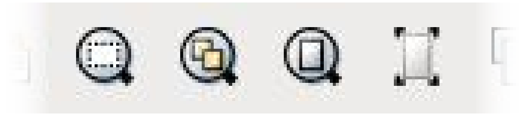
Written by Mark Crutch

Inkscape - Part 129

Aside from the snap controls that I described last month, at first glance it may not look like much has changed in the main Inkscape window with the release of version 1.2 – apart from the welcome return of these zoom buttons to the main Control bar, of course:



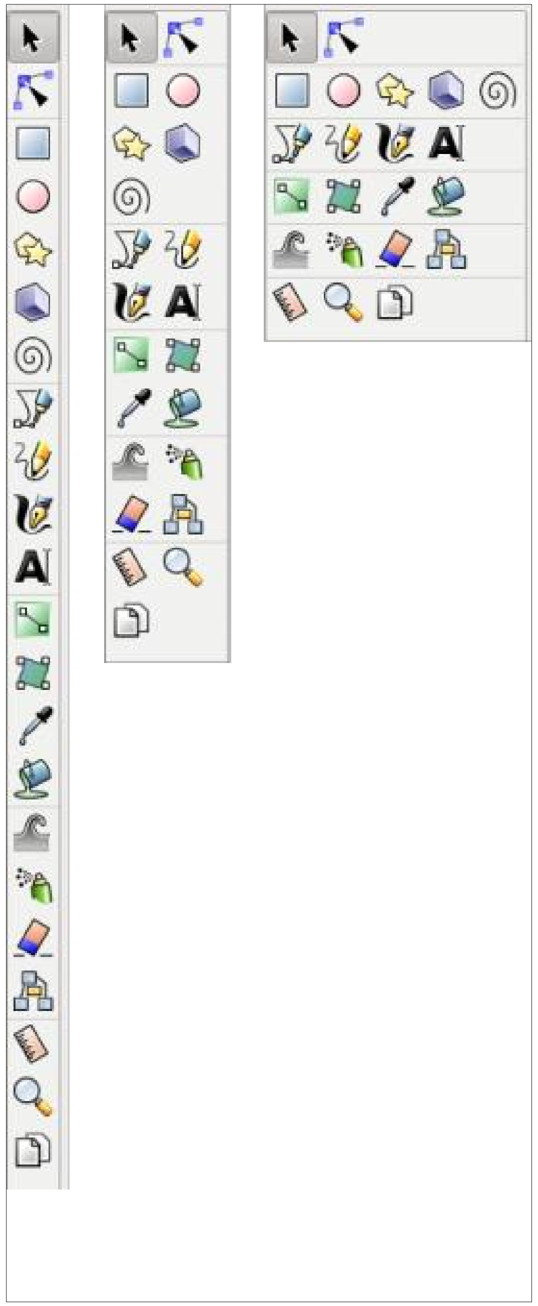
Observant readers may notice that not only have the 'Zoom to fit selection', 'Zoom to fit drawing', and 'Zoom to fit page' buttons reappeared, but they're accompanied by a new sibling. Clicking this fourth button will center the page in the window, but without changing the current zoom. This can be useful when you've lost your way in a drawing, either because you've zoomed in or out too far, or have gone a bit wild with the pan option.

A less obvious addition to the main window is a change to the toolbox. Last month, I looked at the new preferences that allow you to turn individual tool icons on and off, or to change their size. But those are not the only changes to the way these buttons are displayed. It's now possible to adjust the width of the toolbox by dragging the right-hand separator. There's no obvious visual affordance for this (i.e. a 'handle'), but if you move the mouse slowly over the dividing line between the toolbox and the ruler (or canvas, if you have rulers hidden), your mouse pointer will change to indicate where you can click-and-drag. This allows you to transition seamlessly from the classic vertical list of icons, to a squarer layout in which each group of buttons occupies its own row – or, indeed, something in-between if you prefer.

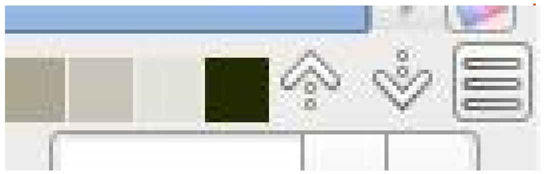
Personally I think that the one-row-per-group view is rather wasteful, as it's not possible to dock other dialogs in the empty space below the icons. But I can certainly see the benefit of changing to two columns if you're working on a screen with limited height, but still want access to all of the icons rather than turning some off.

One thing to note is that it's possible to drag the divider the other way, to collapse your toolbox down until it's no longer visible. That's not the same as using the View > Show/Hide > Toolbox menu entry – that menu will still indicate that the toolbox is visible, even though you can't see it. To avoid confusion, I suggest not collapsing the toolbox by dragging the separator: if you want to hide it, use the menu entry instead. If that's something you do regularly, then you can assign a keyboard shortcut to the menu option via Edit > Preferences > Interface > Keyboard. It's in the 'Canvas Display' section of the keyboard shortcuts.

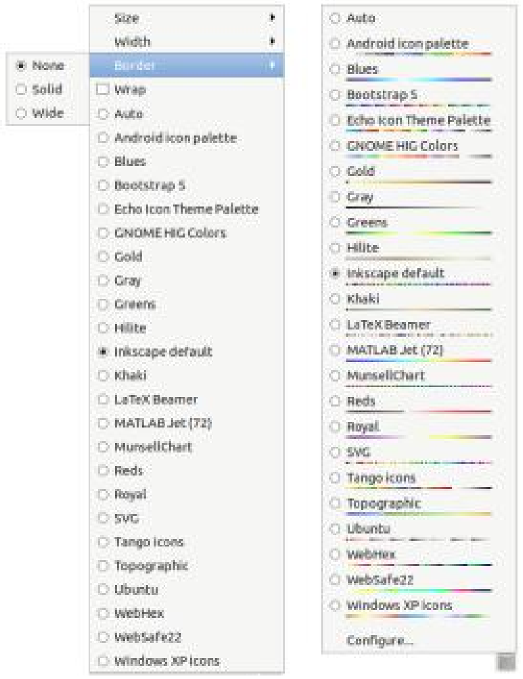
Once again, observant readers may notice the addition of a new icon to the end of the toolbox. This is for managing multiple pages in an Inkscape document, which is such a major new feature that I'll be covering it separately in future.



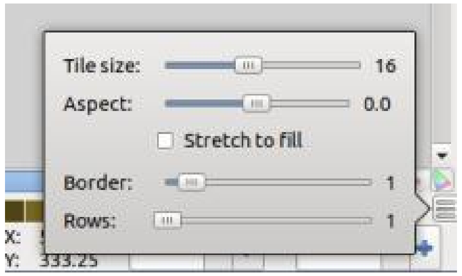
There's been one other significant change to the main window but, like the resizeable toolbox, it's not immediately obvious that anything is different. The color palette, at the bottom of the window, has seen a major overhaul. Most of the changes take place towards the right of the palette, where you'll find a pair of up/down buttons, and a menu button.



We'll skip the up/down buttons for now, and go straight to the menu button. Users familiar with earlier releases may know that there's been a pop-up menu hanging around in this corner for a long time, allowing you to switch between different palettes and adjust a few settings related to the display of the swatches. The old design was a text-only menu, with some submenus providing a limited selection of options (e.g. None/Solid/Wide for the border around each color swatch). The new menu is a richer UI widget, allowing for a small thin-line preview of each palette – which makes it much easier to pick the right one. This image shows the old and new menus, for comparison.



The various options for configuring the display of the palette have now been moved into a single pop-up which is opened via the 'Configure…' option at the bottom of the menu.



Tile size: 16
Aspect: 0.0
☐ Stretch to fill
Border: 1
Rows: 1

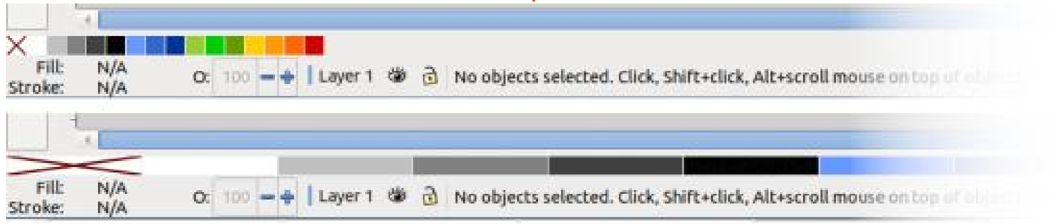
X:
Y: 333.25

The options in here are pretty self-explanatory. 'Tile size' sets the basic size for the swatches, though the actual height and width will vary based on the aspect ratio. I'll gloss over the fact that the 'Aspect' slider runs from -1.0 to 1.0 (that's not really how aspect ratios work), and just state that negative values make the swatches taller than they are wide, positive values make them short and fat, and zero makes them square. The nice thing about all the controls in this pop-up is that you can see their effect on the swatches dynamically as you change them, so it's not worth getting too caught up in the specific values – just drag the sliders until the palette looks the way you want it to.

The 'Stretch to fill' option is a little odd. Ticking it disables the Aspect slider entirely, which in 'normal' UI terms should really mean that the checkbox is put above the slider to indicate the parent-child relationship between them. That's not what makes it odd though: the weird thing is that it really does what it suggests it will only for palettes with few entries. Let's start by looking at a case where it does work: the very limited color set of the 'Android icon palette'. Here's how that palette appears with the 'Stretch to fill' option disabled (top) and enabled (bottom).

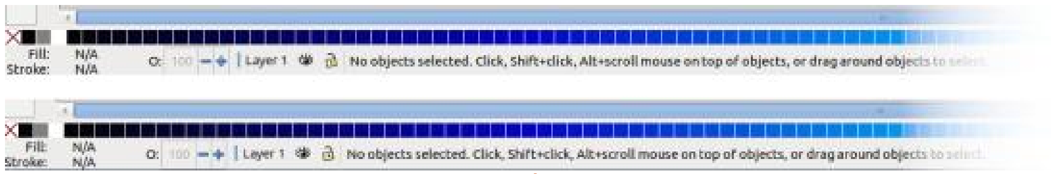
It doesn't take a genius to see that the 'Stretch to fill' option has stretched the individual swatches to fill the available space. No big surprises there – it's exactly what you would expect an option with that name to do. But what happens if there are a larger number of colors in the palette – something like the 'Blues' palette, for example. Here's how that palette appears with the 'Stretch to fill' option disabled (top) and enabled (bottom).

If you're struggling to spot the



Fill: N/A
Stroke: N/A
O: 100
Layer 1
No objects selected. Click, Shift+click, Alt+scroll mouse on top of

Fill: N/A
Stroke: N/A
O: 100
Layer 1
No objects selected. Click, Shift+click, Alt+scroll mouse on top of



Fill: N/A
Stroke: N/A
O: 100 | Layer 1 No objects selected. Click, Shift+click, Alt+scroll mouse on top of objects, or drag around objects to select.

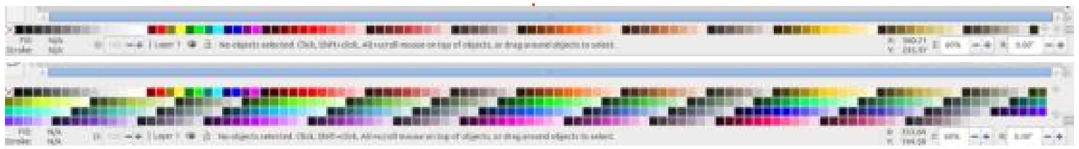
Fill: N/A
Stroke: N/A
O: 100 | Layer 1 No objects selected. Click, Shift+click, Alt+scroll mouse on top of objects, or drag around objects to select.

difference, you're not alone. In both cases, the swatches actually extend over three rows (on my screen), with those up/down buttons being used to switch between them (you can also use the scroll wheel on your mouse, if you prefer). But clearly the swatches haven't been made narrower in order to fit them all in the available space, as you might expect or want. 'Stretch to fill' very specifically means 'Stretch' and not 'Compress'.

Perhaps you're thinking that Inkscape is just being sensible, as there are too many colors in that palette to fit on one row in a usable way. Not so: unchecking that option and manually setting the tile size to 11 and the aspect to -0.7 is enough to get the entire palette to fit in a single line.

Moving on from the inconsistencies of stretching swatches, the 'Border' option adjusts the amount of space around each swatch. If you want your palette to look like a continuous gradient of color (assuming the tones are so arranged), then set this to zero. Higher values add more space around each swatch, which may be useful to avoid mis-clicks, or simply to ensure each swatch appears as an individual item rather than blending with its neighbours.

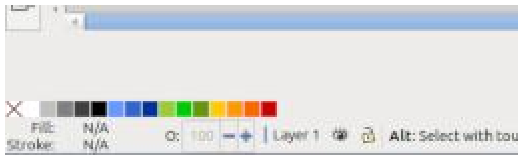
The final option, 'Rows', exposes what I think is the most significant flaw in the new palette interface. When the number of swatches is simply too large for them all to appear in the available space, they overflow onto multiple rows. You can then scroll through them using the up/down buttons or the mouse wheel, as I described above. The 'Rows' control determines how many rows can be displayed in the palette area – basically, how tall the palette area is. This can allow you to display an entire palette at once, rather than having to use the up/down buttons to access them all. This image shows the difference between Rows=1 and Rows=5 when using the Inkscape Default palette.



No objects selected. Click, Shift+click, Alt+scroll mouse on top of objects, or drag around objects to select.

No objects selected. Click, Shift+click, Alt+scroll mouse on top of objects, or drag around objects to select.

If you're the sort of person who wants to see all the colors in the palette, all the time, this might be fine. But what if you're also the sort of person who switches between different palettes, depending on your needs? Here's how the palette area looks when you've got Rows=5 but you've selected the Android Icon Palette.



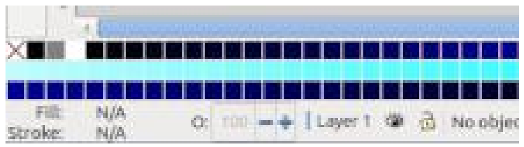
Fill: N/A
Stroke: N/A
O: 100 | Layer 1 Alt: Select with tou

That's a lot of wasted space. Enabling the 'Stretch to fill' option doesn't help much either, as that only stretches the swatches horizontally, and does nothing to fill the additional blank rows. The 'Rows' parameter sets a fixed number of rows to display, even if fewer rows would make more sense. It would be nicer, I think, to have the number of displayed rows adjust dynamically, using the 'Rows' parameter to set a maximum.

But in my opinion there's a bigger issue to consider. In previous versions, the palette would scroll horizontally if it couldn't all fit on screen. You could switch to a vertical scrolling mode by enabling the 'Wrap' checkbox in the palette menu, which would give the same effect as the palette in version 1.2. But the new release doesn't offer an equivalent way to revert to the horizontal scrolling design of its predecessors. You're forced to use a vertically scrolling palette, whether you like it or not.

This may seem like a trivial gripe, but there are pragmatic reasons why a horizontally scrolling palette is arguably better in some cases. Let's consider that sweeping blue palette once more – here it is with Rows=3 on my machine.

Try to imagine that as one long



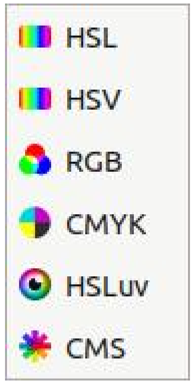
Fill: N/A
Stroke: N/A
O: 100 | Layer 1 No objec

horizontal palette. Yes, you can only see a small 'window' of colors at a time, but scrolling the mouse wheel over the colors smoothly moves the line along, with no breaks or discontinuities. It's easy to see the relative colors of every swatch when compared with its neighbours.

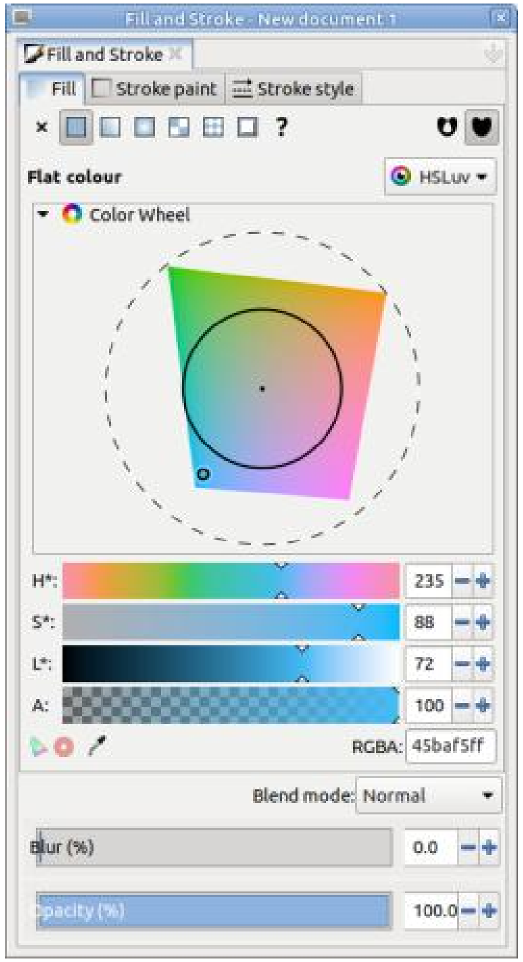
With a multi-row display, however, there are breaks artificially added to the flow of color. Your eyes have to scan from the end of one row to the start of the next in order to continue progressing along the palette. Swatches can now be completely surrounded by up to 8 immediate neighbours, not just one on each side. This affects your eye's ability to discriminate between the colors, especially where (as in this screenshot), some light colors are sandwiched between darker rows. And the position of those breaks aren't consistent, but change with the window size and the parameters chosen for the palette display. Your favourite shade of blue might be at the start of the second line one day, but buried somewhere in the middle of a block of color the next.

For now, us 'single row' advocates just have to live with this forced vertically scrolling design, but I do hope that a horizontally scrollable palette makes a return in a future release.

Moving on from the palette to a related topic: the Fill & Stroke dialog has seen some useful tweaks and improvements. You may recall that previous versions offered tabs, and then buttons, to switch between different color pickers. The choices were RGB, HSL, HSV, CMYK, Wheel, and CMS. Those tabs/buttons have been replaced once more, with a pop-up menu that adds 'HSLuv' to the mix.



You may have noticed that there's no 'Wheel' option on the menu. Fear not! The wheel has been moved to a collapsible section in the HSL, HSV and HSLuv modes, giving you the option to use both the wheel and the sliders in combination far more easily. I don't understand why it didn't also make it into the RGB view, as I can't see any technical reason for preventing it (Inkscape uses RGB internally, so, even if you pick your colors using CMYK, what ends up in the file is actually an RGB conversion of your color). The HSLuv view is an interesting one: I can't really make sense of it, even with the visualisation offered by the wheel view, but I daresay there are users with more specific color requirements (and knowledge) who will benefit from the addition of this mode.



Flat colour HSLuv

H*: 235
S*: 88
L*: 72
A: 100
RGBA: 45baf5ff
Blend mode: Normal
Blur (%) 0.0
Opacity (%) 100.0

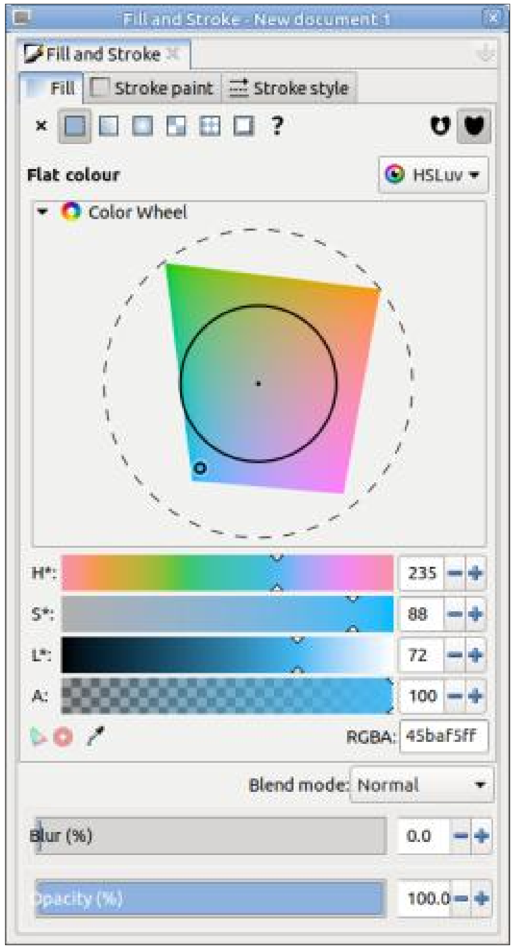
The color pickers aren't the only change in the Fill & Stroke dialog… but I've reached my word count for this article, so the other new additions will be the subject of next month's instalment.

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

Written by Mark Crutch

Inkscape - Part 130

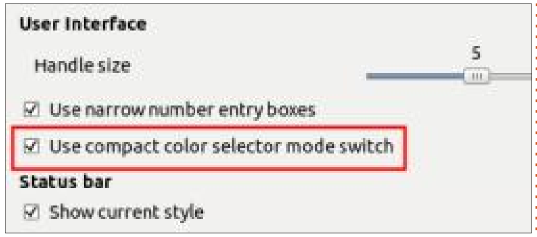
Last time, we looked at the updated color pickers in the Fill & Stroke dialog, including the more widely available color wheel, and the new HSLuv color picker.



H*: 235; S*: 88; L*: 72; A: 100; RGBA: 45baf5ff; Blend mode: Normal; Blur (%): 0.0; Opacity (%): 100.0

To be honest, I still struggle to understand how to pick a specific target color in the HSLuv picker, but it's apparently intended to make it easier to choose colors that have an optically similar saturation. You may find that one practical use for this mode is to perform your 'normal' color selection using the palette or the other pickers, but then switch to this mode if you need to create additional colors that share a similar 'strength'. For example, this might be helpful if you're trying to stick to pastel colors in your design, without having to work around all the more vibrant ones in the other color pickers.

As I mentioned last time, switching between the new color pickers is done using a pop-up menu. This keeps the UI looking neat and tidy, but does mean a bit more mouse work, either in clicks or drags, to switch back and forth between the choices. If you do need to change between different pickers quite frequently, then you might find it more convenient to return to a UI that presents all the options at once. In that case, open the Inkscape Preferences (Edit > Preferences) and select the 'Interface' entry at the left (select it, don't just expand it). Part-way down, you'll find the 'Use compact color selector mode switch' option.



User Interface

Handle size: 5

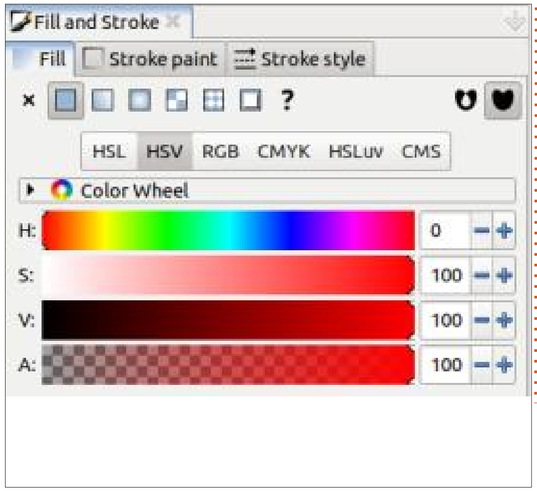
☑ Use narrow number entry boxes

☑ Use compact color selector mode switch

Status bar

☑ Show current style

Un-check that option and the color picker pop-up will be replaced by a group of radio buttons instead. It's not the nested-tabs interface of old, but it's a bit closer to the previous design, and requires fewer mouse interactions.



