13, 14, 15]$ , then  $V[3] \Rightarrow 13$ .

Subarrays can be picked too by specifying an array for the indices.

Example:  $V[[1, 3, 5]] \Rightarrow [11, 13, 15]$ .

## ENTITIES

Geometrical entities created graphically are automatically named by the system. They can be referenced within an expression by prefixing their name with the @ symbol.

Example:  $@C1 \Rightarrow$  the circle C1.

An entity can be interrogated and characteristics extracted for calculation by following its name with a dot and a parameter. A list of the parameters for each kind of entity is given in the [reference tables](#).

Example:  $@P1.position \Rightarrow$  the coordinate of point P1.

In the particular case of points, the point can be used directly as a coordinate in calculations.

Example:  $hypot(@P2-@P1) \Rightarrow$  the distance between points P1 and P2.

## FUNCTIONS

Many calculations will involve functions. A full list appears in the [reference tables](#). Each function takes one or more arguments and returns one or more values. The arguments are a list separated by commas and contained within round brackets after the function name.

Example:  $\sin(45) \Rightarrow$  the sine of 45 degrees.

Example:  $\log(10, 65) \Rightarrow \log(65)$  to base 10.

All functions which take or return an angle will work in terms of degrees. To convert from radians to degrees use the  $\deg(x)$  function, and to convert from degrees to radians use the  $\text{rad}(x)$  function.

## MATHEMATICAL CONSTANTS

A small number of mathematical constants are known to the system. They are prefixed with the # symbol and are listed in the [reference tables](#).

When text is added to the model, expressions can be embedded in the text using the format `% (...expr...)`. The expression is evaluated, and numbers are presented in scientific format.

A variable may incorporate a text string by enclosing it in quotes: `"..."`.

## ADVANCED SYNTAX

### GATING

Particularly when using `time`, it can be a good idea to “gate” a variable which changes frequently so as to make it easy to disable.

For example, instead of:

```
pos = cos(time/10), sin(time/5)
```

it is better to say:

```
gated = time*1
```

```
x = cos(gated/10), sin(gated/5)
```

because then time can be easily stopped by changing `gated = time*0`.

### NAMED MEMORIES

Within an expression, a named memory can be defined with the clause `{let name subexpr}`. This will evaluate the sub-expression `subexpr` and store the result in a memory called `name`. This value can then be used subsequently within the expression by citing the name.

A memory can be created almost anywhere in an expression, but note the need for a comma afterwards if continuing the expression beyond, and the possible need for brackets if it occurs within an operation.

```
Example: {let r 10}, #pi*r*r
```

```
⇒ #pi*10*10
```

```
⇒ 314.159.
```

A named memory can updated later in the expression.

```
Example: {let r 10}, #pi*r* ({let r r*2}, r)
```

```
⇒ #pi*10*20
```

```
⇒ 628.319.
```

An alternative way of putting a value into a named memory is with `{use name}`. This will remove the most recent value within the evaluation and store it in the memory.

```
Example: 10, {use r}, #pi*r*r
```

```
⇒ 314.159 (not [10, 314.159]).
```

A variable's memories are created anew on each evaluation of the variable, and exist only for the duration of the evaluation.

Memories are only accessible by the expression in which they are defined. Two variables with memories of the same name will each see only their own. If an expression defines a memory with the same name as another variable then citing that name within the expression will give a reference to the memory, not the variable. It is recommended to avoid giving the same names to memories and variables in order to avoid any confusion.

## ITERATION

Named memories are useful when we require iteration. Iteration is performed with the syntax `{while cond subexpr}`.

The sub-expression `subexpr` is evaluated repeatedly while the condition value `cond` is not `#false`.

Example: `{let n 1}, [ {while (n<=5) n, {let n n+1}} ]`

$\Rightarrow [1, 2, 3, 4, 5]$ .

Note that there is a limit of 1000 conditional tests within an evaluation, to guard against accidental infinite loops. For unusually lengthy calculations, consider breaking the expression down across user functions, as described below.

## USER FUNCTIONS

A variable can act as a function if its expression begins with one or more `{use name}` clauses. The variable by itself cannot be computed because its expression will not evaluate a value for the memory.

An expression in another variable (or geometrical construction) can refer to this variable and provide values for the memories using the parameter `.call(...)`.

Example:

`square = {use r}, r*r  $\Rightarrow$  function`

`area = #pi*square.call(10)  $\Rightarrow$  314.159.`

A description of syntax is given [here](#).