H: 0; S: 100; V: 100; A: 100

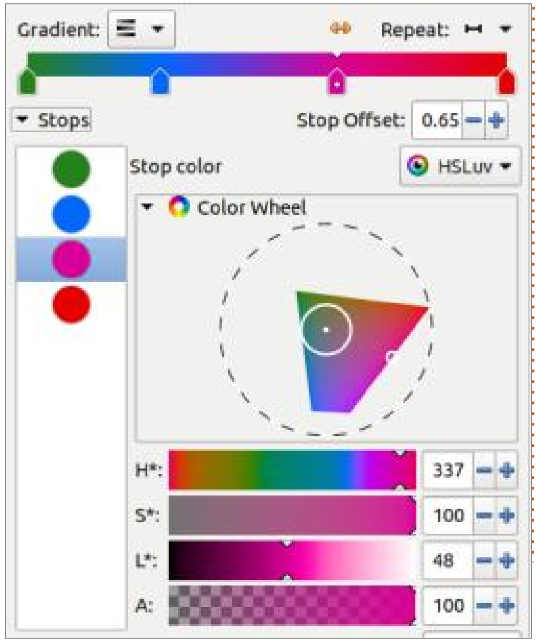
So far, I've shown these color pickers in the 'Flat Color' mode of the Fill tab, but this whole section is a common component that is used elsewhere in the application, whether it's the 'Swatch' section of the Fill tab, or the 'Replace Color' tool in the Extensions > Color menu. And, in all cases, the preference to use radio buttons is honoured, so you won't find yourself dealing with inconsistencies in that part of the UI across the application.

One place that makes use of the color pickers is in the Gradient Editor. Recent Inkscape converts might only be familiar with the on-canvas editing of gradients, but veteran users will remember the days of a simple gradient editor accessed via the Fill & Stroke dialog. For a long time, it was possible to re-enable that with a preference, but there's no longer a need for that as a brand-new gradient editor is now available by default.

The old editor was sparse and

could best be described as 'functional'. But it was still extremely useful at times. When dealing with very small objects or complex scenes, it's often easier to edit a gradient via a separate UI, rather than on the canvas. The re-addition of a dedicated editor, as well as the on-canvas tools, now offers the best of both worlds. And the new editor is a lot more functional than the old one, too.

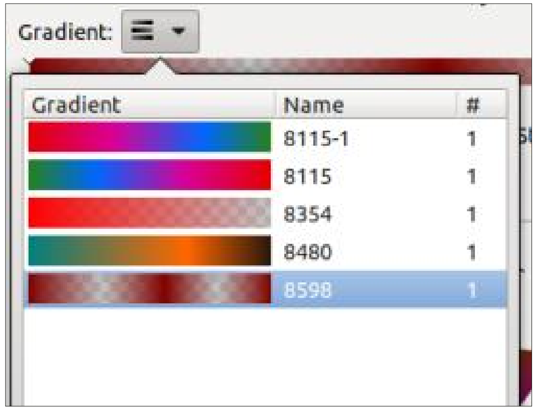
As has long been the case, linear and radial gradients are separate buttons at the top of the Fill or Stroke Paint tabs of the dialog. You can also find Mesh Gradients here, but they have nothing to do with the gradient editor, and still have to



Gradient: ☰ Repeat: ⟼
Stops
Stop Offset: 0.65
Stop color HSLuv
Color Wheel
H*: 337
S*: 100
L*: 48
A: 100

be modified entirely on-canvas. Having selected either Linear or Radial gradient mode, you're presented with a compact UI that squeezes a lot of options into a small space.

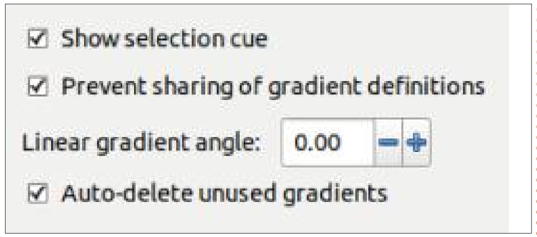
At the top-left is one of the most important parts of this interface, but it's one that is easy to gloss over. This pop-up houses the gradient list that used to occupy this entire tab in previous releases!



Gradient: ☰

Gradient	Name	#
	8115-1	1
	8115	1
	8354	1
	8480	1
	8598	1

You may wonder why I think this is so important: this is the key to avoiding gradient proliferation. Particularly in very complex drawings, it's possible to end up with a large number of gradients, many of which are either identical, or similar enough, that they could be replaced with a single entry. In that case, you may wish to consolidate some of those gradients into one, ensuring that all the objects update when you change the gradient, and potentially resulting in a smaller file size too. Conversely, you might want to ensure that each gradient is used only once, so that edits for one object don't affect any others.



☑ Show selection cue
☑ Prevent sharing of gradient definitions
Linear gradient angle: 0.00
☑ Auto-delete unused gradients

This pop-up shows you all the gradients that currently exist in the document, together with a count of the number of objects they're applied to. With an object selected on the canvas, pick a gradient from this list to have it applied, replacing any existing gradient it might have had (you should also see the count – in the '#' column – increase when you do this). The pop-up is also a good way to see if you still have any gradients in the document that aren't being used (the count column shows 0), and which can therefore be removed by using File > Clean Up Document, before saving. Note that these will also be auto-removed if the 'Auto-delete unused gradients' setting is enabled in Edit > Preferences, within the Tools > Gradient pane, so check that if you find that unused gradients are being purged when you don't want them to be.

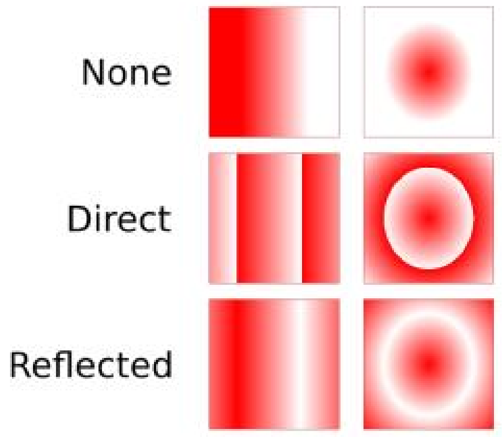
At the bottom of this pop-up are plus and minus buttons. The former will duplicate the currently selected gradient and apply it to the currently selected object(s). This is useful if you need a gradient that is similar to an existing one, but want a separate copy that can be edited without affecting other objects. The minus button deletes the currently selected gradient if it is 'unused'. The use of this word in the tooltip is perhaps a little misleading: it will delete a gradient if it is actively used on the currently selected object, but no others – i.e. if there's a '1' in the count column (so, not really 'unused' then). The button becomes disabled if the column holds '2' or more, preventing really serious disasters that might affect many objects. What it doesn't do is clear out any non-selected gradients, whether they have a '1' or even a '0' in the count column. You either have to select-then-remove each of those separately, or use the File > Clean Up Document option to delete the '0' entries en masse.

One feature which is still

missing from this part of the UI, and which I would love to see added, is an option to select all the objects that use a particular gradient. It's all well and good knowing that I've still got one object left on the page using a gradient, but it would be better still if the application could show me which object it is!

Moving on from this pop-up, on the same row of controls, you'll next find a small button. In my screenshot, it's shown as a double-headed arrow, but it may differ depending on what icon set you use. This button reverses the entire gradient, flipping the order of all the stops. Note that using this button will automatically create a copy of your gradient (assuming the 'Prevent sharing of gradient definitions' preference is enabled in the Tools > Gradient pane of the prefs), so any other objects sharing the original won't be adversely affected. I don't really know why this button doesn't deserve to have a label when all the other controls on this row do, but it means that it doesn't stand out quite as much as it should.

Finally, on this row is another pop-up – complete with label. This one says 'Repeat', and the pop-up lets you choose whether the gradient should not repeat ('None'), should repeat as a direct copy of itself ('Direct'), or should alternate its repeats between direct and reversed copies ('Reflected'). This has a visible effect only on objects where the gradient end stops are placed inside the element, and determines what happens to the parts of the object beyond the stops. In the case of 'None', the end stop colors are simply continued, so you won't end up with a big gap beyond the handles of the gradient, whereas the other two options repeat the gradient stops. In no case does the gradient simply stop where the end points are placed, so you won't end up with an object that has unfilled corners (unless your gradient is designed that way).



In this example, I've used the same red-white gradient in both linear and radial modes. The end stops have been moved to be well within the outlines of the squares, and you can see how the three options deal with coloring the areas beyond those stops.

Below that first row of controls in the dialog, we get to a wide colored track that previews the gradient itself, with handles below it indicating the stop positions and colors. Note that you'll still see a linear representation of the gradient here, even if you're actually editing a radial gradient. That's not usually a problem in practice, but you may find it easier to still make some changes to the individual stops on the canvas where you can at least see the gradient used in context.

Each of the gradient stop handles can be moved – even the end ones – by dragging them along this track. They can't pass each other, though, so if you've got three stops set to red, green, blue, then no amount of sliding will change the order if you actually wanted green, blue, red. Instead, you'll have to set each stop color separately by selecting the handles one-by-one, and using the color picker section below. When a stop is selected, a small circle is displayed in the handle.

New stops can be added by double-clicking in the track, or removed by selecting them and pressing the Backspace or Delete key on your keyboard. The editor doesn't allow for single-stop gradients, however, so you won't be able to delete the first or last stops if there are only two stops in the gradient. If dragging the handles to position the stops is too coarse for your requirements, it's also possible to adjust the selected stop's position numerically using the Stop Offset control below the gradient track. The number in this box runs from 0.00 at the left of the track, to 1.00 at the right. Even this approach prohibits you from swapping stops around, though, so don't think you can sneak that green stop into first place just by nudging the red one up a bit, then setting green to zero.

The individual stops are also displayed as a vertical list at the left of the display. It's possible to hide this list using the 'Stops' control just above, though I can't see any particularly good reason why you would want to. The selection in this list is kept in sync with the selected handle on the preview track, and vice versa, so you can just select the stops using whichever is most convenient at the time.

This list also provides plus and minus buttons at the bottom. The latter is pretty self-explanatory: it deletes the currently selected stop, unless there are only two remaining. The plus button is slightly more complex: when a stop is selected, clicking this button will create a new stop placed exactly halfway between the selected stop and the next one. The only exception is if the last stop is selected, then the new stop is added halfway between the penultimate stop and the last one. When a stop is added (even if via a double-click on the track), Inkscape sets its color to the existing value of the gradient at that point. This ensures that the gradient remains undisturbed by default, until you start to shift the stop around, or change its color.

The last part of the gradient editor is the large color picker to the right of the stop list, used for setting the stop's color. As this reflects the same style and operation as the new color picker for flat fills, I'm sure you can work out how to operate it without any further help from me. Just note that SVG gradients can include translucent or transparent stops. If you actually want opaque colors, then make sure to set the Alpha channel accordingly – I've been caught out more than once when I've found that the 'white' in my gradient was actually just the page color showing through, causing problems with my PNG exports.

As a frequent user of gradients, I'm extremely pleased to see the return of a dedicated editor – and very happy with the way it's turned out. I would like to see the addition of a context menu to the color stops, however – providing a convenient way to select common colors (e.g. black, white, most recently used), or to set one stop to the same color as another without having to resort to copy-pasting the hex code. For that matter, being able to drag and drop palette entries onto stops would be a nice addition, too. But those are just items for the wish list, and aren't meant to undermine the great work that the developers have already put into this feature.

As great as the color pickers and gradient editor are, there's yet more that has been added to the Fill & Stroke dialog! So far we've looked at the controls that are common to both the fill and the stroke, but the Stroke Style tab carries a few things that are specific to strokes alone. But, again, the word count catches up with me, so the additions and changes in that tab will be the subject of next month's instalment...

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

FreeCAD - Part 2

Welcome back! Ok, I need you to create a new project, a new body, a new sketch, add a centered rectangle, and stop there in the Sketcher workbench.

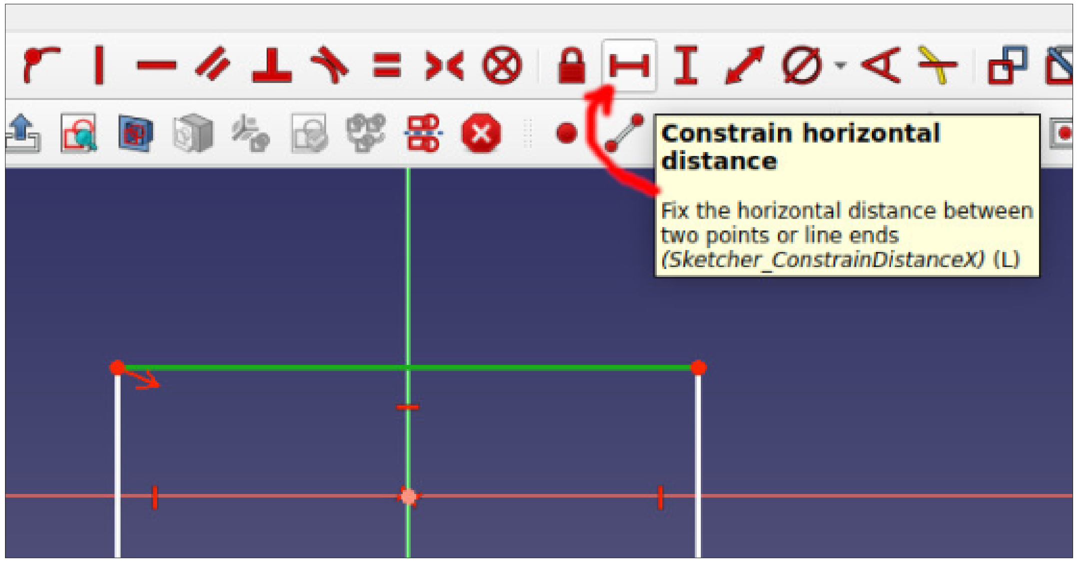
Let's look at getting this piece locked in place and adding (or removing) some bits.

If you look in the left panel, you'll see a bit that says '2 DoF(s)'. This means two things aren't locked in place. Not the end of the world, but it means items could move and mess things up later.

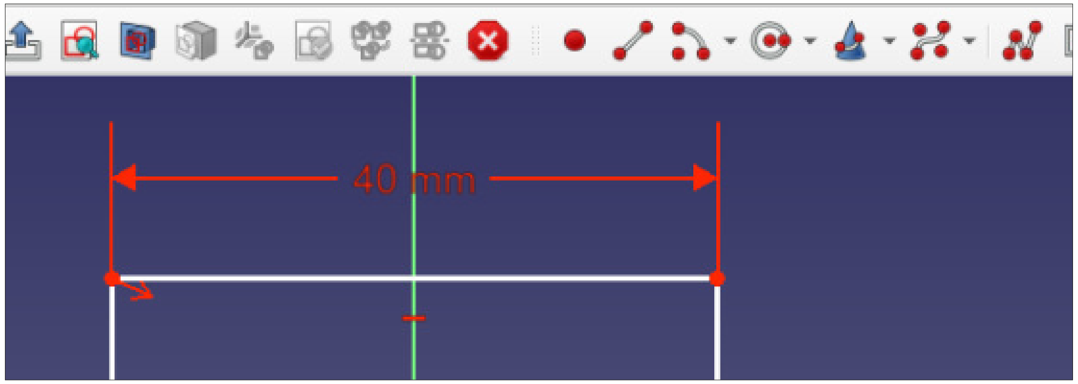
You can click the '2 DoF(s)' link and it'll show you what's not locked, but I can tell you that it's all four corner points. In other words: we can move all four sides. The middle is locked in place as we did that by choosing to make the rectangle from the middle point outwards.

So, left-click on the top edge of the rectangle and it'll change to green. Now left-click the Constrain horizontal distance icon. This will give us a popup that'll let us lock that size in place.

I'm going for 40mm.


Constrain horizontal distance

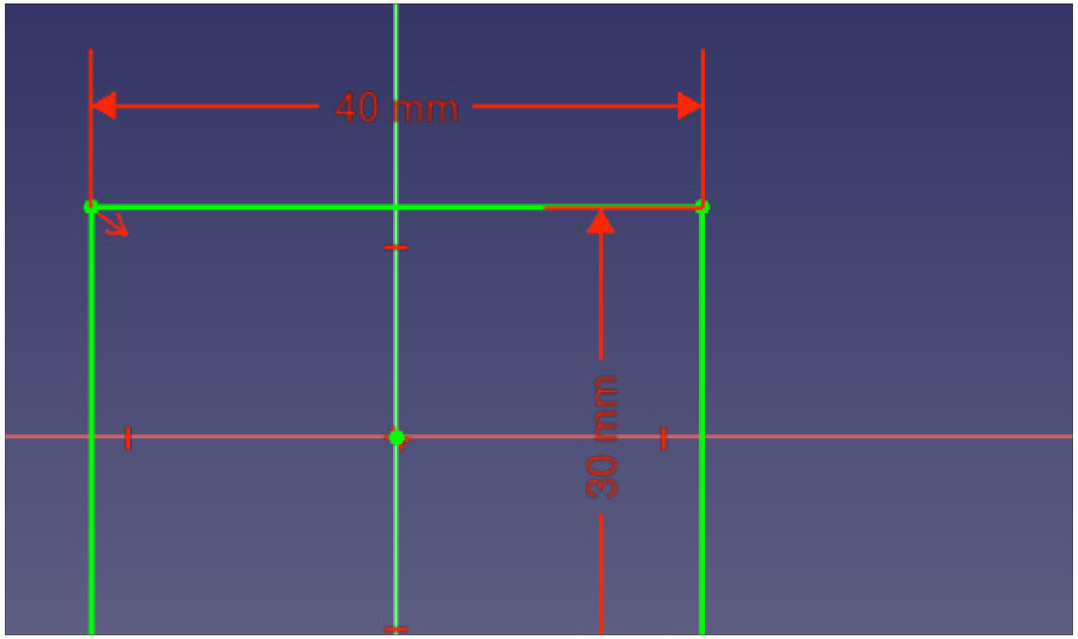
Fix the horizontal distance between two points or line ends
(Sketcher_ConstrainDistanceX) (L)

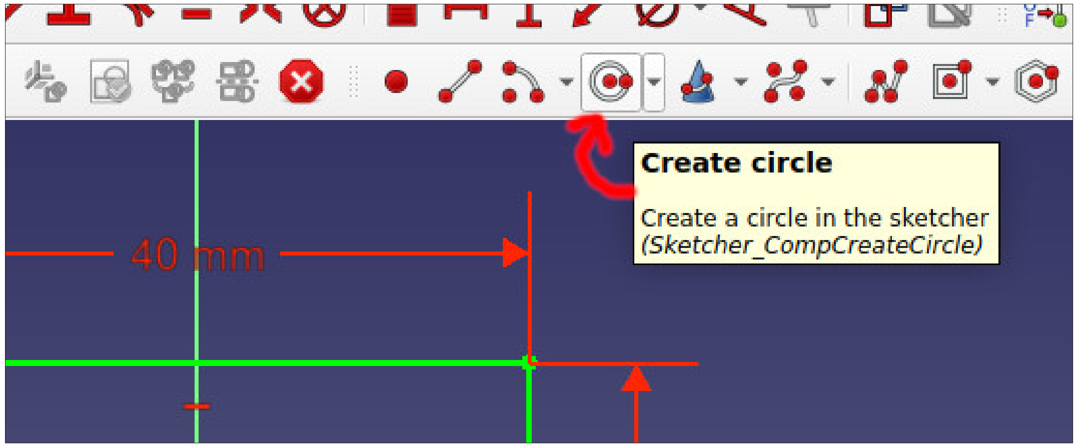


You'll see that we now have a size on our drawing.

Let's do the same for the left or right side, but choose the vertical icon beside the horizontal and make it 30mm.

Lovely. Now you'll notice the whole thing is green. This is because it is locked in place and can't be moved by accident. The left panel also says 'fully constrained'.





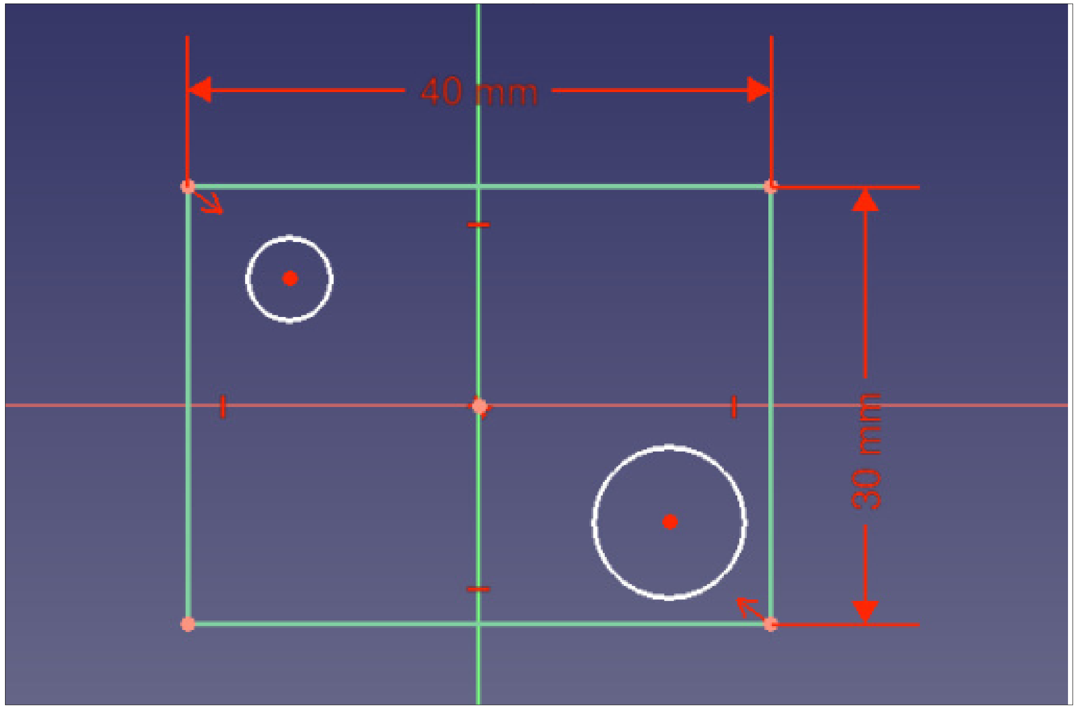
I'm not keen on the 30mm text being on the inside though. You can click and drag on it to move it outside the box. I prefer that.

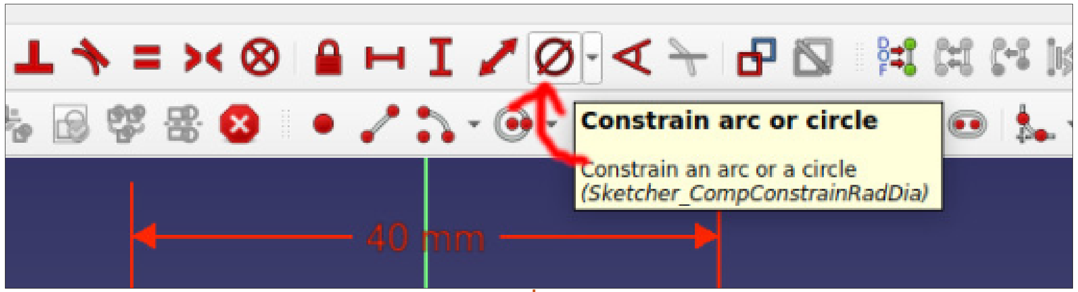
Now, you could close that and go back to the Part design workbench, and pad it, but let's add some circles inside it. We'll think of those as screw holes for now.

Click the Create circle icon.

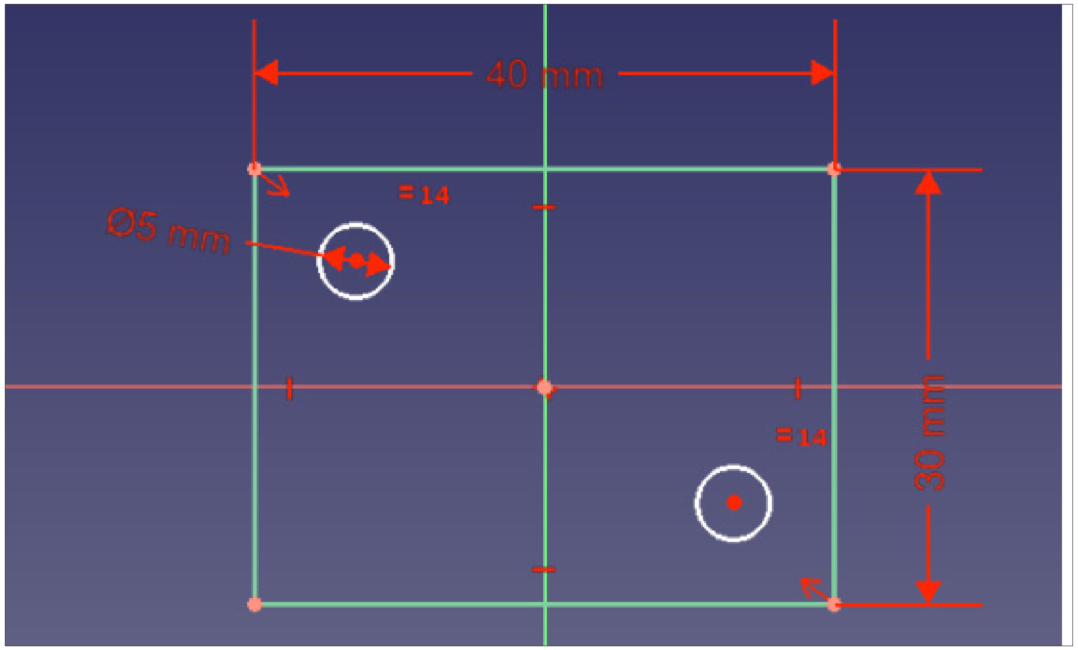
Same as the rectangle, we'll need two clicks to draw the circle. One in the center and one on the outside edge. So, make two circles inside the rectangle. I'll put one top-left and one bottom-right. Don't forget to right-click (or Esc) to come out of creating circles mode.

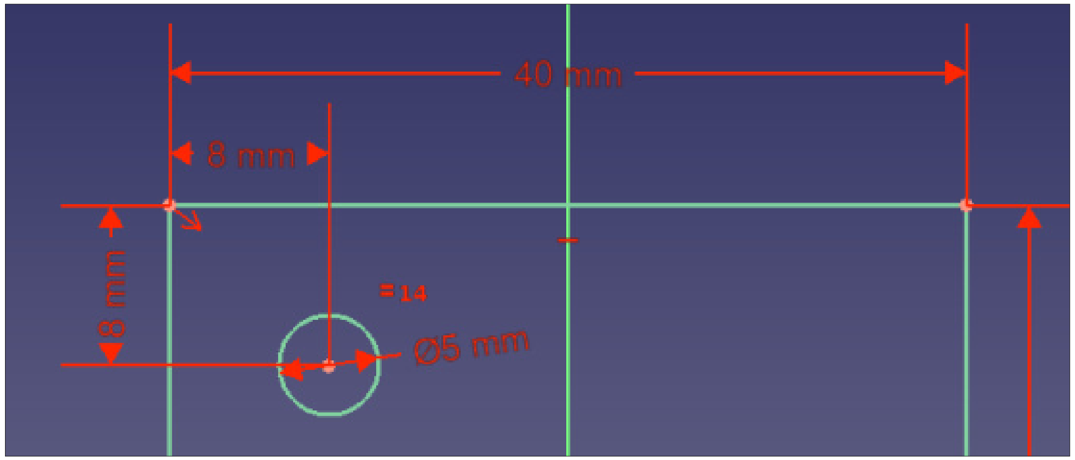
If you need to move the circles, you click and drag on the center point. You can click and drag on the circle outline to resize it.





Let's lock a size in for these circles. Click the outline and choose the Constrain arc or circle icon. This is currently set for diameter, but you can click the down-arrow and choose to make it a radius if you like. I'll make it 6mm. I need the other one to be the same. So click the outline of the 6mm one we just did, hold down the CTRL key, and click the other circle. Now click the





Constrain equal icon. It looks like a big red equals sign (=). Voila! The two circles are the same.

If you double-click the 06mm label on the top-right one, you can change it, and it'll change them both. Try it. Set the top-left circle to 5mm.

We need to constrain these to make sure they're in the right place. Click on the center point for the circle top-left. Hold down CTRL. Click the top-left corner of the rectangle and choose the horizontal constraint (like we did with the rectangle sides). Let's go with 8mm. Select the same two points and choose vertical constraints and choose 8mm again.

I'll let you figure out how to do the other circle.

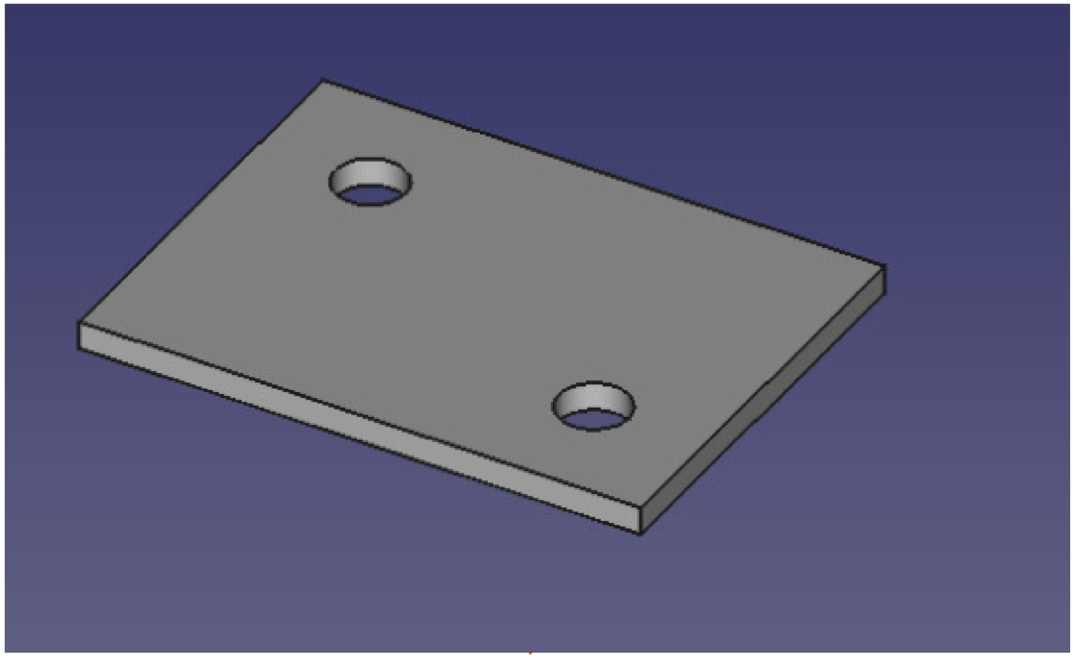
So now our drawing is green and all locked down.

Again, like last time, click Close in the left panel.

This time, though, look at the top icons in the Part design workbench. See the yellow and red ones?

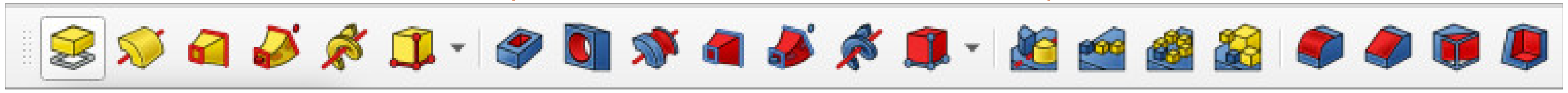
Well, yellow means add, red means remove. That first one with the yellow box is the pad icon. It does the exact same as we did last time. Click the pad icon and make it 2mm this time.

If you want to marvel at your creation you can use the middle mouse button (or scroll wheel held down) to move the model about. Hold both middle and right buttons to rotate.



On the left panel, click the right-arrow beside Pad. This will show a greyed out Sketch below it. Double-click the Sketch to go back into it.

Feel free to add another two holes to either corner of the rectangle to give four holes. We'll start with that four hole plate next time.



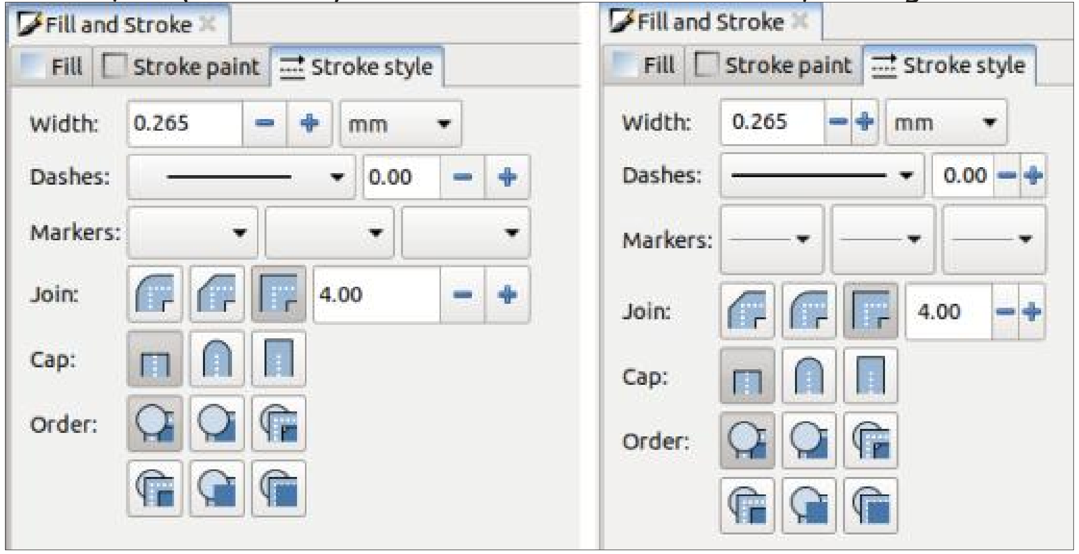
Ronnie is the founder of Full Circle and, somehow, still editing this thing. He also paints, draws and does woodcarving in his spare time.

contents ^

Written by Mark Crutch

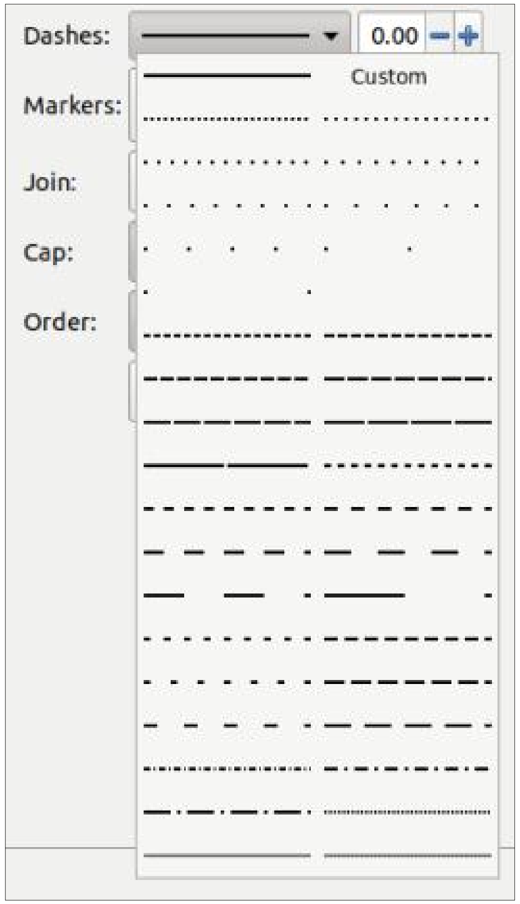
Inkscape - Part 131

It's hard to believe that I'm now onto my third article about the changes and additions that Inkscape 1.2 brought to the Fill & Stroke dialog, yet there's still plenty to write about. I love that: this dialog is one of the most frequently used parts of Inkscape for almost any workflow, so the fact that the developers have chosen to put some serious effort into improving it should bring benefits to every user. Previously, I looked at the new color pickers and the return of a gradient editor. Now it's time to turn my attention to the changes in the Stroke Style tab. Let's begin by comparing the old version, 1.1 (below left) to the new version, 1.2 (below right).



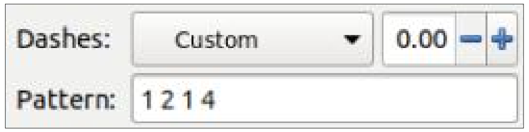
I sense you may be a little disappointed with that grand reveal. On the surface little has changed. The Markers buttons now show a thin line when no marker is selected, and the order of the Join buttons has changed to better match that of the Cap buttons, but otherwise things are about the same, right?

In truth, there have been two significant additions to this tab, but they both hide behind pop-up menu buttons. The first is the Dashes pop-up, which now shows as a two-column list, making it easier to see all the dash options at once, especially on smaller screens.



While I like the idea of a two-dimensional display for the dashes, I'm not sure the current arrangement works terribly well. The 'order' of the entries runs along the rows first, rather than the columns. This makes it difficult to see the relationship between groups of patterns that were previously vertically adjacent to each other. Using a column-first order would have been a better choice, I think. Better still would be to take the new design a step further with a three-column layout: one for dots, one for dashes, and one for mixed patterns.

The Custom option, which used to live at the bottom of the list, has been promoted to the top – and with good reason. In earlier versions of Inkscape, this was a misleading option to say the least. It was more of an output than an input; an indicator, rather than a control. If you were to load an SVG file with a custom dash pattern, then Inkscape would switch to this option in the UI. But manually selecting it for your own Inkscape-drawn lines did nothing, and certainly didn't allow you to create a custom dash pattern (other than via a trip to the XML editor). That has now changed, and selecting this option expands the fields in the dialog a little, adding a 'Pattern' field.



Dashes: Custom 0.00
Pattern: 1 2 1 4

The content of this field needs a little explanation, if you're not intimately familiar with the way dash patterns are defined in SVG. It takes a series of space separated numbers. The first number is the length of the first dash (or dot – which is really just a small dash in this case). The second number is the length of the space that follows it. The third is the length of the second dash, and the fourth is the space that follows that. And so it continues, with odd entries defining the lengths of the dashes, and even entries defining the length of the spaces.

Fortunately there is a shortcut built into the spec to make your life a little easier. There's no need to list enough entries to cover the entire length of your path: the pattern is automatically repeated for you. Entering '1 2 3 4' is exactly the same as entering '1 2 3 4 1 2 3 4 1 2 3 4…'. But this also has a significant effect if you enter a series with an odd number of entries: this automatic repeating means that it's the same as entering a series with an even number, in which the second half is a duplicate of the first. So '1 2 3' becomes '1 2 3 1 2 3…'. The significance of this is that any odd entry in the first half (denoting a dash) turns into an even entry in the second half (denoting a space), and vice-versa.

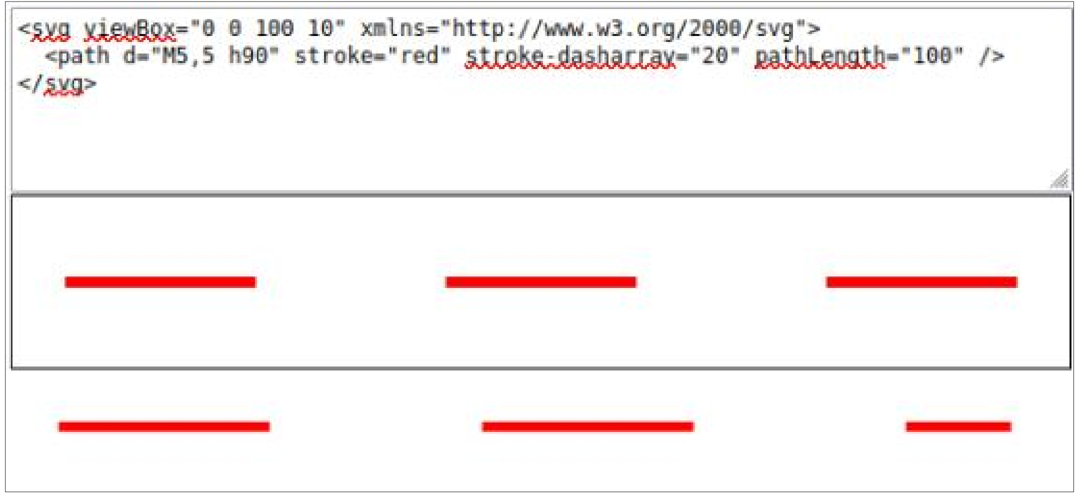
From this you might have worked out that the simplest dash pattern is just a single number. '5', for example, would actually be equivalent to writing '5 5 5 5 5…' - i.e. a series of equally sized dashes and spaces, each 5 units in length.

Ah yes, the units. You may have noticed that all my examples have been simple numbers with no units. The SVG spec states that the numbers are all in 'user units', which should relate to the viewBox used in the SVG. Unless you've been messing around with that in the Document Properties dialog, Inkscape will generally set the viewBox dimensions to be the same as the document size. So if your document is defined in mm (e.g. an A4 page), then each 'user unit' will be 1mm. At least that's the theory. But, in practice, it doesn't seem to work like that, so you're probably best just using a bit of trial and error to get the parts of the pattern to be the lengths you want.

The SVG spec also supports percentages in the dash pattern, but Inkscape does not appear to do so. Percentages are actually less useful than you might expect for this purpose. It would make sense for percentages to be based on the length of the path segment. That would allow you to enter a single value of '20%' in order to get three equal length dashes (with two matching spaces between them), which exactly fills the length of the path. But for reasons that I can't fathom, the SVG spec uses the size of the viewport (i.e. the window) as the basis for calculating percentages, making them practically useless, in my opinion.

There's a workaround for that limitation, in SVG terms. You can add a 'pathLength' attribute to a <path> element which tells the renderer to override the actual path length when performing some calculations – such as defining the dash pattern. By setting pathLength to 100, any values entered in the dash pattern should therefore work as percentages. Here's an example SVG file which draws a 90 unit long path, but uses pathLength to tell the browser to treat it as 100 units, so that the dash pattern value (in the 'stroke-dasharray' attribute) is effectively treated as 20% of the path length.

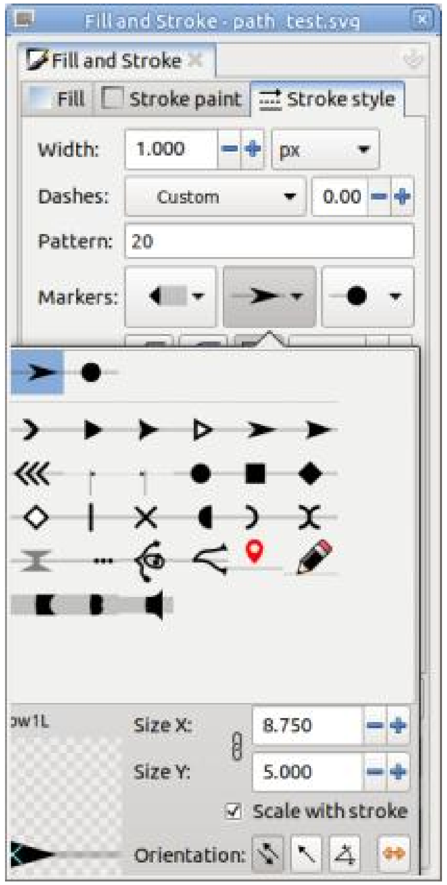
The top line is the output when this path is rendered in Firefox. You can see that there are three equally



```
<svg viewBox="0 0 100 10" xmlns="http://www.w3.org/2000/svg">
  <path d="M5,5 h90" stroke="red" stroke-dasharray="20" pathLength="100" />
</svg>
```

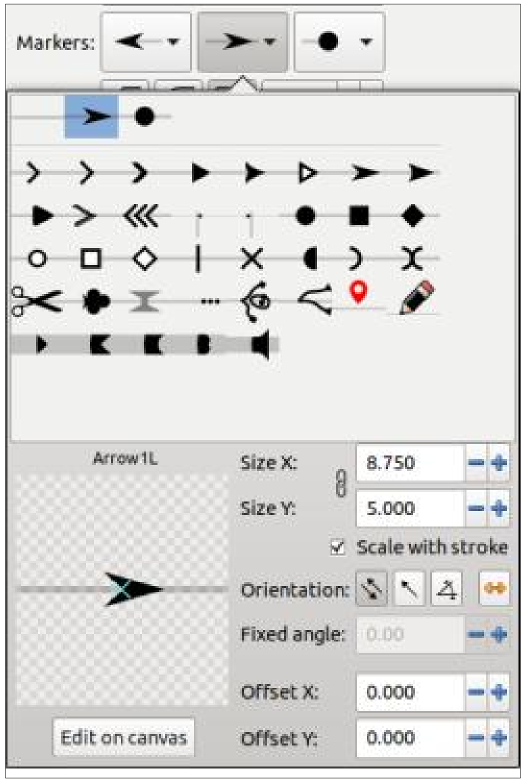
sized dashes, with two equally sized spaces between them, as expected. The bottom line is the same file rendered in Inkscape. The pathLength attribute is ignored, so the pattern does not fit exactly into a 90 unit path, resulting in longer dashes and spaces, with the last dash being cut-off. It's a real shame that this attribute isn't honoured in Inkscape, as it's the only way to create dash patterns that always fit the path exactly. For now you're stuck with some trial and error in the dash pattern field, and you'll have to resign yourself to repeating the process any time the path length changes.

You may have noticed that the dash pattern always starts with a value for the first dash, with the first space coming after it. What if you want your pattern to start with a space, followed by a dash? There's no way to specify that in the pattern itself, but the spinbox to the right of the Dashes pop-up button lets you apply an offset to the rendering of any dash pattern, whether it's one of the built-in options or a series of your own devising. This can be used to nudge your pattern forwards or backwards to make it start with a space rather than a dash (or, indeed, with a half-dash or quarter-space, or any other adjustment you care to make).


Fill and Stroke - path_test.svg

Width: 1.000 px
Dashes: Custom 0.00
Pattern: 20
Size X: 8.750
Size Y: 5.000
☑ Scale with stroke

Moving on from dashes, the pop-ups for the start, mid and end markers have also seen a major overhaul in version 1.2, turning markers into a hugely more versatile tool than they previously were. Unfortunately, if you're on a small screen, you might find a bit of a problem with the new UI. My personal preference for dialogs is to have them floating in separate windows, rather than docked to the main window. This goes doubly for a small screen, as it gives me the ability to move them out of the way easily or, courtesy of my window manager, to 'shade' them (i.e. collapse them to just the title bar) when I'm not actively using them. But if you try to open one of these pop-ups when the Fill & Stroke dialog has been reduced to its smallest practical size, you will find that many of the controls are cut off.


Arrow1L
Size X: 8.750
Size Y: 5.000
☑ Scale with stroke
Fixed angle: 0.00
Offset X: 0.000
Offset Y: 0.000

The only workarounds for this that I've found are to either make the dialog larger, or to dock it at the side of the Inkscape window. In either case, clicking one of the marker fields will open the full pop-up.