## REFERENCE TABLES

[Operators](#)      [Functions](#)      [Parameters](#)  
[Variables](#)      [Constants](#)

### OPERATORS (I)

#### Operator      Yields (Example)

$+ x$	positive X +5
$- x$	negative X -5
$* x$	Expand array [X] $*[1, 2, 3] \Rightarrow 1, 2, 3$
$! x$	Logical NOT X $!#false \Rightarrow true$

$x \wedge y$	X to the power of Y $5^2 \Rightarrow 25$
--------------	---

$x * y$	X times Y $4 * 2 \Rightarrow 8$
---------	------------------------------------

$x / y$	X divided by Y $11 / 4 \Rightarrow 2.75$
---------	---

$x // y$	Ratio of X:Y $11 // 4 \Rightarrow 2.75$
----------	--

$x \% y$	remainder of integer X/Y $11 \% 4 \Rightarrow 3$
----------	---

$x + y$	X plus Y $3 + 5 \Rightarrow 8$
---------	-----------------------------------

$x - y$	X minus Y $5 - 3 \Rightarrow 2$
---------	------------------------------------

$x < y$	X less than Y $3 < 5$
---------	--------------------------

$x <= y$	X less than or equal to Y $5 <= 5$
----------	---------------------------------------

$x \approx y$	X approximately equal to Y $\text{sqrt}(2) \approx 1.41421356$
---------------	---

$x = y$	X exactly equal to Y $1 + 1 = 2$
---------	-------------------------------------

$X > Y$	X greater than Y $5 > 3$
$X \geq Y$	X greater than or equal to Y $5 \geq 5$
$X \approx Y$	X not approximately equal to Y $\text{sqrt}(2) \approx 1.4$
$X \neq Y$	X not exactly equal to Y $1 + 0.999 \neq 2$

$X \& Y$	Logical X AND Y $(A > 0) \& (B > 0)$
----------	---

$X   Y$	Logical X OR Y $(A > 0)   (B > 0)$
---------	---------------------------------------

$X    Y$	Logical X Exclusive-OR Y $(A > 0)    (B > 0)$
----------	--

- The  $\approx$  and  $\neq$  operators compare to about 1 part in 10 million, and are used to tolerate rounding errors in calculations.
- Testing for exact equality with  $=$  and  $\neq$  should not be used in situations where rounding errors can occur.
- The ratio operator  $X // Y$  is the same as division, except a divide-by-zero will yield  $\pm$ infinity instead of an error.
- The remainder operator  $X \% Y$  is equivalent to  $X - Y * \text{trunc}(X/Y)$
- For logical operators and comparisons, anything that is not false is true. Truth values should not be used in arithmetic.
- For all logical tests, anything that is not exactly equal to the constant `#false` will be considered true.

## OPERATORS (II)

A further class of operator is that of the ternary conditional operator:

Operator	Yields (Example)
$X ?? Y ? : Z$	Y if X is true, else Z $(A > 0) ?? \text{sqrt}(A) ? : 0$

Only the sub-expression (Y or Z) required for the result is evaluated and the other not evaluated. This makes it suitable for avoiding invalid arithmetic.

If this operator is to be used within a wider expression then the whole clause must be contained within brackets ( $(X ?? Y ? : Z)$ ). This is because it is composed of three terms but represents only a single term.

## FUNCTIONS

Name	Yields
$\text{abs}(X)$	absolute value of X
$\text{acs}(X)$	arc-cosine of X
$\text{ang}(X, Y)$	angle (degrees) of vector [X, Y]
$\text{asn}(X)$	arc-sine of X
$\text{atn}(X)$	arc-tangent of X
$\text{cos}(X)$	cosine of X

<code>count(X)</code>	the number of elements in array X
<code>deg(X)</code>	convert X radians into degrees
<code>hypot(X,Y)</code>	hypotenuse: $\sqrt{X^2+Y^2}$
<code>ln(X,Y)</code>	Natural logarithm of X
<code>log(X,Y)</code>	Logarithm of Y to base X
<code>max(X,Y)</code>	Highest of X or Y
<code>min(X,Y)</code>	Lowest of X or Y
<code>rad(X)</code>	convert X degrees into radians
<code>reverse(X,Y,...)</code>	[...Y,X]
<code>round(X)</code>	X rounded to nearest whole number
<code>sin(X)</code>	sine of X
<code>sqrt(X)</code>	square-root of X
<code>sum(X,Y,...)</code>	X+Y+...
<code>tan(X)</code>	(trigonometric) tangent of X
<code>trunc(X)</code>	X truncated to whole number

- The functions `ang`, `hypot`, `reverse`, `strcat`, and `sum` can take either multiple scalars or a single array.