Until you select a marker type, the bottom half of the dialog will be disabled. The first icon, at the top-left of the pop-up, is always just a thin gray line. This is the 'no-marker' option, and is the thing to click if you wish to remove the current marker selection. Following that, on the top line, is an entry for each marker definition that is currently used in the document. This makes it much easier to re-use the same marker for the start and end of the line, or to maintain consistency throughout a document. Do note, however, that using this option will 'link' all the uses of that marker to a single reference in the SVG; if you subsequently edit one of the markers (see below), all the linked versions will also change. For that reason it's often best to choose your marker from the lower part of the palette, not from this 'currently used' section, if you think there's any chance you might want to edit any of the markers later. This section will wrap onto multiple lines, if necessary, and is also where you'll find any custom markers you've created using the Object > Objects to Marker menu entry.

Below a horizontal separator that, oddly, doesn't stretch to the full width of the pop-up, you'll find the complete set of markers that are built into Inkscape. Unfortunately, I haven't been able to find a practical way to edit this collection. If you need to use the same custom markers regularly, the best approach seems to be to create your own default SVG file or template which contains them.

There are some interesting things to note about the marker icons in this section. The first is that each is presented with an associated gray line. This represents the stroke of the path on which the marker will be applied, and carries a couple of useful bits of information: the relative size of the marker compared with the line (e.g. those on the last row are the same width as the stroke thickness, so appear more like line caps than markers), and the position of the marker relative to the line (e.g. the pencil is offset to one side of the path).

It's also worth noting that the position of each icon on the gray line, and the direction the marker faces, will be different depending on whether you're selecting a start, end, or mid marker. For example, with mid markers, the gray line passes right through them, whereas for the others, the icon shows the marker placed at the appropriate end of the line. This is a subtle bit of UI design, but it helps to make more sense of how the final marker will appear in your document.

Once you've selected a marker, the bottom section becomes enabled to allow you to edit it to some extent. The options are mostly self-explanatory, and the preview pane on the left gives you a good idea of what the end result will look like as you play around with the spinboxes. The 'Orientation' buttons in the middle are worth some extra explanation, though.

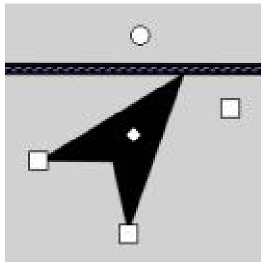
These all deal with which direction the marker will face in. For some shapes, such as circles, these will make little or no difference. But for something like an arrow head, these buttons are essential to ensure that your markers point in the right direction.

The second button is the simplest to explain: this makes the marker point in the direction of the path, from start to end. With this enabled, using an arrow head as a start marker will show it as pointing in the direction of the path's end, which is probably not what you want in this case. For mid and end markers, however, this is usually likely to be the best choice.

The first button behaves similarly, but flips the direction by 180° for start markers. If you find your start maker is pointing the wrong way, switch between the first and second buttons as necessary. Initially you might think that the last button is the right one to use in this case, but it's probably not. That button flips the actual path definition, not the way it's used in one specific instance – which means it also flips the direction of any other uses of that marker within the document (i.e. if you selected the same entry using the 'current document' line of markers at the top of the pop-up).

The third button simply allows you to specify a fixed angle for the marker. Note that this is an absolute angle, not relative to the path direction. For example, if you set this to -90° to make mid-marker arrows point upwards, they'll point directly upwards for every node, regardless of the twists and turns of your path. If you actually need a marker to run at an angle that is relative to the path direction, rather than the page, you'll need to create your own marker that is suitably oriented, use Object > Objects to Marker, choose it in the pop-up, and make sure to use the first or second button in this section.

In the bottom-left of the pop-up is an 'Edit on canvas' button. Click this and the on-canvas version of the marker will gain some basic editing handles. If you have multiple mid markers, these handles will appear on the first one. Using any of the three square handles you can resize the marker (hold Ctrl to scale proportionately); the circle is used to rotate it; the



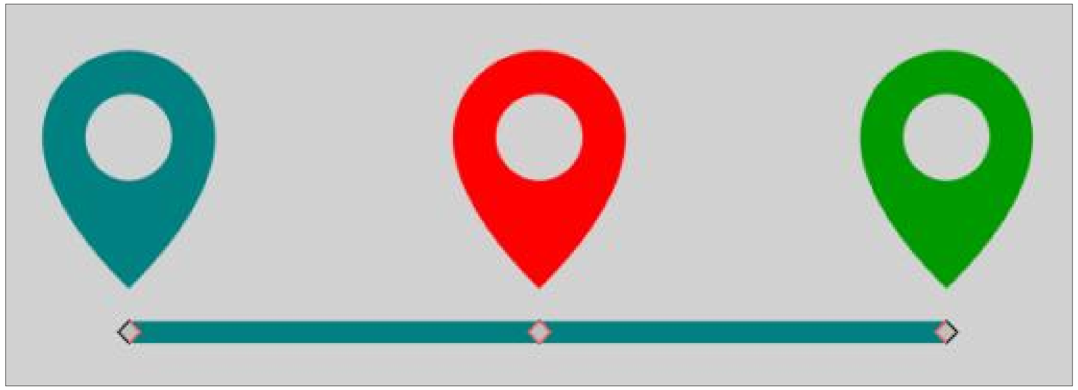
diamond in the middle can be used to move it, adjusting the offset from the path.

This will usually be a more intuitive way to adjust the size, position and rotation of the markers. Note that adjusting the rotation will automatically switch the 'Orientation' option to the third button, so steer clear of that circular handle if you need your marker to follow the direction of the path.

Finally, a note on color. SVG2 defines two keywords, 'context-stroke' and 'context-fill', as a shorthand for the stroke and fill color of the current object, respectively. Many of the markers make use of these to determine what color(s) should be used to render them. As a general rule, arrows and terminators tend to follow the stroke color (which is usually what you want), while the markers which appear with a white fill in the pop-up also use the fill color (which may not be what you want at all). If you wish to use different colors for your markers, you'll need to dig around in the XML editor, find the marker definition, then drill down to the path itself (or some other part of the design, as needed) and modify the style attribute. It's not for the faint-hearted, so make sure you save a backup of your file before you start messing around with the XML editor like this.

A particularly fine example of the context-stroke keyword at work is in the marker that looks like a pencil. This combines the keyword with some translucent panels in order to create a pencil whose body matches the stroke color of your path, but which still has some shadows and highlights based on that same color.

At the opposite end of the spectrum is the marker that looks like a location pin from a mapping site. This is hard-coded to always be red. If you want it to match your stroke color (or fill color), find it in the <defs> section of the XML editor, drill down to the path that's inside the <marker> element, and change the 'style' attribute so that the 'fill' property is 'context-stroke' (or 'context-fill') instead of '#ff0000'. As an example, here is an image in which I've set the start marker to a fill color of 'context-stroke', left the mid marker as the default red, and set the end marker to a hard-coded value of '#009900'.



These new features in the Fill & Stroke dialog initially seem like small changes, but I'm sure you can see that they open up a lot of the possibilities that SVG offers, but which were previously hard to use within Inkscape. It's a shame that changing the colors of markers requires a trip to the XML editor, and the technical knowledge that comes with that, but most users probably won't need that level of control anyway.

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

FreeCAD - Part 3

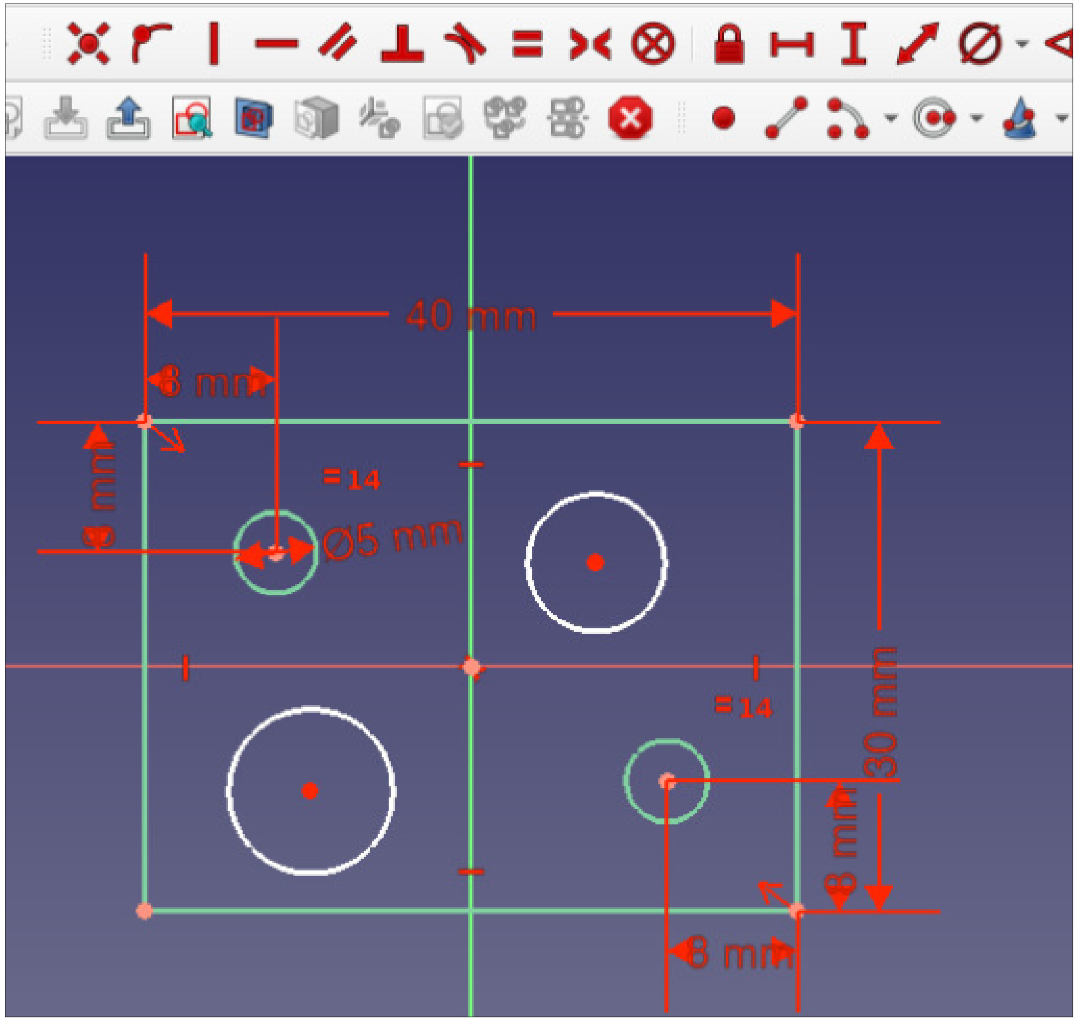
So, you've got your plate with four holes? Good. Now I'll bet you did those additional two the same as the other two? Well, let me show you another, quicker, way.

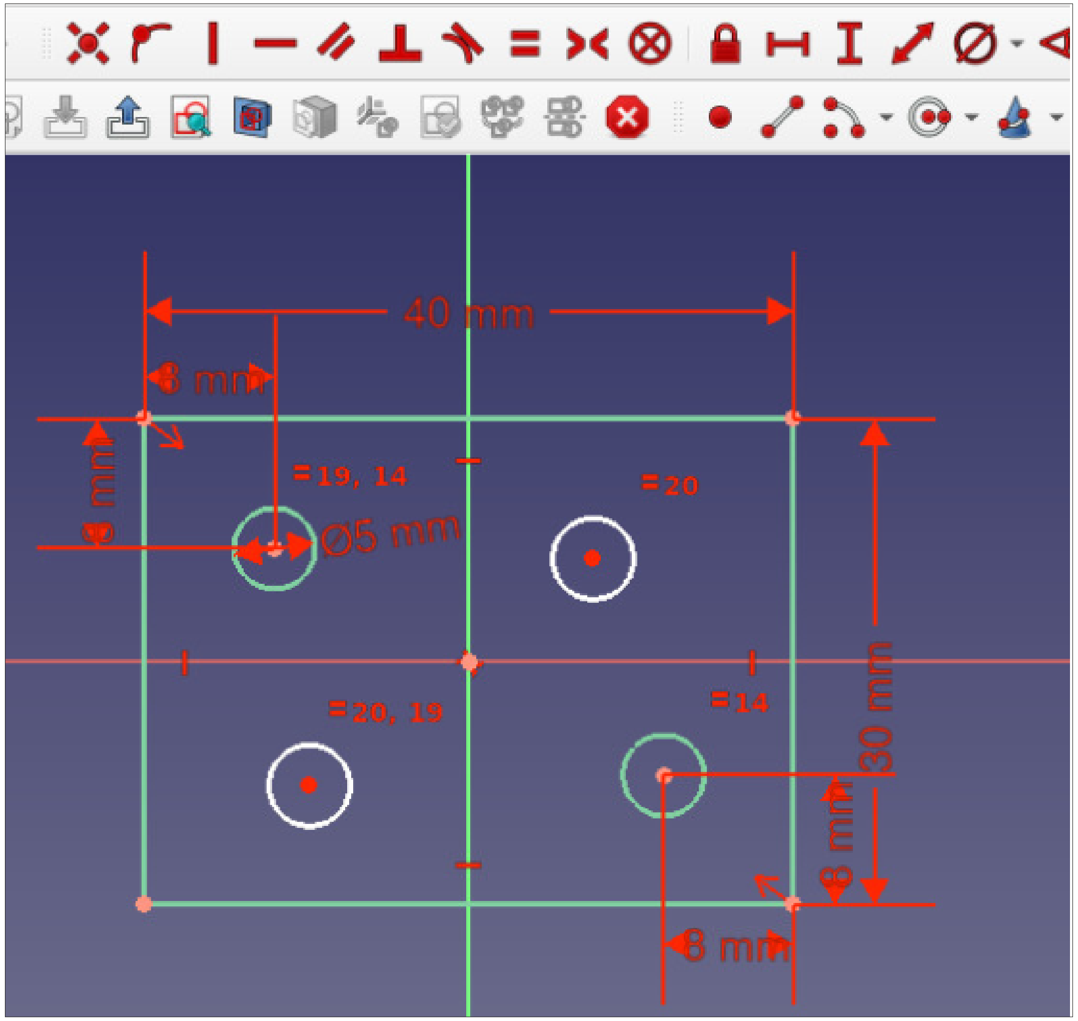
Here's my two additional holes.

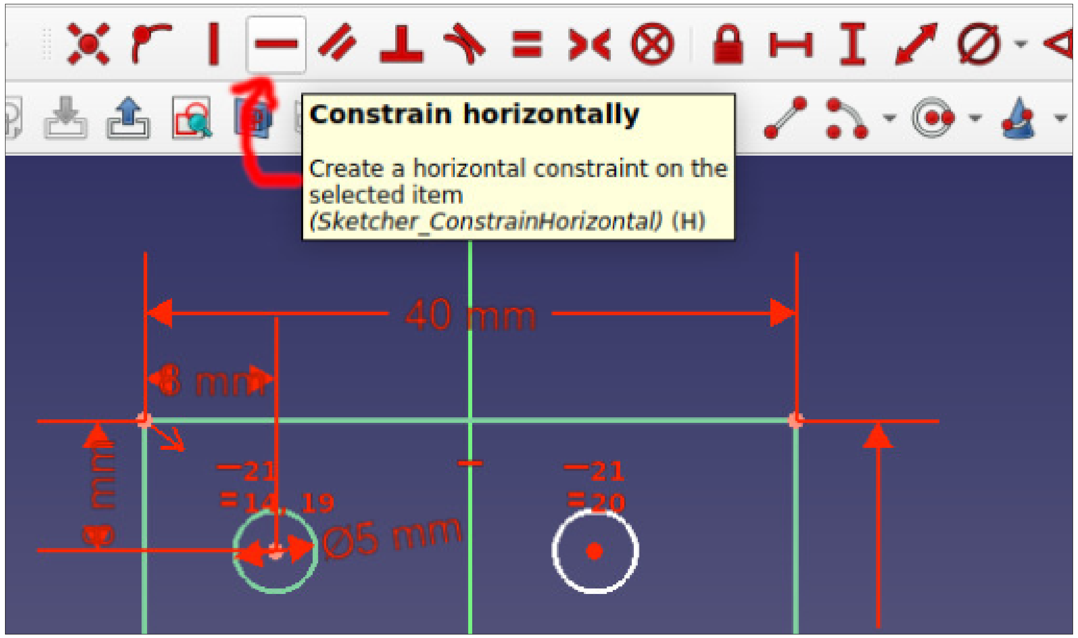
I'll select the outline of the one 05mm constrained hole, then the two new ones. Now select the equals icon. That's them set to the same diameter. So now changing that one circle will change the diameter of all four.

To set them in place, I'll use a new thing: constrain horizontally/vertically. No, not the ones we used last time as they're for distance. I'll select the center point of the top-left circle, then the center point of the top-right one, and click the constrain horizontal icon. Just looks like a red minus-sign.

So now those two circles are lined up. If I do the top-right and





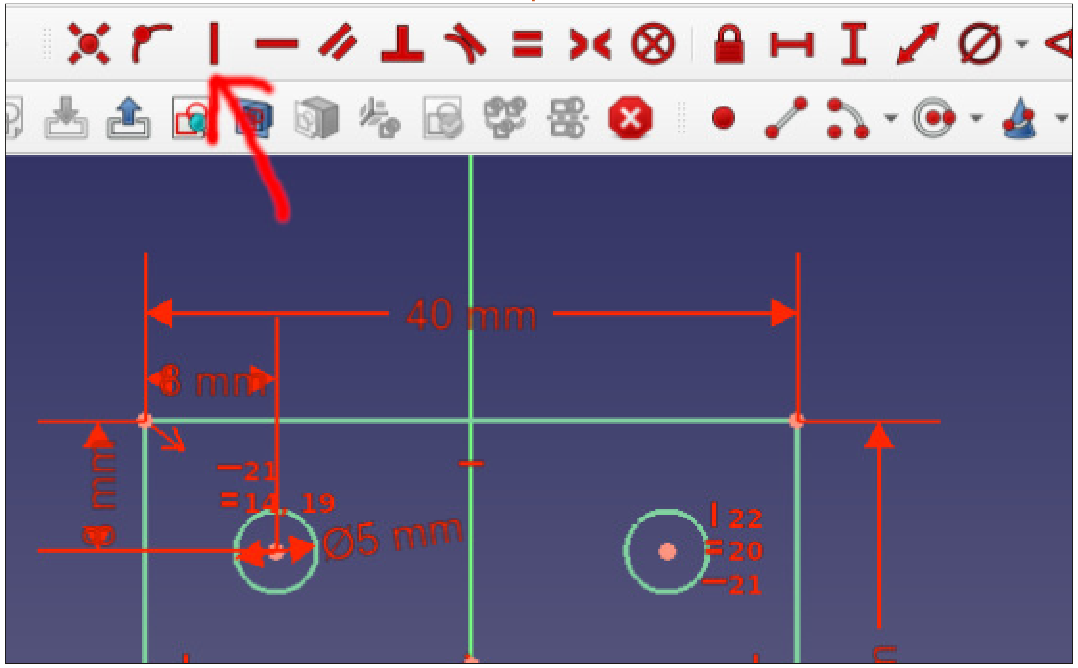


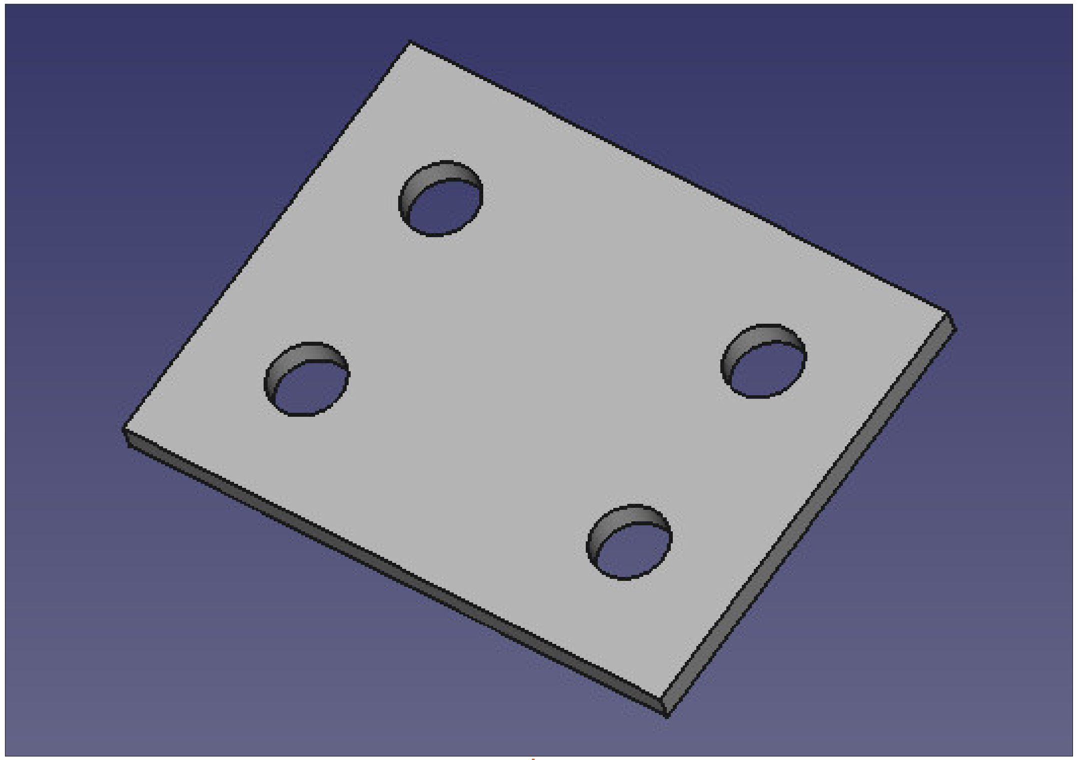
Constrain horizontally

Create a horizontal constraint on the selected item *(Sketcher_ConstrainHorizontal)* (H)

bottom-right center points and do a vertical constraint, then that'll lock that top-right one in place now.

Same deal for the bottom-left circle.

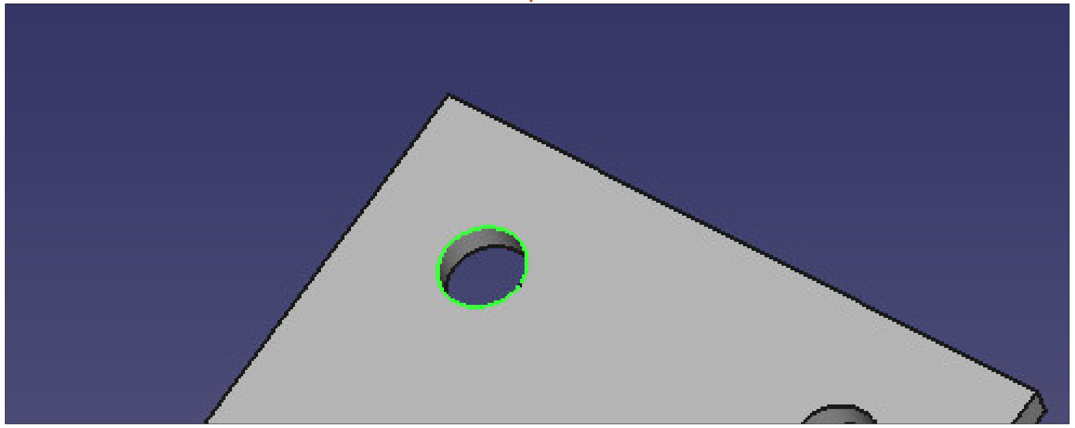


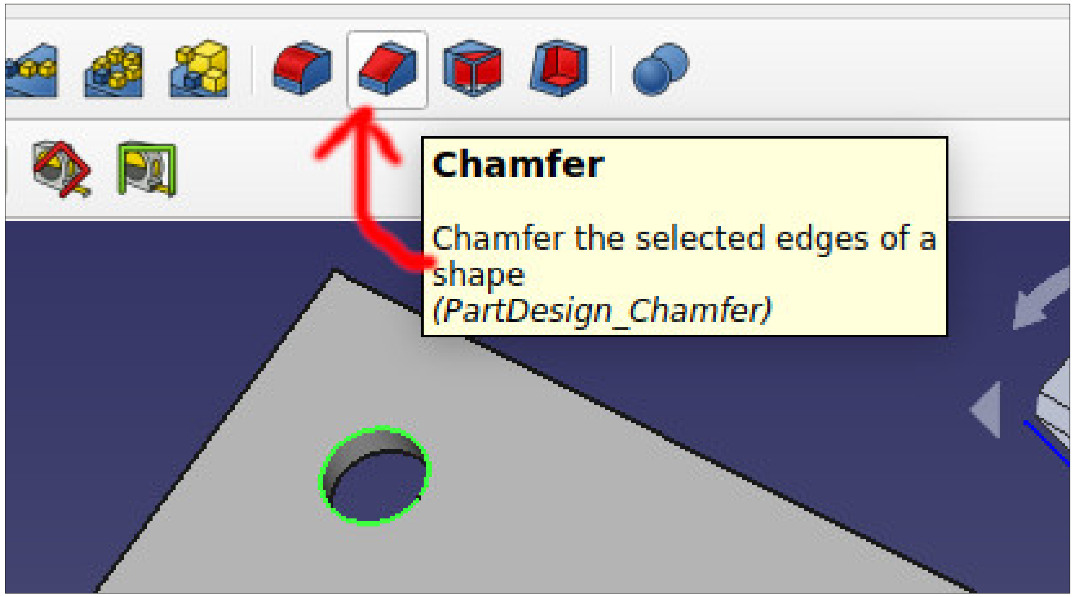


And that's me locked down.

Clicking Close will take me back to the Part design workbench. Looking good.

Let's end this part by making it look a bit more fancy. While in the Part design workbench, we want to click one of the top outlines of one of the holes. Let's say top-left. It'll

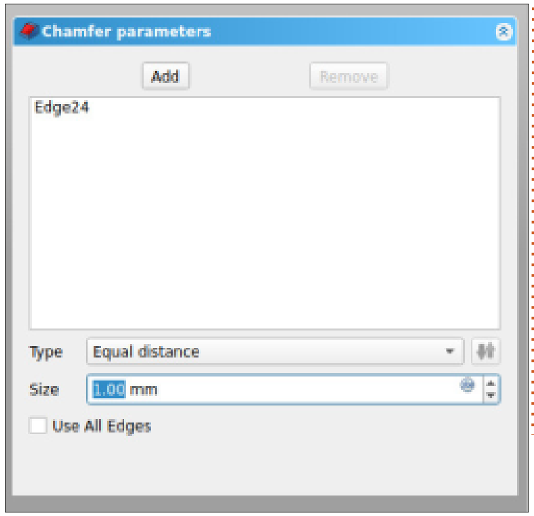




go green.

Now click the red chamfer icon.

Nice! That'll let the screw head sit level with the plate. The items on the left panel will let you edit the size of the chamfer. Or, let's

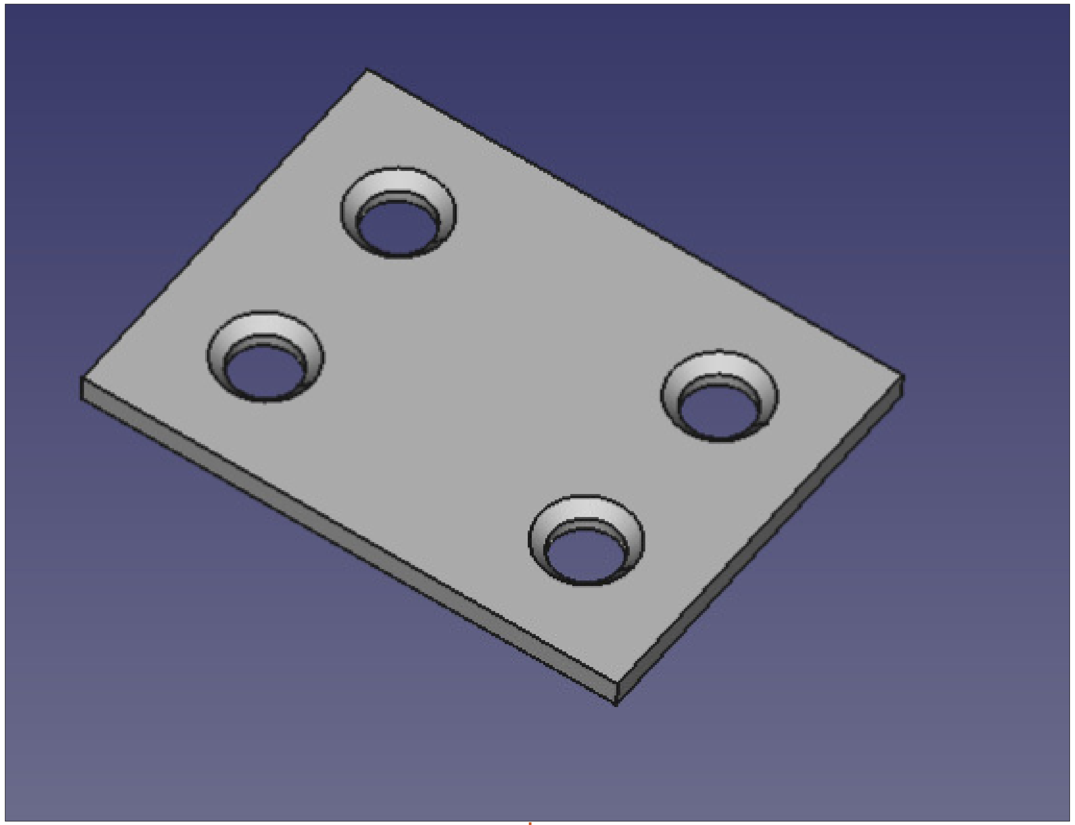


click the Add button and click on the outline of the other three circles. You only need to click Add once. When you've clicked all the outlines you click OK to end adding.

Note on the left panel that Chamfer is now added to the list. Think of this as a list of past actions that you can still jump back to.

Next time, we'll add more to the plate.

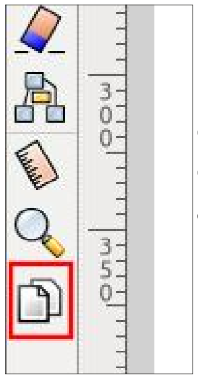
Now I'm not saying you need to, but feel free to click on a top edge of the plate and choose the fillet icon (beside chamfer). Just saying…

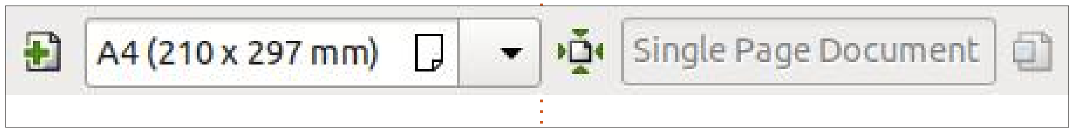


Ronnie is the founder of Full Circle and, somehow, still editing this thing. He also paints, draws and does woodcarving in his spare time.

Perhaps the most important addition in Inkscape 1.2 is the ability to support multiple pages in a single document. This is a much requested feature which opens up a lot more power for working with multi-page PDF files (both importing and exporting), though it's not without its limitations, particularly when used on the web. More on that topic next month. This time we'll take a look at how to create and work with multiple pages from within Inkscape itself.

By default Inkscape opens with its traditional single-page view. Whereas many applications might provide a dedicated dialog or menu options for adding and removing pages, Inkscape treats such tasks as more akin to a drawing task than a management one. Gaining access to the new multi-page features therefore requires you to switch drawing tools, choosing the new icon at the bottom of the toolbox (outlined in red).

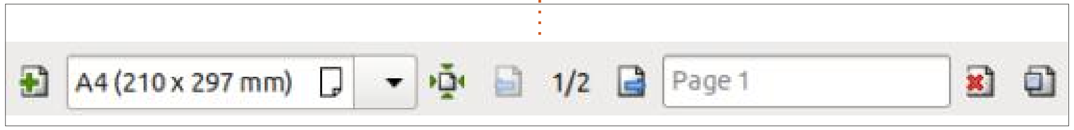




With the correct tool selected your page will gain a small, square handle at each corner, and the tool control bar will change to show the initial set of controls for managing pages.

Using the square handles you can dynamically change the size of the existing page on the canvas. I recommend using the bottom-right handle for this for the simple reason that pressing Ctrl-Z will undo this change 'cleanly'. If you move any of the other three handles then an undo operation will actually cause the page to shift on the screen. The end result is pretty much the same, especially if you pan the canvas afterwards, but the movement is somewhat jarring. It's also worth noting that these handles don't honor the usual Inkscape modifier keys for resizing in proportion, or along a single direction, which is a shame.

Manually resizing the page isn't your only option though. You can use the third control on the toolbar (the page with four arrows pointing to it) to resize the page. If you had no objects selected when switching to the page tool, this button will resize the page to fit the contents; if you had items selected then it resizes the page to fit the selection. Unfortunately there's no facility to include an offset or margin, so if you don't want your page boundary to absolutely hug your objects tightly you'll either have to manually resize afterwards, or add a temporary object to your page that is larger than the combined elements, then delete it afterwards. Unfortunately this same omission is now present in the re-designed Document Properties dialog, which is rather annoying for those of us that prefer to include a little breathing room around our creations.



If you prefer to set your page size to specific dimensions, you can do that using the input field and drop-down combination widget in the tool control bar. Click in the field and enter dimensions with units directly (e.g. 100mm x 100mm). Or select one of the presets from the drop-down list.

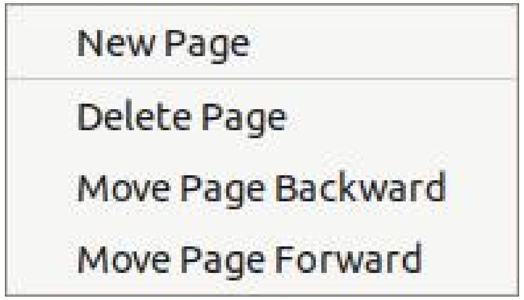
So far all we've looked at with this 'multi-page' tool is how to resize the existing single page. Let's add a second page to our document, by clicking on the first icon in the tool control bar. Inkscape creates a new page on the canvas – you may need to zoom out to see both. The new page will be the same size as the previously selected one. Now that you have more than one page you can click in each of them to change the 'current' page (the one that will be affected by clicks and changes in the tool control bar), or click and drag inside the page area to move it around on the canvas. With more than one page in the document, the tool control bar itself also gains more widgets:

In the middle we have a pair of buttons that allow you to reorder the pages. Currently the first of the two pages is selected, so the text reads “1/2”. There are buttons for moving the order of the page: the left one is disabled, because we can’t move this page earlier than position 1. The right button is enabled, though, so we could click that to move the selected page to position 2 (effectively swapping the page order, since we only have two pages). Using these buttons also moves the pages around on the canvas, as well as changing the order in which they are listed in the SVG file.

The text field reads “Page 1” in grey text. Inkscape gives each page a default label of this form, if you don’t provide one of your own, in order to help you keep track of which page is the currently selected one. Often it may make more sense for you to name some or all of your pages explicitly. Simply type something into the text box, and the corresponding page on the canvas will be decorated with a small tag at the top-left which shows the name you entered. A manually entered label will be stored as part of the SVG document, whereas the automatic labels are not, and are generated by Inkscape when needed. It’s perfectly okay for you to combine both approaches, using manually defined labels for key pages, and falling back to the automatic “Page n” syntax for the others.

The button after the text field is used to delete the currently selected page. However, the exact behaviour depends on the state of the last button. That last button is actually a toggle, which determines how objects are moved when a page is moved around the canvas. With it toggled off, the objects and the pages are treated as two entirely separate things. If you move a page, or even delete it, the objects themselves remain unaffected.

Toggle it on, however, and things become more complicated. In that state, any object that intersects with a page that is moved or deleted will also be moved or deleted. The important word there is ‘intersects’. When an object is wholly contained within the page boundary, the behaviour makes sense. Move a page, and the contents of the page move with it. Delete a page and its contents disappear too. But this also applies to any object that overlaps with the page boundary, even a little. It might be 99.9% on the canvas, barely touching the page, but it will get dragged around or deleted just the same as something entirely within the page. This can be particularly troublesome when pages are positioned close together, and an object ‘bleeds’ out of one, and slightly into another. Moving that second page will drag the object with it, even if it was meant to be kept on the first page.



Unfortunately there’s no fine control with this. Either every intersecting object moves (button toggled on), or none of them do (button toggled off). There’s no way to ‘de-select’ specific objects prior to the move, or anchor them more tightly to the correct page. In practice it’s not usually a huge problem, but it’s definitely something to be aware of, especially if you keep your pages close to each other, but let your objects bleed out of the boundaries.

If you do manage to accidentally move or delete an object, the usual Edit > Undo (Ctrl-Z) option will generally save you. But the multi-page code is still new, and I have experienced a few crashes when using it, so I wouldn’t rely on that safety net too heavily. As usual I recommend saving frequently and making sure you have autosave enabled (Edit > Preferences > Input/ Output > Autosave).

With the Page tool selected you actually have other options besides the tool control bar for performing some of the page-related tasks. You can create a new page simply by drawing a rectangle on the canvas. This lets you create new pages that aren’t the same size as the currently selected one – but you also don’t get accurate control over the dimensions. Having created the page, you can select it and then change the page size from the input field and drop-down in the tool control bar – but if you’re going to do that then I’m not sure I see much benefit in drawing out the page on the canvas first anyway.

Deleting a page can also be done simply by pressing the Delete key on your keyboard, as you might imagine. Right-clicking on a page will bring up a context menu with a few common options, too:

There's no context menu when you click on the empty canvas. It would have been nice to at least have a 'New Page' option that creates a new page with the currently selected page's dimensions, but placed at the mouse position.

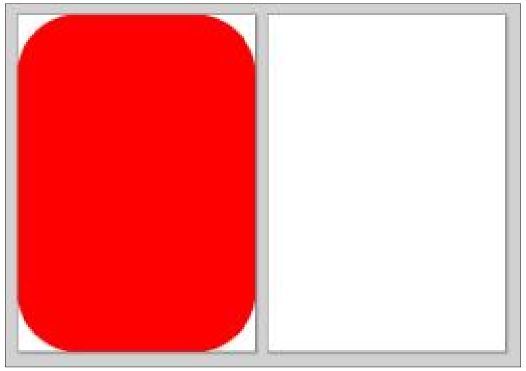
Since this is new code, not every part of Inkscape is fully aware of this multi-page world. I've already described how to change the page size using the page tool, but it's a little disappointing to find that the Document Properties dialog only knows about the first page. It doesn't matter which page you have selected, changing the size in that dialog will only affect the first one. I would actually prefer it if this dialog acted as a 'master' control, setting the size for all the pages. That would be more consistent with the other controls in this dialog, such as the color picker for changing the page background, which does apply to all the pages. As it is, there's no way to select multiple pages if you want to resize them all to the same dimensions, so you'll have to resize each one by hand by adjusting its dimensions in the tool control bar.

Conversely, the Align & Distribute dialog does recognise different pages. If you select 'Page' in the 'Relative To' pop-up, your objects will align to the page they're currently in. If they straddle more than one page it's a little more hit-and-miss as to which one they'll align to. There is some tricky-to-explain logic to it, but it's such a corner case that it's not worth trying to put it into words. If you find yourself in this situation, and your objects align to the wrong page, you should just move them to be well and truly inside the correct page and try again.

Guides in Inkscape have always extended beyond the page boundaries and into the infinite space of the canvas itself. The multi-page world doesn't change that, so any guides you create will happily appear on top of all the pages in their path. This is a good thing, as it makes it easier for you to snap objects to a common baseline, for example. Another good thing is that pages themselves can be snapped to guides, or to other pages, which definitely helps with alignment. They don't snap to objects though, so you may want to use Object > Objects to Guides if you have to create alignment anchors based on your content (don't forget to duplicate the object first!).

On that note, it would have been really nice if the Objects to Guides option did something useful when the Page tool is selected. The obvious thing would be to create guides based on the selected page, while not removing the page itself. That would make it very easy to align pages to each other, but unfortunately the option just does nothing when a page is selected.

A similar missed opportunity is present in the spacing placed between new pages when you create them. Starting from a single page document, clicking the New Page button in the tool control bar creates a second page, then a third, and so on – each of them offset from the previous one with a small gap. Now let's consider the following example: I've created a second page, but haven't moved either of them from their initial positions. I've also drawn a rectangle that fills the first page, though I've rounded the corners so you can see the page behind it.



It's easy to imagine that this rectangle might equally be a border shape, a watermark, a line marking a header or footer, or some other object that you wish to copy to the exact same position on the second page. Your immediate thought might be to copy it to the clipboard, then use Edit > Paste in Place, after switching to the Page tool and selecting the second page. That doesn't work. It just pastes the copy on top of the original object, which is the same behaviour it's always had, but seems slightly wrong when dealing with multiple pages.

Another approach might be to duplicate the object, then try to position it using the arrow keys.

This is where the gap size becomes a problem. A seasoned Inkscape user probably knows the trick of moving things around by holding the Shift key while pressing the arrow keys. This moves objects in discrete steps that are always the same, regardless of your zoom level. I often use this technique when I have to temporarily move an object out of the way, then back again – provided I count the number of keypresses I made, I can be certain it will return to the exact same position, no matter how much panning and zooming I've done in the meantime. This would be an ideal way to move an object to precisely the same position on another page… all it requires is that the gap is an exact multiple of the Shift-Arrow step size.

As you've probably guessed by now, it's not. It's just a little out, leaving your object misaligned one way or the other. Yes there are other ways to align objects on the page. But this small oversight complicates an otherwise simple task.

Having created your multi-page document, you might want to know what you can actually do with it. Loading it into a web browser only shows the first page – but more on that topic next month. You can save as a PDF file, which also preserves any page labels, and successfully handles different page sizes. For anyone trying to use Inkscape as a desktop publishing program, this will likely be more than sufficient. I'd still recommend learning to use Scribus if this is the sort of thing you do regularly – Inkscape works well enough for the occasional one-off, but it's no replacement for a fully-fledged DTP program.

Another use for multiple pages is to help organise a document that contains several images which you may later wish to export in another format for use as assets in a game or website, for example. I'm talking about the old 'Export PNG Image…' option in the File menu, which is now just 'Export…'. The name change is for a good reason: this release adds many more options to this dialog which solve some long-standing complaints about the application. We'll take a look at this using the following multi-page document:

When creating a single export image, the dialog has been somewhat de-cluttered, presenting only the main options. The more obscure parameters now appear in a second dialog when you start the export process, provided you UN-check the 'Hide Export Settings' checkbox. The main Export dialog also includes a preview image that will give you a good idea of what the final export will look like, and in the case of selecting the 'Document' button for a single export, this shows that pages are ignored completely, and the exported file will be a single image for the entire document.

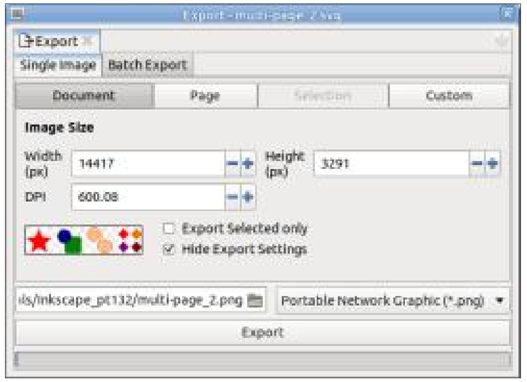
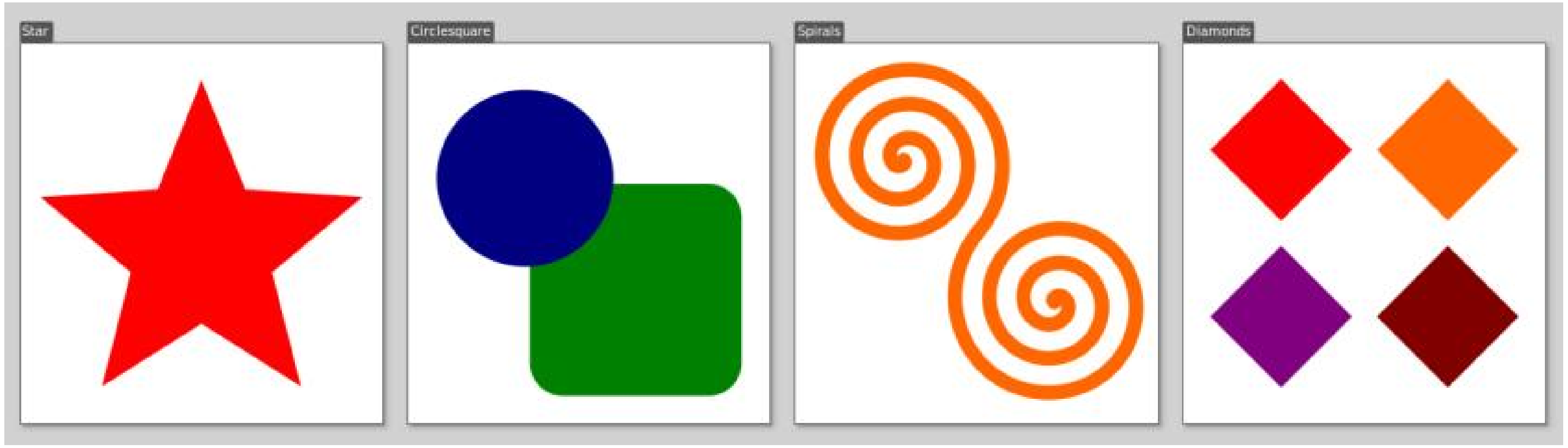
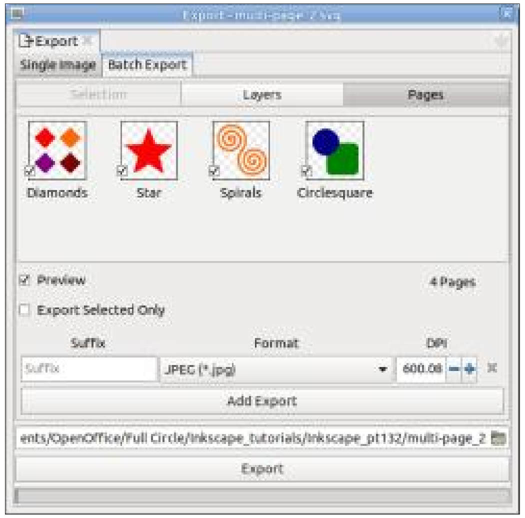


Image Size
Width (px) 14417
Height (px) 3291
DPI 600.08
☐ Export Selected only
☑ Hide Export Settings
ils/inkscape_pt132/multi-page_2.png
Portable Network Graphic (*.png)

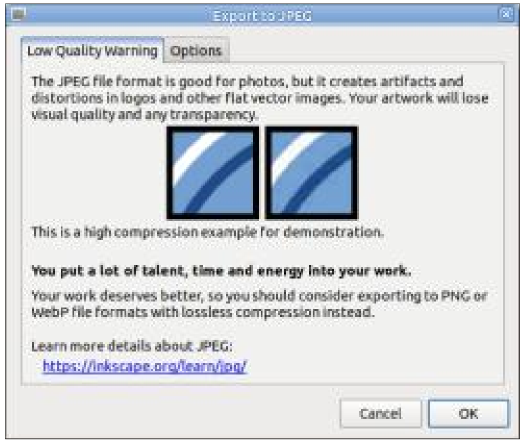
Clicking on the 'Page' button updates the UI a little, showing just a single page at a time in the preview. Just above that is a less-than-obvious set of UI elements: the page label, with back and forward buttons on either side. These buttons let you select which page to export, and the main canvas will zoom and pan to fill with the relevant page as you change



them. This can actually be rather annoying if you were quite happy with your previous view of the document, so I'll take this opportunity to remind you of the View > Zoom > Zoom Previous menu option, which is assigned to the backtick (`) key by default.