## ENTITY PARAMETERS

Name	Yields
<code>@P<sub>n</sub>.position</code>	coordinate of point P <sub>n</sub> as an array [x,y]
<code>@L<sub>n</sub>.intercept</code>	y-intercept b at x=0, such that $y = m.x + b$
<code>@L<sub>n</sub>.slope</code>	slope m, such that $y = m.x + b$
<code>@L<sub>n</sub>.through</code>	a point on line L <sub>n</sub> as an array [x,y]
<code>@L<sub>n</sub>.vector</code>	the normalised direction vector of line L <sub>n</sub> as an array [ $\Delta x, \Delta y$ ]
<code>@C<sub>n</sub>.area</code>	area of circle C <sub>n</sub> as a +ve scalar
<code>@C<sub>n</sub>.centre</code>	centre of circle C <sub>n</sub> as a coordinate [x,y]
<code>@C<sub>n</sub>.center</code>	
<code>@C<sub>n</sub>.circumference</code>	circumference of circle C <sub>n</sub> as a +ve scalar
<code>@C<sub>n</sub>.radius</code>	radius of circle C <sub>n</sub> as a +ve scalar
<code>@D<sub>n</sub>.angle</code>	angle of arc dimension D <sub>n</sub>
<code>@D<sub>n</sub>.length</code>	length of dimension D <sub>n</sub>
<code>@I<sub>n</sub>.area</code>	enclosed area of ink I <sub>n</sub> (see important note below)
<code>@I<sub>n</sub>.centroid</code>	centroid of enclosed area of ink I <sub>n</sub> (see important note below)
<code>@I<sub>n</sub>.length</code>	length of ink I <sub>n</sub>

- The values given by `@In.area` and `centroid` are only correct for ink either around circles or forming a simple closed loop in which the ink must start and end at the same point and no edges cross over each other. If any edges do cross then the value will differ by plus or minus the intersected areas.
- If any parameter is attached to an array of entities then it is applied to each entity in the array, yielding an array of results.

Example: `sum([@D1,@D2,@D3].length)`  $\Rightarrow$  summation of dimension lengths.

## BUILT-IN VARIABLES

Name	Meaning
------	---------

<code>@time</code>	A 10Hz clock measuring seconds.
--------------------	---------------------------------

- It is recommended that the variable `time` be gated, as described in the [syntax documentation](#).

## CONSTANTS

Name	Value	Meaning
<code>#e</code>	2.71828...	Base of natural logarithm
<code>#false</code>	0	Logical FALSE
<code>#golden</code>	1.61803...	Golden ratio
<code>#pi</code>	3.14159...	$\pi$ (pi)
<code>#tau</code>	6.28318...	$2 \times \pi$
<code>#true</code>	<code>!#false</code>	Logical TRUE

- The constants `#true` and `#false` should rarely be needed directly because the `?` operator tests implicitly for truth. Do not use them in calculations. Comparisons against `#true` should not be made explicitly. For an explicit truth test, compare against `#false` instead.

## ENTITY NAVIGATOR

While Algernon does not allow invalid geometrical entities to be created, a pre-existing one may become invalid if its parameters are unsuitably changed.

For example, a circle with radius  $\text{sqrt}(r)$  will become invalid if variable  $r$  is changed to a negative number. In this situation, either  $r$  needs to be edited to remain positive, or the circle's radius expression be changed to  $\text{sqrt}(\text{abs}(r))$ .

When a geometrical entity is invalid, it cannot be drawn, and so it cannot be selected in the geometry view for editing. This is where the navigator becomes useful. It allows expressions to be traced from the top level (entities and variables upon which no others are dependent) down through the hierarchy of dependencies to find the entity or variable at the root of any problem. In the example above, the circle is a top-level entity which is dependent upon the variable  $r$ .

The Navigator screen (tab ) displays the dependencies for a single item, an entity or variable which here we will call  $V$ . The screen is divided in half: the left side shows the items which are dependent upon  $V$ , and the right-hand side shows those upon which  $V$  is dependent.

Items on either side can be tapped to move through the hierarchy. If a label is red then it means that the item is in error. Tracing red items to the right allows us to find where an error begins.

The aim is to fix the root problem first and then work back up to the left until all are fixed.

When viewing a variable, tapping on **Select** will go to the variable manager tab with the variable highlighted.

When viewing an entity, tapping on **Select** will go to the geometry tab with the entity selected (any pre-existing selection is cleared). If the entity is in error then the selection might not be visible, but the opportunity exists for the entity to be edited or swapped with another.