Switching to the Batch Export tab, we find a minor UI inconsistency in that this tab contains sub-tabs rather than buttons for switching between the different export modes. One of those sub-tabs is labelled 'Pages' and allows you to export each individual page as a separate file. The order in which the pages are presented can be a little odd, as shown in this screenshot. It doesn't seem to reflect the order of the pages on the canvas, their creation order, alphabetical order, nor the order they appear in the SVG file. If you're going to be performing batch exports then it makes sense to give your pages clear and sensible labels to help work around this confusion.



The JPEG file format is good for photos, but it creates artifacts and distortions in logos and other flat vector images. Your artwork will lose visual quality and any transparency.

This is a high compression example for demonstration.

You put a lot of talent, time and energy into your work.

Your work deserves better, so you should consider exporting to PNG or WebP file formats with lossless compression instead.

Learn more details about JPEG:
https://inkscape.org/learn/jpg/

One thing to note about the new Export dialog is that there's now a pop-up 'Format' menu. No longer are you limited to just exporting PNG images. You can also export JPEGs directly, or even to export individual objects, pages or layers as their own separate SVG files, which will greatly simplify some workflows. Just be conscious of the different pros and cons of each format: JPEGs don't support transparency, for example, so Inkscape will automatically add a background based on the page color (set via the File > Document Properties… dialog). In fact the Inkscape developers seem rather keen for you not to use JPEG, as demonstrated by the options dialog that opens when you start the export.

One final feature of the Batch Export mode is the presence of the 'Add Export' button, just below the controls for selecting the file suffix, format and DPI. Click this to create additional rows, allowing you to export to multiple formats, or multiple DPI settings at once. For anyone who needs to generate assets in a variety of formats or sizes this could be a huge time saver.

The new multi-page functionality is a huge step forward for Inkscape – especially combined with the improvements in batch exporting. But the limitation of only seeing the first page in web browsers is a frustrating and, quite frankly, unnecessary one. Next month I'll be looking at how multiple pages have been implemented in the SVG file, and demonstrating how it is actually possible to view the additional pages within a web browser, if you know what you're doing.

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: @PeppertopComics

contents ^

Written by Ronnie Tucker

FreeCAD - Part 4

So let's add a piece to our plate. Let's say we need to have our finished plate screwed to a wall and a PVC pipe is to be inserted into a socket of some sort.

Here's where we're at thus far (bottom left).

In the left panel it's listed as Body (the thing we're making) then the Pad which was done on the (indented text) of the Sketch. Finally we did the Chamfer (on the holes).

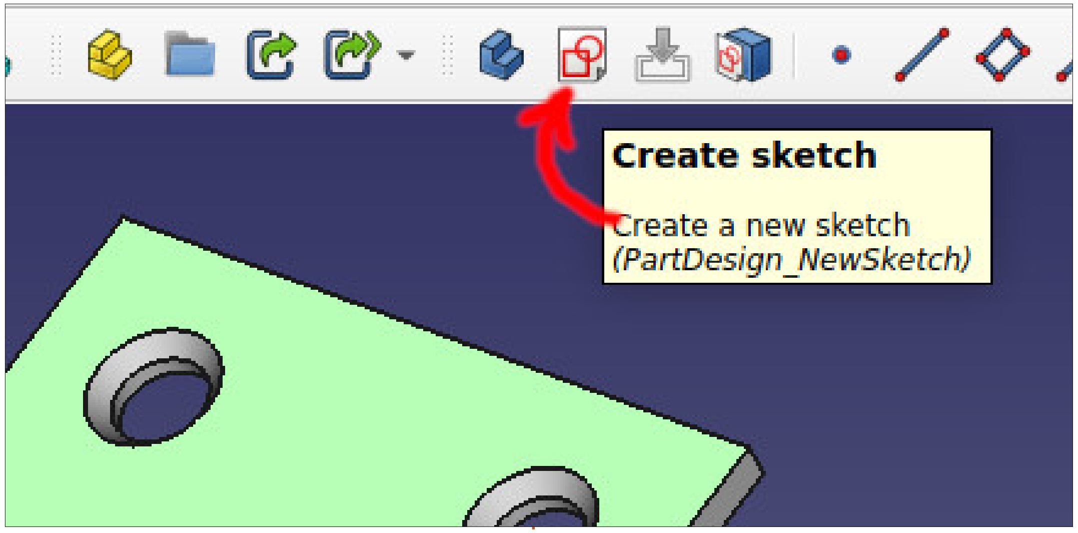
To add a pipe socket, we need to add the socket to the front face of the plate. Right? So left click to select the top face of the plate. It'll turn green. Next, click Sketch (see image top right).

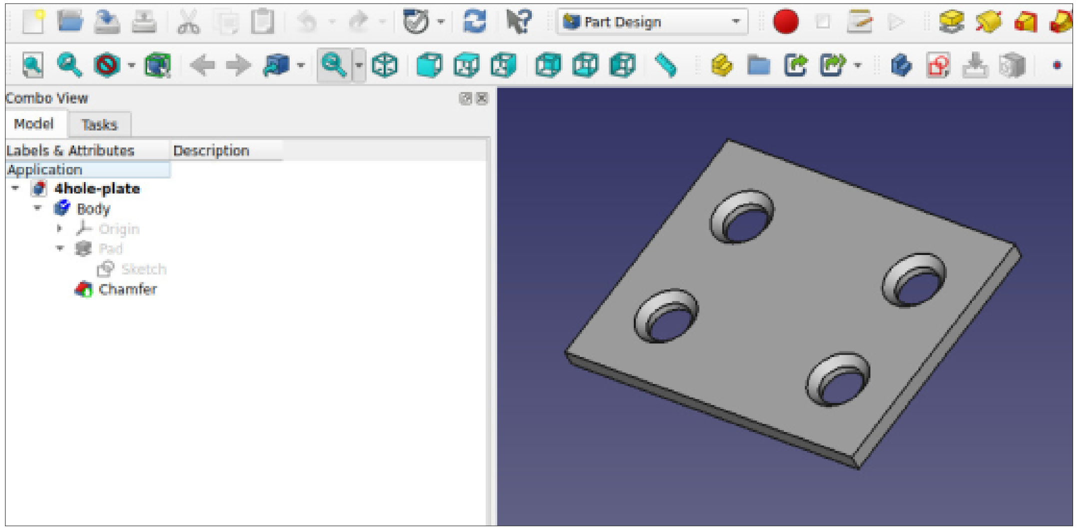
Well, this (bottom right) looks familiar.

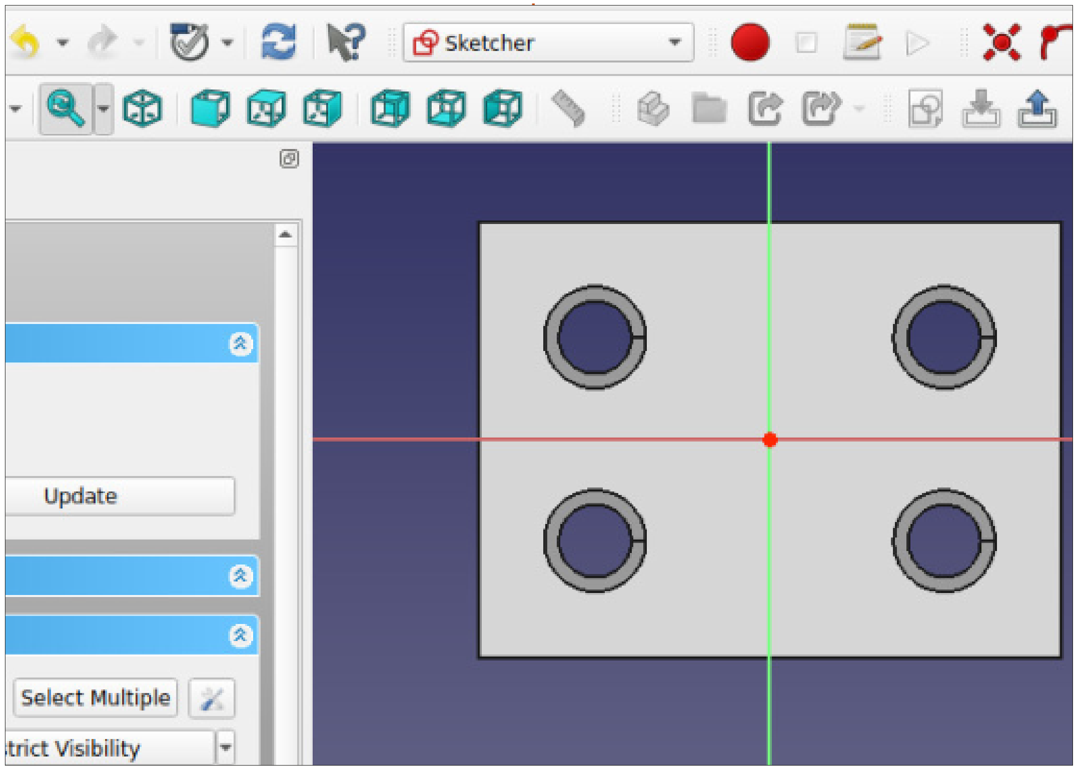
But instead of a blank sketch, we have our plate as the background. The red dot is the center point of the plate.

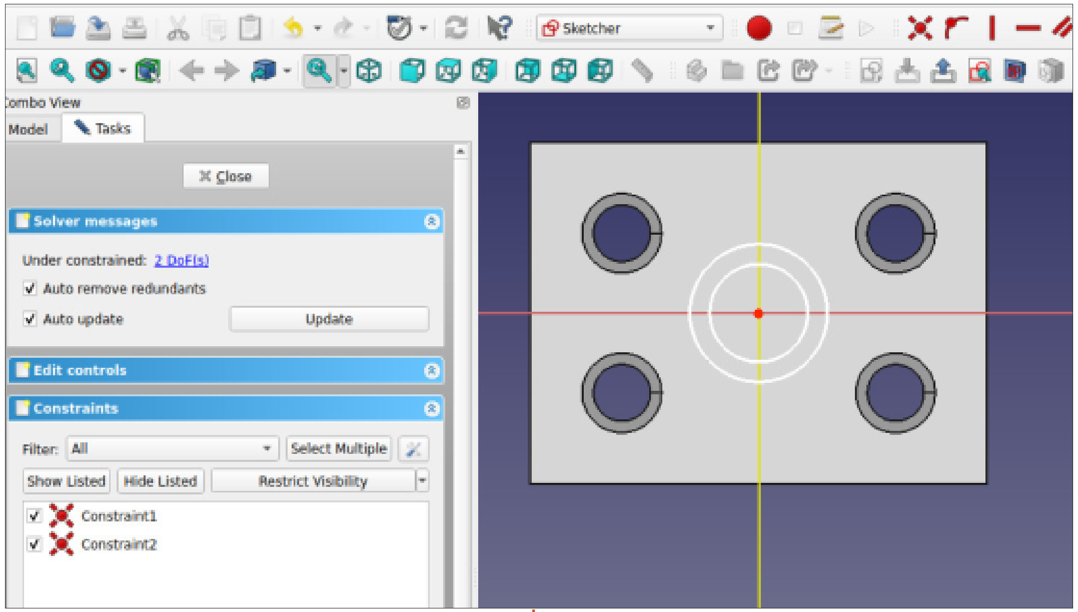
Use the circle tool to draw two circles going from the red center point out (see image, next page, top left).

Like in previous examples, lock the circles down with diameters or








Combo View

Model | Tasks

Close

Solver messages

Under constrained: 2 DoF(s)

✓ Auto remove redundants

✓ Auto update

Update

Edit controls

Constraints

Filter: All

Select Multiple

Show Listed | Hide Listed | Restrict Visibility

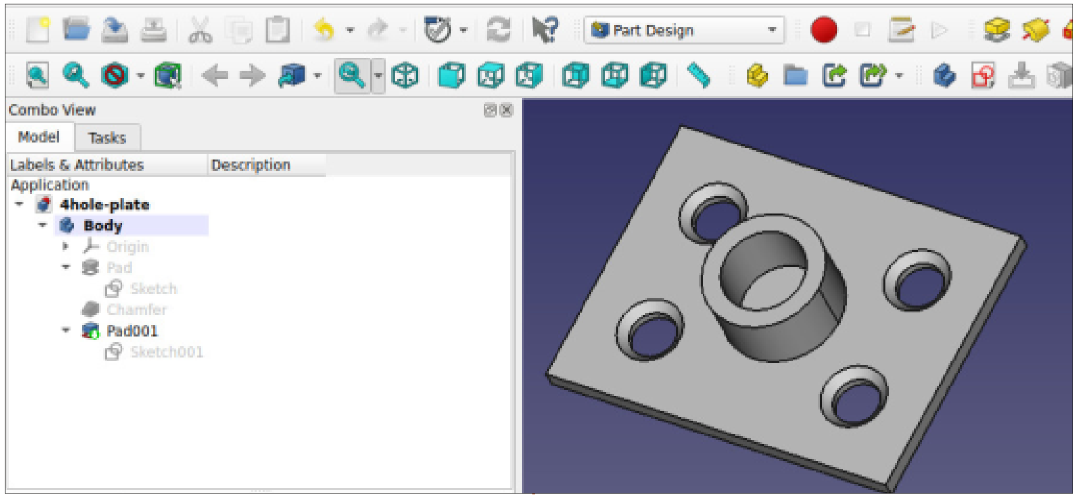
✓ Constraint1

✓ Constraint2

radius. Whichever you prefer.

I went with an inner diameter of 9mm and an outer of 13mm.

Like when we made the plate, click Close (in the left panel) to return to our Part Design workbench. And, like last time, click Pad. Voila! Adjust the height of the pad as you see fit (top right).


Combo View

Model | Tasks

Labels & Attributes | Description

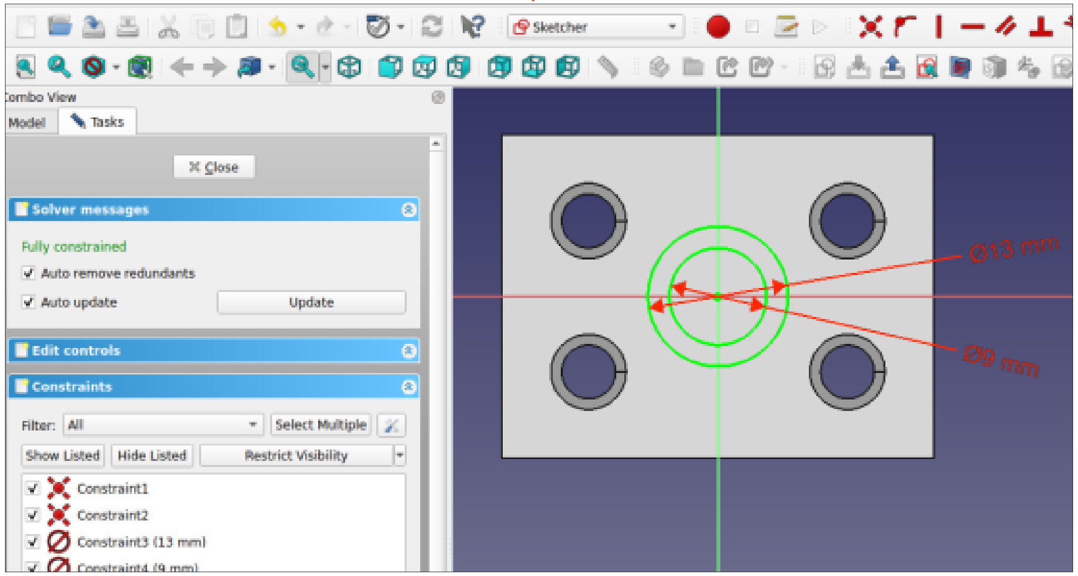
Application
- 4hole-plate
  - Body
    - Origin
    - Pad
      - Sketch
    - Chamfer
    - Pad001
      - Sketch001

Note, on the left we have (below Chamfer) a new Pad and new Sketch. They only apply to the pipe socket.

If you need to adjust the depth of the socket you can click on the Pad001 (in my case) and you'll see the attributes.

Feel free to click the edge where the socket meets the plate and choose to put a chamfer or fillet on it. Make it look spiffy.

Next time we'll abandon our plate and try something new…


Combo View

Model | Tasks

Close

Solver messages

Fully constrained

✓ Auto remove redundants

✓ Auto update

Update

Edit controls

Constraints

Filter: All

Select Multiple

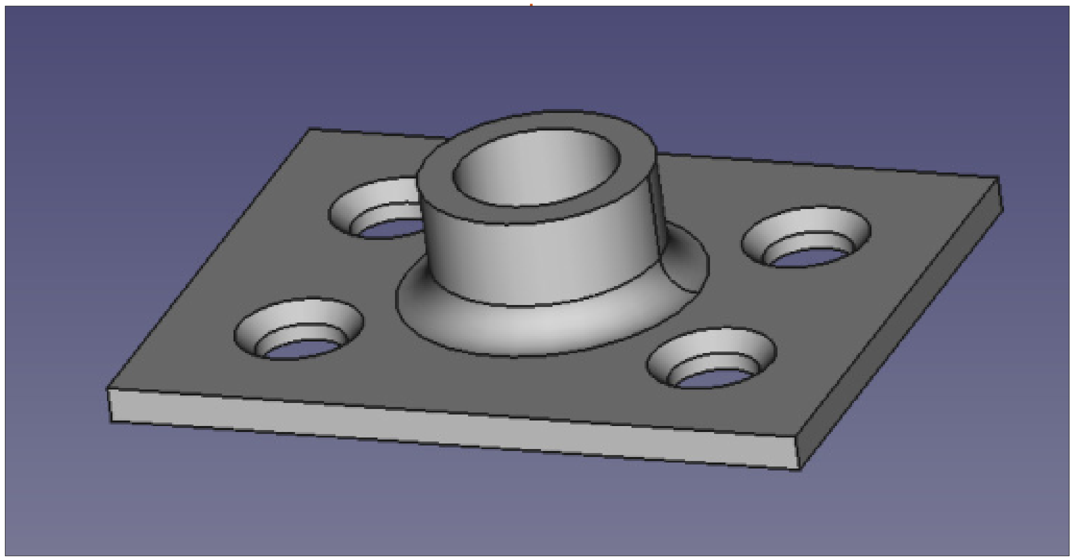
Show Listed | Hide Listed | Restrict Visibility

✓ Constraint1

✓ Constraint2

✓ Constraint3 (13 mm)

✓ Constraint4 (9 mm)



Written by Mark Crutch

Inkscape - Part 132

In the previous instalment, I looked at the new multi-page tool introduced in Inkscape 1.2. As I often delve into the guts of Inkscape's SVG files in this series, I thought it would be interesting to see exactly how multiple pages have been implemented. What I've found is a little disappointing, because I think the developers have made a slightly bad choice that limits the usefulness of multi-page mode more than was necessary. In this article, we'll take a look at exactly what that mistake was, and a couple of ways to work around it.

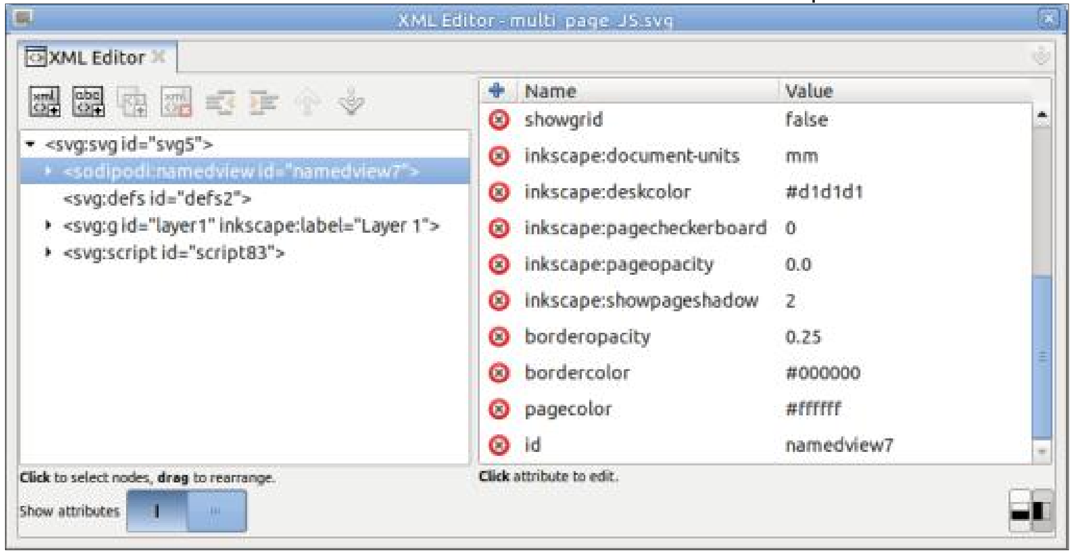
To investigate this, we'll be using a simple multi-page Inkscape file. Each page contains a colored rectangle that fills the whole page area (so it's easier to see the page boundary when loaded into a web browser), plus a random single object in the middle, as a placeholder for the real content you might put in your document.

The first thing to do is to load the SVG file directly into a web browser and see what appears. You don't win any prizes for guessing that only the first page is displayed. This is simply the same behaviour as a single-page Inkscape document. It could be argued that the main purpose of the multi-page tool is for importing and exporting PDFs, so it's hardly surprising that a web browser won't show the extra pages, but since SVG is a first-class format for the web, it does seem a shame that there's no easy way to view the other pages. But the key word there is "easy". If you're prepared to get a little technical, it is possible to work around this limitation.

First, let's take a look at what's going on inside the SVG file itself. You can simply open it in a text editor, but as this is an Inkscape column, I'm going to look at it via the Edit > XML Editor… option (below).

In the left pane, we can see the tree of XML elements that make up our file. At the top is the 'root' node, shown as <svg:svg …>, followed by a <sodipodi:namedview …>. If you're new to Inkscape, you may not realise it was forked from an older program named Sodipodi, the fingerprints of which are still present inside Inkscape's SVG files. In this case, we have a <namedview> element which is in the 'sodipodi' namespace. Namespaces are a means of mixing multiple different XML languages together in one file and ensuring they don't clash with each other. In the case of Inkscape, it means that anything in the 'sodipodi' or 'inkscape' namespaces are not part of the SVG standard, and will simply be ignored by most other programs.

As hinted by its Sodipodi heritage, the <namedview> element has been present in Inkscape files right from the start. It contains metadata about how the file should be displayed when loaded into Inkscape: this is where



```
▾ <svg:svg id="svg5">
  ▸ <sodipodi:namedview id="namedview7">
    <svg:defs id="defs2">
  ▸ <svg:g id="layer1" inkscape:label="Layer 1">
  ▸ <svg:script id="script83">
```

Name	Value
showgrid	false
inkscape:document-units	mm
inkscape:deskcolor	#d1d1d1
inkscape:pagecheckerboard	0
inkscape:pageopacity	0.0
inkscape:showpageshadow	2
borderopacity	0.25
bordercolor	#000000
pagecolor	#ffffff
id	namedview7

Click to select nodes, **drag** to rearrange.

Click attribute to edit.

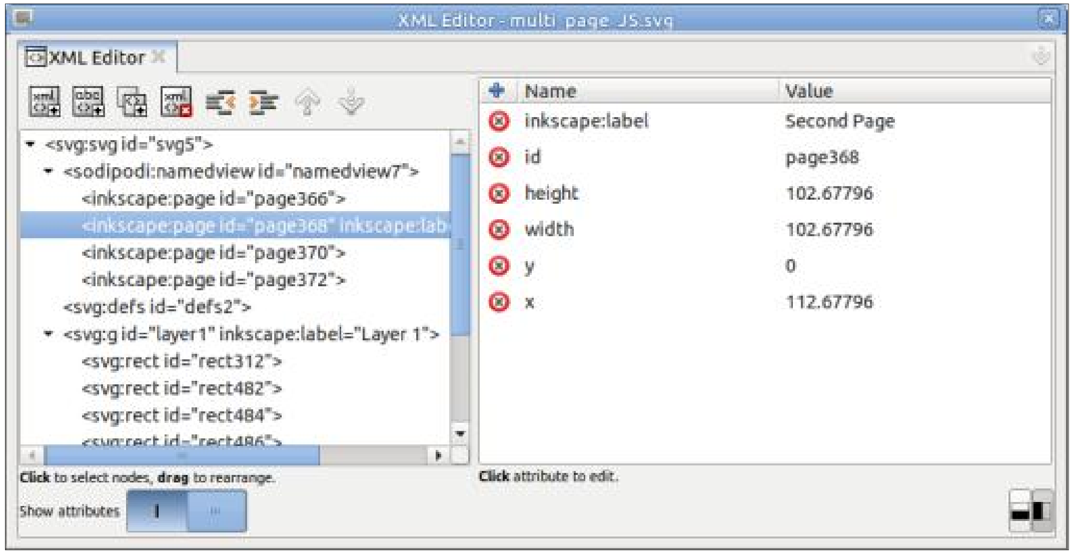
Show attributes

contents ^

you'll find XML attributes that store the window size, zoom factor, and page color, amongst other things. But with a multi-page Inkscape document the <namedview> element has gained something else: children. Expanding the <namedview> entry in the XML editor reveals an <inkscape:page … > element for each page in your document. To clarify, these are <page> elements in the 'inkscape' namespace, so other applications – including your web browser – have no idea what to do with them.

In this screenshot, I've selected the second page in the document. You can see the attributes of the <page> element in the right-hand pane. These consist of the width and height of the page, and the coordinates of the top-left corner of the page (x and y). There's also an auto-generated ID, and an 'inkscape:label' attribute which holds any custom name you may have given to the page. If you're familiar with the internals of SVG files, then you may recognise the four dimension-related attributes as being the same as those used in an SVG viewBox definition. That fact gives us our first workaround for viewing pages in a browser.

When loading an SVG file into a web browser, there's a little-known trick that can be used to override the default viewBox definition from the main <svg> element. I covered this previously in part 79 of this series (FCM issue #139), but the summary is that you can use the


XML Editor - multi_page_JS.svg

XML Editor

```
▾ <svg:svg id="svg5">
  ▾ <sodipodi:namedview id="namedview7">
      <inkscape:page id="page366">
      <inkscape:page id="page368" inkscape:lab
      <inkscape:page id="page370">
      <inkscape:page id="page372">
    <svg:defs id="defs2">
  ▾ <svg:g id="layer1" inkscape:label="Layer 1">
      <svg:rect id="rect312">
      <svg:rect id="rect482">
      <svg:rect id="rect484">
      <svg:rect id="rect486">
```

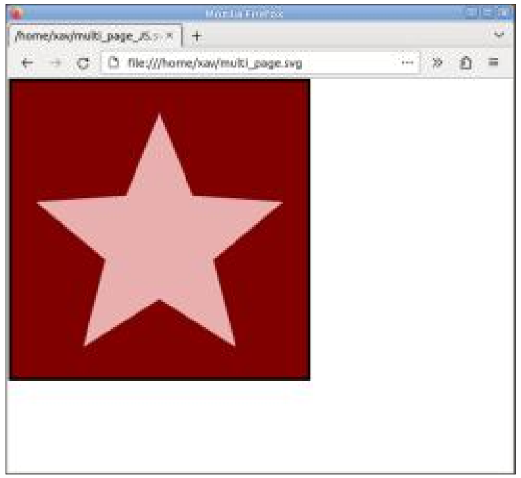
Name	Value
inkscape:label	Second Page
id	page368
height	102.67796
width	102.67796
y	0
x	112.67796

Click to select nodes, **drag** to rearrange.

Click attribute to edit.

Show attributes

'fragment identifier' of the file's URL to specify the x, y, width and height values of the viewBox you wish to use. Let's look at this with an example: first, we'll load the SVG file directly into Firefox and, as expected, we see only the first page.

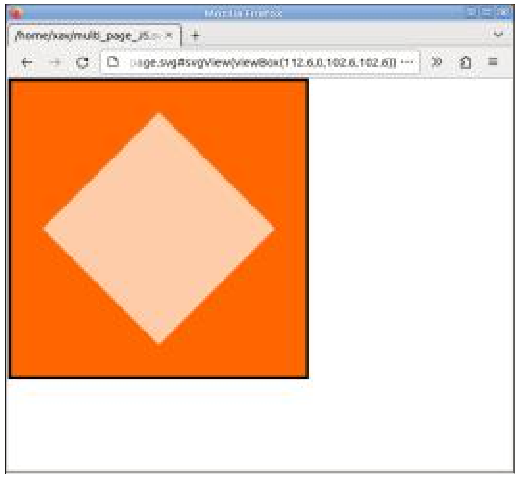


To access other pages, we first need to find the coordinates (x, y) and dimensions (width, height) from the relevant <inkscape:page> element. You can see from the screenshot of the XML editor that Inkscape stores these to quite a high precision but, in practice, you can usually truncate them to just a couple of decimal places – perhaps even less if there's a little free space between the edge of the page and the content. You then need to take these values, swap them into the following string, and append the whole thing to the URL in your web browser:

```
#svgView(viewBox(x,y,width,he
ight))
```

In the case of this example, the filename is 'multi_page.svg', and the values for the second page are shown in the earlier screenshot of the XML editor. The URL for viewing the second page therefore becomes:

```
.../
multi_page.svg#svgView(viewBo
x(112.6,0,102.6,102.6))
```


Mozilla Firefox
/home/xav/multi_page_JS.s
age.svg#svgView(viewBox(112.6,0,102.6,102.6))

This syntax works wherever the browser expects an image URL, so web developers can also use it in <img> elements, and even in CSS url() values.

It's a clever trick, which gives you access to all the pages in a multi-page Inkscape file, but it's not without its limitations. The biggest of these, quite clearly, is the need to dig into the SVG file to find the page size and position, then copy those values into the URL directly. The resultant link also lacks semantics – that specific combination of numbers doesn't exactly scream 'page 2' to the uninitiated. However, SVG also allows us to add 'named views' to the file (not the same thing as the <sodipodi:namedView> element), which still requires digging out the magic numbers, but does at least allow us to map each viewBox to a more meaningful name. The downside of this approach is that we will need to edit the SVG file, either with a text editor or using Inkscape's XML editor.

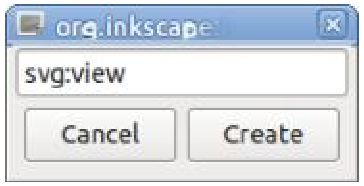
If you're comfortable editing XML files in a text editor, that's probably the easiest method. Just make sure you don't have the file open in Inkscape at the same time, or you may find your hand-crafted edits are automatically replaced. That approach is also described in part 79 of this series, so this time I'm instead going to show you how to make the same edits within Inkscape.

Once again, you'll need the x, y, width and height parameters from the relevant <inkscape:page> element. This time, I'll use the values for page 3 in the file. Again, these were obtained by selecting the relevant <inkscape:page> element in the XML editor, and looking at the attributes in the second pane. Once again, we'll truncate these to 1 decimal place, giving this set of values:
x: 225.3
y: 0
width: 102.6
height: 102.6

Observant readers may have noticed that only the x value has changed compared with the values for page 2. This makes sense, as the third page is the same size as the others, and has only been displaced horizontally. If your multi-page documents all use the same page size, with the pages aligned in a single row, it will always be the case that only the x value differs between them.

Now we need to create a new element in the document, which we can do via the XML editor. This can actually go anywhere in the document, and I've previously advocated putting it in the <defs> section – but now I've changed my mind. To keep similar items grouped together, I'm actually going to create it as a child of the appropriate <inkscape:page> element. To achieve this, first select the <inkscape:page> element in the left-hand pane of the XML editor. While it's highlighted, you might want to note down those all-important dimensions that are shown in the second pane (which could actually be below rather than to the right, depending on the orientation buttons in the bottom-right of the dialog).

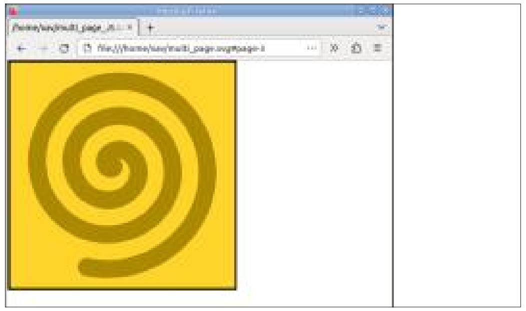
With the existing element selected, click the first button in the dialog's toolbar: "New element node". This will open a much smaller dialog with a single text field and a couple of buttons, where you should enter the name (including the namespace reference) of the new element you wish to create. In this case the string to type is "svg:view" – with no spaces, and without the quotes.

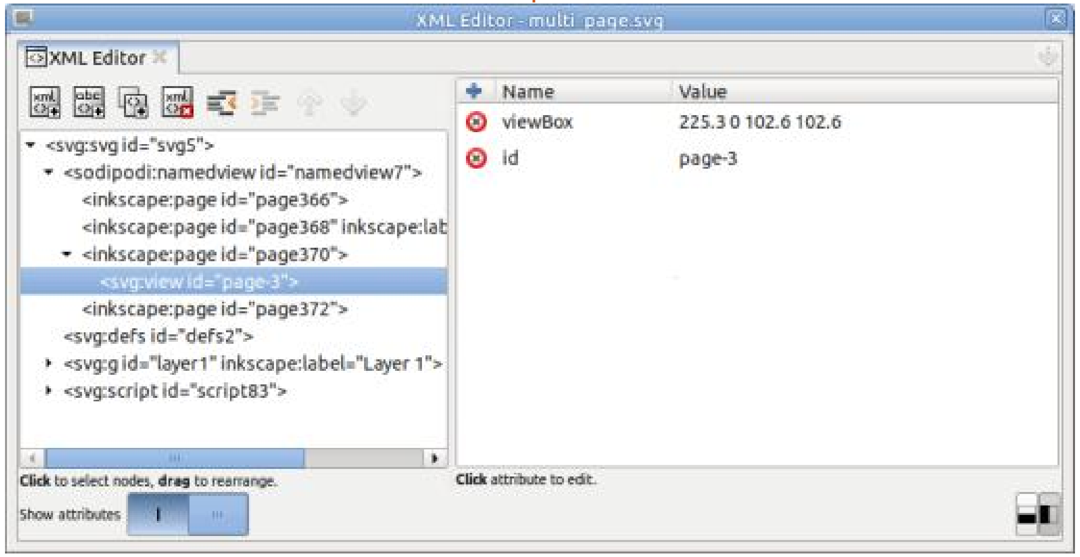


Click the 'Create' button, and you should find your new element appears in the first pane as a child of the <inkscape:page>. It should already be selected, with no attributes present in the second pane. We will need to add two attributes to this element. Let's start with the 'id', which you can create by clicking on the '+' button at the top of the second pane, and entering 'id' into the name field that appears in the list below. When you press enter, the value field will be focused, and it's here that you should put the more semantic name you wish to use for the page. Note that this is an XML ID, which means it can't contain any whitespace characters – so no 'Page 3' for example. In practice it's best to stick to alphanumeric characters, underscores and hyphens. The first character should be a letter, and I tend to stick to lowercase characters. In practice, therefore, something like 'page-3' is ideal.

Now repeat the process to add a second attribute, this time with a name of 'viewBox' (watch the capitalization), and a value of your four values, separated by space characters, in the order 'x y width height'. If all has gone well, the XML editor should look something like that shown below.

Repeat the process for each page, using the appropriate values and different IDs before saving your file to commit the changes to disk. Then, to view a page in the browser, you simply have to use the base URL for your SVG file, and append a hash (#) followed by the ID of the page. In this example, therefore, the URL becomes '.../multi_page.svg#page-3':




XML Editor - multi_page.svg

XML Editor

```
▾ <svg:svg id="svg5">
  ▾ <sodipodi:namedview id="namedview7">
      <inkscape:page id="page366">
      <inkscape:page id="page368" inkscape:lab
    ▾ <inkscape:page id="page370">
        <svg:view id="page-3">
      <inkscape:page id="page372">
  <svg:defs id="defs2">
  ▸ <svg:g id="layer1" inkscape:label="Layer 1">
  ▸ <svg:script id="script83">
```

Name	Value
viewBox	225.3 0 102.6 102.6
id	page-3

Click to select nodes, **drag** to rearrange.

Click attribute to edit.

Show attributes

Once again, this trick works with any place that the browser expects a URL, including <img> tags and CSS. In practice, I suggest picking better semantic IDs than simply 'page-n' if you possibly can. Suppose we had used 'yellow-spiral' as my ID in this case: if we subsequently wanted to rearrange the pages within the SVG document we would only have to update the coordinates in the <svg:view> elements, but any website that referred to the page by that ID would still work without modification.

Perhaps you can now see why I think the Inkscape developers made a mistake when implementing multi-page support. The <inkscape:page> element has attributes for an ID, an optional label (in the 'inkscape' namespace), and the four dimension and position values. The SVG standard <view> element, on the other hand, has an ID and a viewBox (which consists of the four dimension and position values). It's lacking a label attribute, but this could legitimately be added within the 'inkscape' namespace, without breaking any XML or SVG rules. I fail to see, therefore, why the Inkscape developers chose to use a non-standard <page> element in their own namespace, rather than just use the existing SVG <view> element, adding custom attributes where required. Had they done so, every Inkscape page would automatically get a named view, and therefore be much easier to access via a web browser.

Sadly, this missed opportunity leaves users either having to extract the details of each page's viewbox in order to construct a suitable #svgView(...) fragment identifier, or manually creating their own <view> elements to achieve the same effect but with nicer URLs. Next month we'll look at a different approach to 'fixing' this problem – though one that unfortunately comes with its own set of limitations.

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: @PeppertopComics

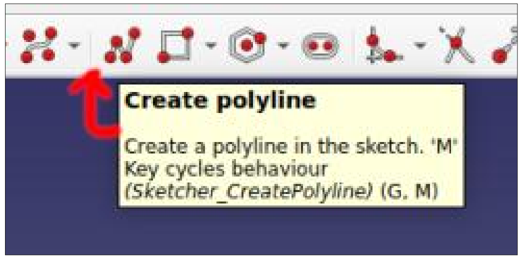
Written by Ronnie Tucker

FreeCAD - Part 5

Let's start anew. Click File > New and then create a new body. If you've just joined us and wonder how, then see part 1 of this series.

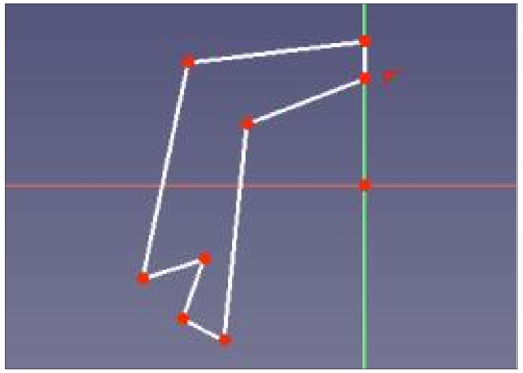
Create a new sketch on the YZ (right) plane.

What we want to do here is create a shape on one side of the vertical line. Do this using the Polyline tool


Create polyline

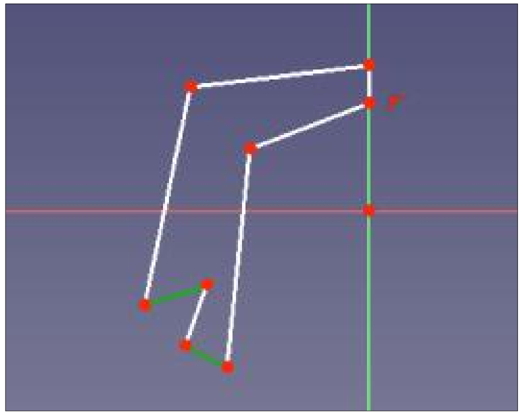
Create a polyline in the sketch. 'M' Key cycles behaviour *(Sketcher_CreatePolyline)* (G, M)

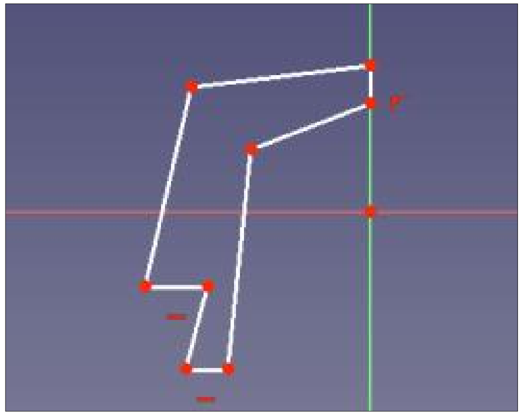
Click somewhere on the vertical line to anchor our shape there. Click, click, click, until you get a shape something like this:



It doesn't have to be exact. Just something similar. But make sure you finish where you started. We want this to be a closed shape.

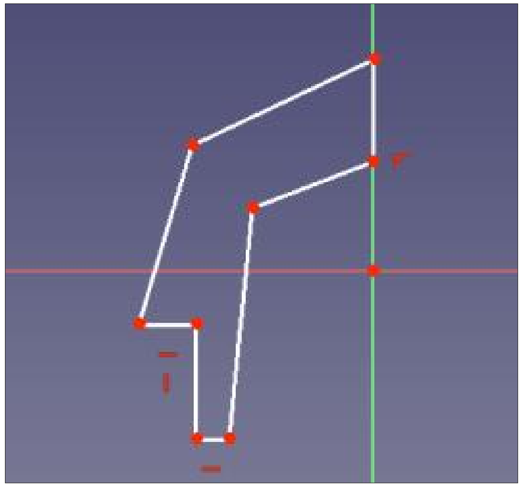
Now, my shape is purposely wonky so that I can show you a couple of new tools before we finish off this object.





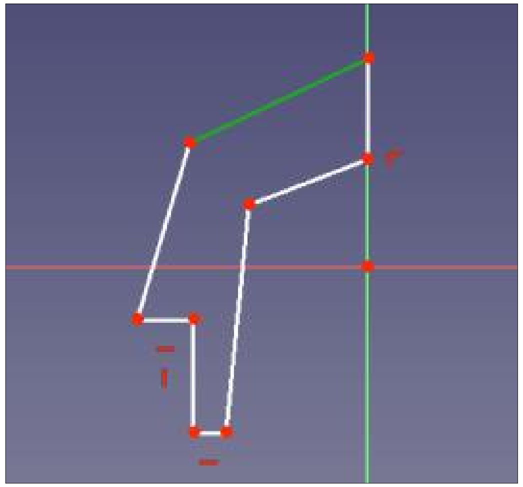
See those two lines in green? Ideally I'd like them to be exactly horizontal. Well, all I need to do is to (one at a time) select the line, then click the Horizontal constraint icon (that we used a couple of parts ago).

See how those two lines now have a red minus sign below them to show they're constrained to be horizontal. I set the line connecting them to vertical. You can still move those points around, but you'll see how they're locked to move only horizontally/vertically.

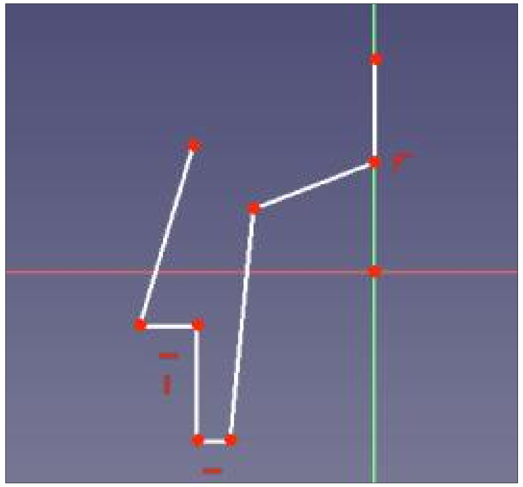


I've moved a couple of points to suit what we're about to do next.

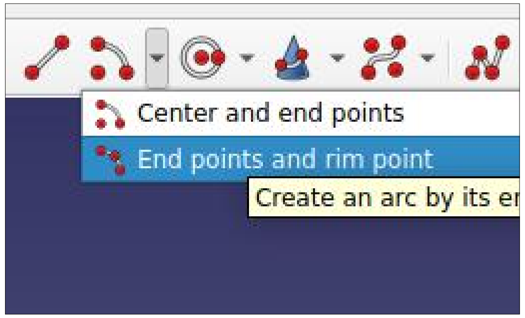
Select the top angled line.



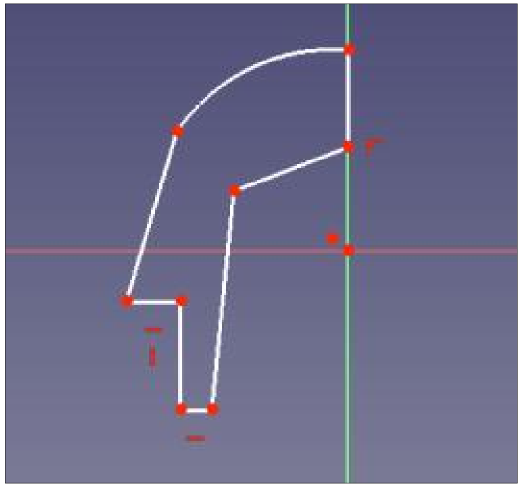
And press delete.



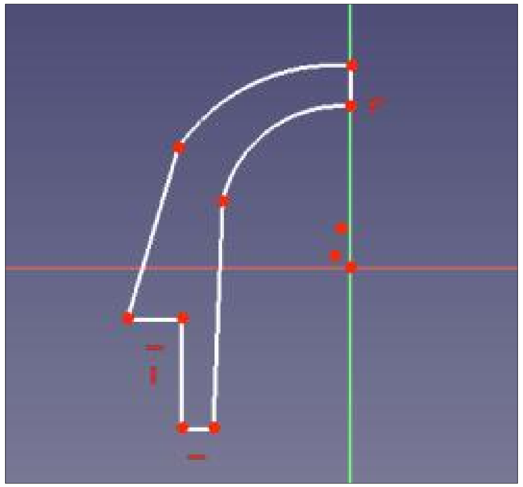
Let's use another new tool. The arc shape. You may need to click the down-arrow beside the arc icon to see the End points and rim point option.