Tapping on **Top** at the top-left of the navigator tab will display the top-level of the hierarchy. At the top level there are no items on the left side of the screen. The right-hand column shows the items which have no dependencies, plus at the top of the list are all those at the root of errors. This gives quick access to those causing problems and should be the first ones to investigate.

## FREQUENTLY ASKED QUESTIONS

### WHY CAN I NOT LOAD A MODEL?

To guard against accidentally losing unsaved data, another model can only be loaded if the currently loaded model has been either saved or discarded.

### CAN I USE INK FOR CONSTRUCTION?

Ink cannot be used directly within geometrical constructions because it is intended to be the final drawing on top of the constructed geometry.

However, inked regions do have parameters (e.g. length, area, centroid) which can be used in formulae and parameters for constructing geometry.

Similarly, the lengths and angles of dimensions can be used in subsequent calculations even though the dimensional entities themselves cannot form part of a geometrical construction.

## GENERAL

- Filing now uses the standard iOS document browser.
- iOS 12.1 now minimum requirement.

## EARLIER CHANGES

### GENERAL

- Updates for iOS 11 and iPhone X.

### GENERAL

- Fixed potential crash when exporting files.
- iOS 9.1 now minimum requirement.
- Miscellaneous improvements.

### USER INTERFACE

- When editing an expression, the cancel and apply buttons at either side of the field have been replaced by an Undo button and an apply button; the latter now indicating to close the keyboard. This is to remove ambiguity with the tick/cross buttons elsewhere.

### TEXT

- Text can now be constructed to fit within a model rectangle by specifying two points. This overrides the given font size in the text's style.

### GENERAL

- Updates for iOS 9.
- Miscellaneous improvements.

### SYNTAX CHANGES

The following changes in syntax might affect pre-existing models which use the old syntax:

- Operator `?...:...` is changed to `??...?:...` to avoid ambiguity.
- Improved consistency of mixed array/scalar handling in `ang()`, `hypot()`, and `sum()`.

If a pre-existing model is affected then expressions will require manual editing to change to the new syntax.

### STYLES

- Added filling of enclosed inked areas.

### TEXT

- Text can now incorporate expressions using `% (...)`.  
e.g. `cos(45) is % (cos(45))` → “cos(45) is 0.707107”

## EXPRESSIONS

- Added `max(x,y)` and `min(x,y)` functions.
- Added built-in `time` variable; a running clock suitable for animating models.
- Added ratio operator `X // Y`.
- Added array indexing, to pick elements from an array.
- Added `count()` function, returning the size of an array.
- Added `reverse()` function, reversing the content of an array.
- Added circle parameters `.area` and `.circumference`.
- Added line parameters `.slope` and `.intercept`.
- Added ink parameters `.area` and `.centroid`.
- Added memories within variable expressions.
- Added iteration within variable expressions.
- Added user functions within variable expressions.
- Expressions can contain (or yield) strings with “...”.
- Added `strcat()` function to concatenate strings.

## GENERAL

- Added navigator to help track down causes of unrepresentable constructions or errors in formulae.
  - Added AirDrop for supported devices.
  - Added DXF export.
  - Colour-coded listing in variables manager according to content of expression.
  - Miscellaneous improvements.
- 

## USER INTERFACE

- Moved variable manager to its own tab.
- Moved styles manager to its own tab, now separated from the picker accessed as before in the Geometry tab.
- Double-tapping with two fingers on the geometry will zoom to the extent of the draughted entities.
- Triple-tapping will zoom to the extent of constructions.
- Summary statistics (length, radius, angle, etc.) are printed at the top of the screen for the selected entity or the constructed entity.

## KEYBOARDS

- Enhanced numeric keyboard, also adding a shift button.
- Added function list to keyboard.
- Added undo button to keyboard.
- Long formulae wrap around onto multiple lines when being edited.

- Accepting/cancelling an edited formula now uses dedicated buttons.
- Fixed bug whereby keyboard would be incorrect size.

## **MATHEMATICAL**

- Added natural-log function.

## **DISPLAY**

- Dimension text is better positioned.
- If (only if) dimension units match the model units then the unit symbol is not printed.

## **GENERAL**

- Expanded on-line help.
- Miscellaneous improvements.

## CONTACT

Website: [www.intesym.co.uk/mobile.php](http://www.intesym.co.uk/mobile.php)

E-mail: [casa@intesym.co.uk](mailto:casa@intesym.co.uk)