Click on one of the open points. You'll see a circle appear. Click the other point and you'll get an arc that you can adjust. Give our shape a nice arc.



Feel free to do the same to the other diagonal line.

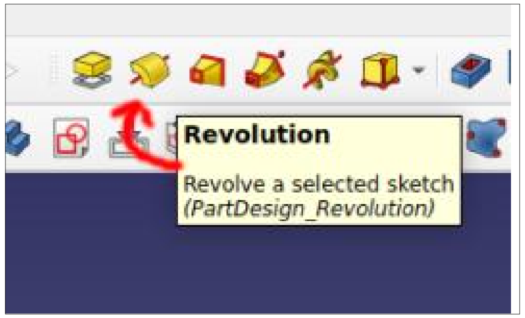


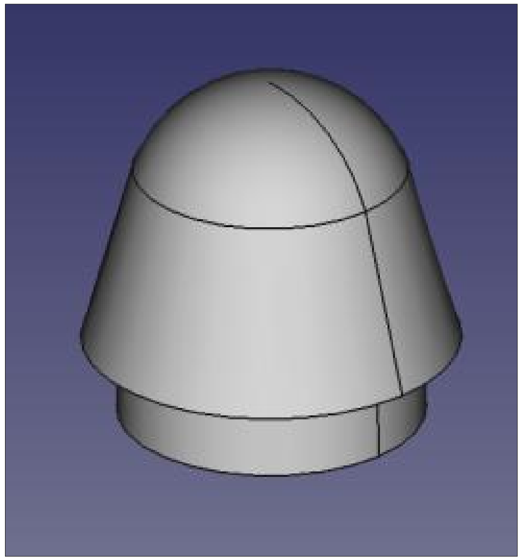
After I did my second arc, I moved one of my vertical points up a bit just to make it look better.

You could, of course, lock down all of those points but, for now, this will do us.

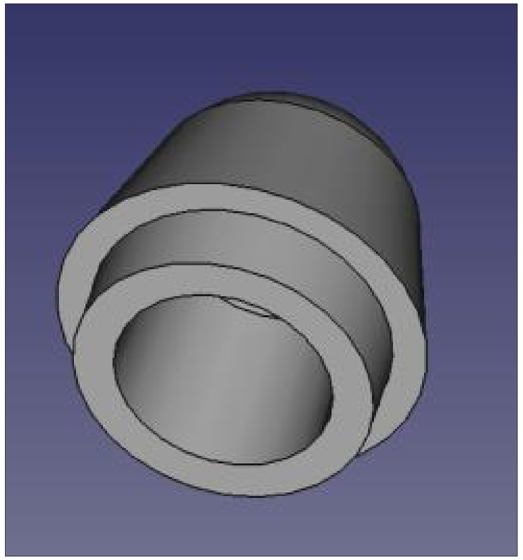
In the left panel, click Close to return back to the Part Design workbench.

Time to do a new thing. Click the Revolution button.

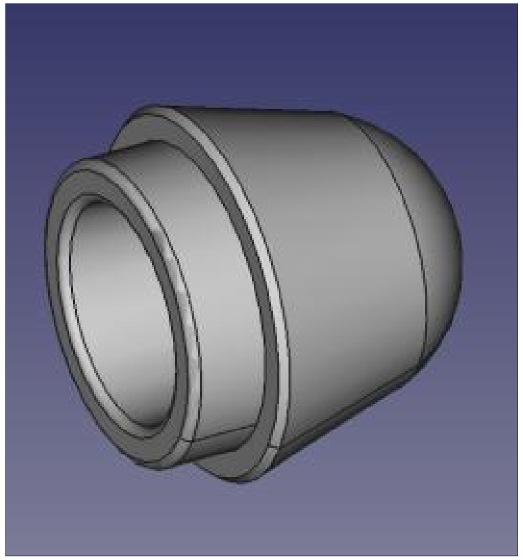




The line you see down the side is where the shape has rotated 360 degrees.



So, as you can imagine, this is an incredibly powerful tool. But what if you need to make something that's not a perfect circle? We'll look into that next time.



As ever, feel free to click edges and make it look pretty.

Ronnie is the founder of Full Circle and, somehow, still editing this thing. He also paints, draws and does woodcarving in his spare time.

contents ^

Written by Mark Crutch

Inkscape - Part 134

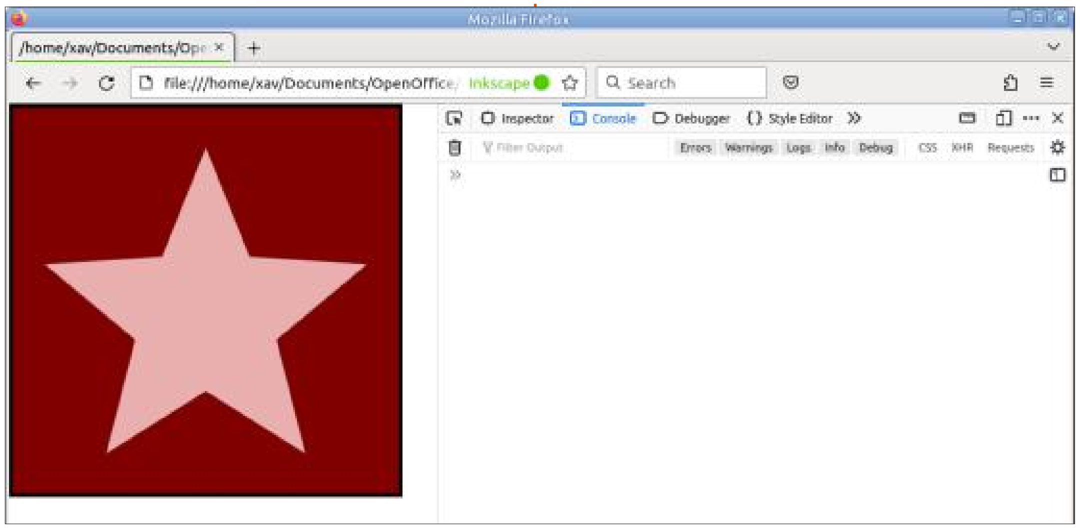
Over the previous few instalments, I've been looking at Inkscape's new multi-page feature, culminating in last month's analysis of how it's implemented within the SVG file. What I found was that the details of each page are stored in a proprietary 'page' element in the Inkscape namespace – appearing in the XML editor as <inkscape:page> – rather than in a standard SVG <view> element. This is a strange choice, in my opinion, as using the latter would have made the extra pages more accessible via a web browser.

Last month, I described a couple of methods by which a browser can be coaxed into showing the extra pages. The first used a little-known version of the URL fragment identifier to target each page by its viewBox values. This doesn't require the file to be modified, but does need you to dig into the XML to extract the relevant numbers. The second approach modifies the file to manually add <view> elements, which is more work but does allow you to use a more semantic URL. If you haven't read the previous article, I strongly advise doing so before proceeding, as this one directly builds on that information.

Our aim here is to make it easier to access all the pages of an Inkscape document via a web browser. One thing browsers do, which Inkscape doesn't, is to run JavaScript. So, by adding a small JS function that will be executed when the file is opened in a browser, we can automatically create <view> elements from each of the <inkscape:page> elements, without the need to manually copy-and-paste coordinates. This is obviously useful when there are a lot of pages to consider, but even for files with just a couple of pages, the ability to just paste the same snippet of code into any file without having to modify it each time makes this approach arguably simpler than manually creating the <view> elements yourself.

We'll be using the same multi-page SVG file from last month, consisting of four pages arranged horizontally, with a different background color and content in each.

Having created a multi-page Inkscape file, the first thing to do is to save it to disk and open it inside a web browser. This will display only the first page, as expected. As we progress through this code, we'll log some data to the developer console, so open the dev tools in the browser (typically by pressing F12), and switch to the 'Console' tab. Below is how it looks in Firefox on my Ubuntu Mate machine.



I've talked about adding JavaScript code to Inkscape files in the past, so I won't go into depth about the options here. For what we're trying to achieve, we just want a simple chunk of script embedded directly into the file

which will run at load time. The easiest way to create this within Inkscape is as follows:
- Open the File > Document Properties… dialog
- Switch to the 'Scripting' tab.
- Within that tab, choose the 'Embedded scripts' tab.
- Click the '+' button below the (empty) list of embedded scripts.
- You should see a new entry appear in the list with an arbitrary ID.
- Ensure that entry is selected.
- Write your code in the 'Content' box below.
- You can save (Ctrl-S) as you develop, without having to close the dialog.
- Whenever you save the file, manually reload in the browser (F5) to see the effect.

Let's begin by logging out the existing <inkscape:page> elements to the console using this code (top right).

```
const inkNS = 'http://www.inkscape.org/namespaces/inkscape';
const svgNS = 'http://www.w3.org/2000/svg';
const pages = document.getElementsByTagNameNS(inkNS, 'page');

Array.from(pages).forEach((page, idx) => {
    console.log(idx, page);
});
```

It's not essential to fully understand the workings of this code in order to use it, but I'll describe it anyway for those who are interested. The first two lines simply set up variables for the namespaces we'll need. We won't be using the SVG namespace just yet, but we will require it soon, so this is a good time to introduce it. The third line uses one of the browser's built-in functions to find all the 'page' elements in the Inkscape namespace, and assign them to a variable as a 'collection'.

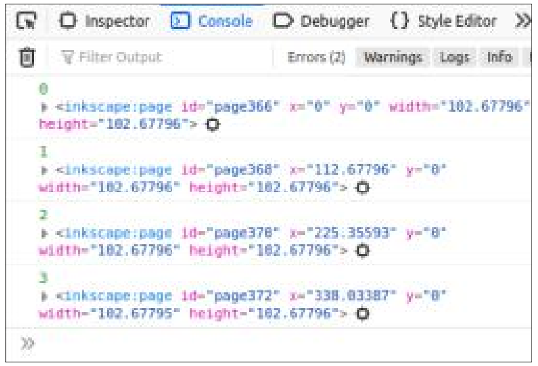
For historical and technical reasons, a 'collection' is very similar to a JS array, but not quite actually the same thing. These days, JavaScript's array functions are pretty powerful, so we would really like to create an array from the content of our collection so we can use those functions. The 'array.from(pages)' part does exactly that, iterating over each entry in the collection to build up a temporary array. We can then use the 'forEach' array method to execute a block of code for each entry in the array.

The forEach() method expects to have one parameter, and that parameter should be a function. We could build a function elsewhere and pass its name in here, but it's more common in JS to see 'anonymous' functions used for small tasks like this one. In this case, the anonymous function is called once for each element in the array, and, each time it's called, it's passed the current element (in a variable we've called 'page') and the index of that element in the array ('idx'). The '=>' syntax is used for so-called arrow functions, and can be thought of largely as an alternative to the 'function' keyword you might be more familiar with if you haven't gone near JS for a while.

Inside curly braces ('{…}') we have the body of the function – just a single line that uses the console.log() function to print the idx and pages values to the developer console. Finally, the last line closes not only the function body, but also the end of the forEach() method.

If you're not very familiar with JavaScript, then do take a few minutes to try to understand the code above. It's particularly useful to examine where each pair of brackets ('(…)') and braces ('{…}') start and end, and what content is inside each one.

With that code in place, save the file and reload it in the web browser, and you should see something in the console looking similar to this:



```
0
▸ <inkscape:page id="page366" x="0" y="0" width="102.67796"
height="102.67796">
1
▸ <inkscape:page id="page368" x="112.67796" y="0"
width="102.67796" height="102.67796">
2
▸ <inkscape:page id="page370" x="225.35593" y="0"
width="102.67796" height="102.67796">
3
▸ <inkscape:page id="page372" x="338.03387" y="0"
width="102.67795" height="102.67796">
```

The green numbers are the idx values, starting at zero because…

well, there are good reasons, but this is not the place to go into them. Suffice to say that most programming languages use zero-based indexing for things like arrays, and JavaScript is no exception.

Following each green number, you can see an XML representation of each SVG element in the array – the four <inkscape:page> nodes we're interested in. Each node also has a whole load of other baggage attached to it in the JS world, and you can see much of that by expanding the small triangle next to each one. In practice, we don't need any of that for our task, so feel free to leave that triangle unexpanded, or to collapse it back down again if curiosity does get the better of you.

So far, our code hasn't really achieved very much – just printing the index, and the same nodes we can see in Inkscape's XML editor. But now that we've got a way to grab a handle to each <page> element, we can start to pull them apart to get to the individual details we'll need. We're going to want to extract the x, y, width, and height values. These are stored as 'attributes' on the element, and can be retrieved using the getAttribute() method. Let's add four lines after the console.log() to retrieve these values, and assign each to a JS variable (x, y, w, h).

```
const x =
page.getAttribute('x');

const y =
page.getAttribute('y');

const w =
page.getAttribute('width');

const h =
page.getAttribute('height');
```

We could log them out at this point, but ultimately we're going to want these formatted into a space-separated string for use in a viewBox attribute. One additional line of code will do this for us, and then we'll log the result:

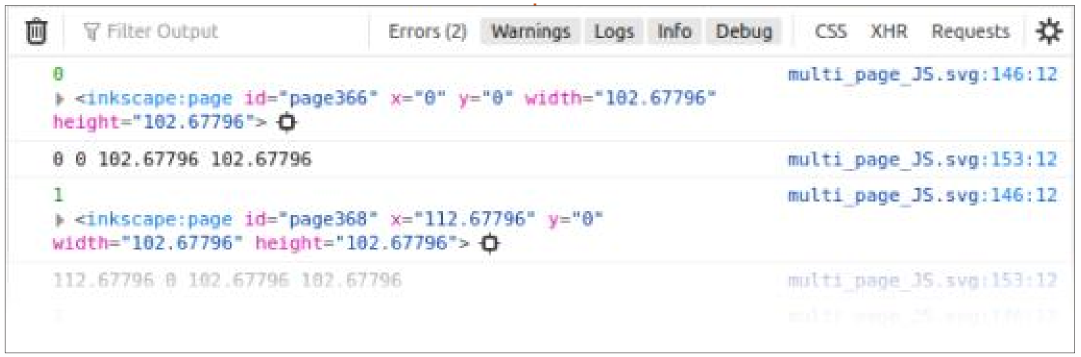
```
const viewBox = `${x} ${y} $
{w} ${h}`;

console.log(viewBox);
```

Be very careful! The 'const viewBox' line uses a feature of JS called 'template strings'. These are delimited by backticks (`) rather than normal quotes or apostrophes. Using this method lets us put our variables directly into the string using the ${} notation, and the JS engine will swap them out for the variables' values when the code runs. If your log ends up containing the actual ${} string, then you've used the wrong type of quotes, and will need to search your keyboard again for the easily-overlooked backtick character.

All those previous 6 lines should have been added after the existing console.log() and before the closing brace. If you've done it correctly, saving the file and reloading it in the browser should show something like this – similar to the previous output, but with the viewBox values displayed after each XML node (see image below).

Now we've got all the details we need, it's time to create a new <view> element for each page. For this, we need to use the document.createElementNS() function – the NS on the end referring to the fact that this lets us specify a namespace for our new element. This is where our earlier assignment of the svgNS variable will be used, ensuring that we end up with what is effectively an <svg:view> element, rather than an <inkscape:view> element, or anything else.

Each <view> element will also require two attributes. One is the 'viewBox' for which we've already prepared the value. The other is an 'id' attribute which will define the string we have to append to our URL to view this page. For the sake of simplicity, we're just going to name the pages 'page-1', 'page-2', and so on, using another JS template string in which we'll also add 1 to the value to rid ourselves of those pesky zero-indexed numbers. Therefore, to create our new element, and set both attributes, we'll need these three lines of code:



```
0                                                              multi_page_JS.svg:146:12
▸ <inkscape:page id="page366" x="0" y="0" width="102.67796"
height="102.67796">
0 0 102.67796 102.67796                                        multi_page_JS.svg:153:12
1                                                              multi_page_JS.svg:146:12
▸ <inkscape:page id="page368" x="112.67796" y="0"
width="102.67796" height="102.67796">
112.67796 0 102.67796 102.67796                                multi_page_JS.svg:153:12
```

```
const view =
document.createElementNS(svgN
S, 'view');

view.setAttribute('id',
`page-${idx + 1}`);

view.setAttribute('viewBox',
viewBox);
```

There's just one thing left to do. Although we've created our new <view> element, it currently just lives as an object in the JS world, and needs to be inserted into the browser's internal model of the document. We'll insert each <view> as a child of the corresponding <inkscape:page> element in order to keep things neatly together. This last line will do the job:

```
page.appendChild(view);
```

With that, the final code should look like this – albeit that you can't actually see all the lines at once in Inkscape's unfortunately inflexible editor field (top right).

You can optionally remove the console.log() lines if you wish, as they're purely there for educational and debugging purposes, and have no effect on the actual operation of the code.

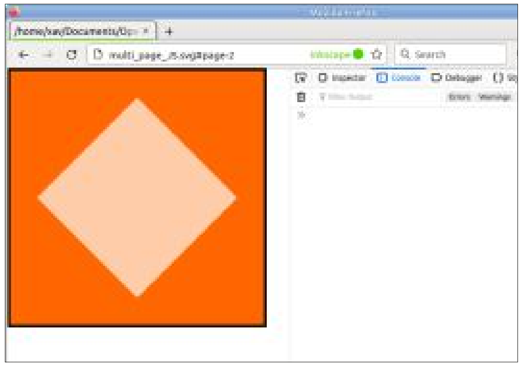
Now that the code is done, how do you actually use it? Simply load the SVG file directly into your browser, and append '#page-2' to the end of the URL to view the second page. I'm sure you can work out the syntax for the other pages. Entering an invalid ID (e.g. '#page-22') simply causes the browser to show the first page.

There we have it: a small chunk of JS that you can add to any multi-page Inkscape file to make the additional pages available via a web browser. What more could you possibly want?

Quite a bit as it happens. This code is good, but suffers from a significant limitation: in order for the browser to execute it, the SVG file has to be loaded directly (or within an <object> tag). Most common ways of including SVG files in a web page – via an <img> tag or a CSS url() function – are deliberately prevented from executing JavaScript. This significantly limits the usefulness of this code – at least as it stands.

Another issue is that we've just given each page a rather generic ID. Within Inkscape, it's possible to name each page – wouldn't it be nicer if we could use those names when referring to each page, rather than just page-1 and page-2?

```
const inkNS = 'http://www.inkscape.org/namespaces/
inkscape';
const svgNS = 'http://www.w3.org/2000/svg';
const pages = document.getElementsByTagNameNS(inkNS,
'page');


Array.from(pages).forEach((page, idx) => {
   console.log(idx, page);

   const x = page.getAttribute('x');
   const y = page.getAttribute('y');
   const w = page.getAttribute('width');
   const h = page.getAttribute('height');
   const viewBox = `${x} ${y} ${w} ${h}`;
   console.log(viewBox);

   const view = document.createElementNS(svgNS, 'view');
   view.setAttribute('id', `page-${idx + 1}`);
   view.setAttribute('viewBox', viewBox);
   page.appendChild(view);

});
```



Next month, I'll extend this code a little further to address both these issues. It still won't give the simple, seamless experience that we could have had if Inkscape natively created named views, but it's better than being stuck with multi-page files that can display only the first page!

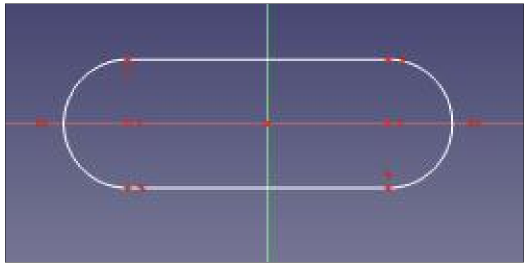
Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

Written by Ronnie Tucker

FreeCAD - Part 6

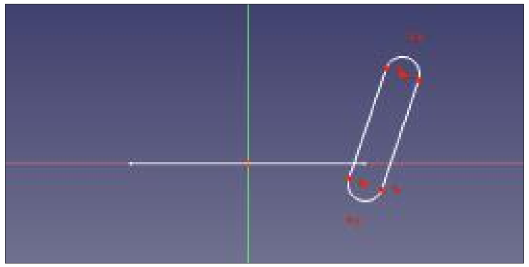
So, let's start a new project. It is, of course, File > New, then create a new body, and a new sketch, in the XY (top-down) plane.

This is where we want to make the path that our shape will follow.



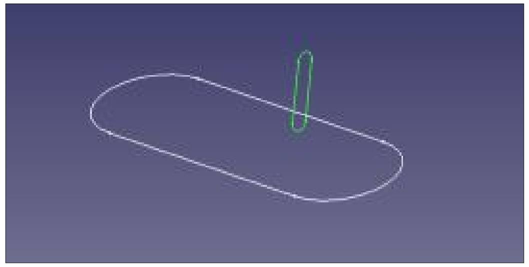
For this I used the Create Slot tool.

Now close that sketch and create a new sketch in the YZ (right) plane. That white line you see is our last sketch, the path. So I used the slot tool again to draw a quick shape.



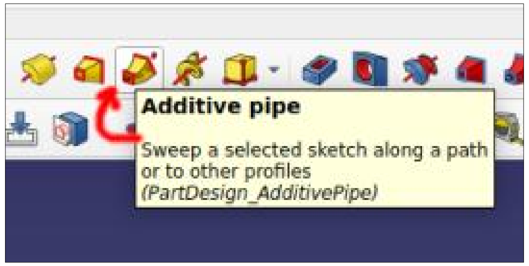
Now for the magic.

Close that sketch.



What we're seeing here is the path (in white) and the shape that'll follow that path (in green). Make sure the shape is selected (in green). If not, select it from the left panel which shows both sketches.

With the shape selected, click the Additive Pipe tool.

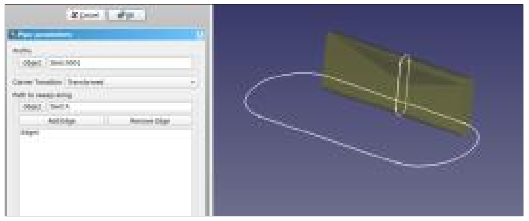


Additive pipe

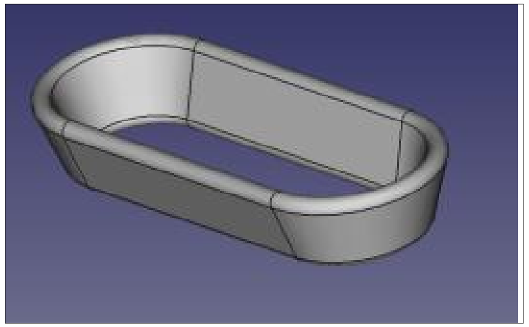
Sweep a selected sketch along a path or to other profiles

(PartDesign_AdditivePipe)

In the left panel, click Add Edge and select a piece of the path.



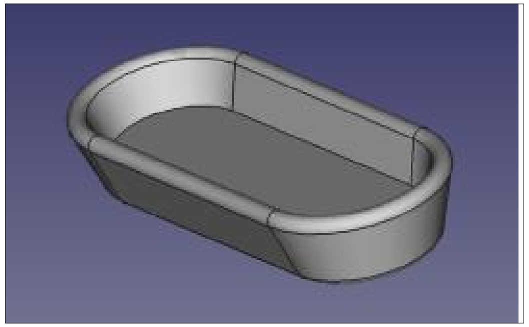
Don't worry about it looking semi-transparent or the color of it. This is just a preview. Click Add Edge and add the remaining edges. When all is done, click OK.

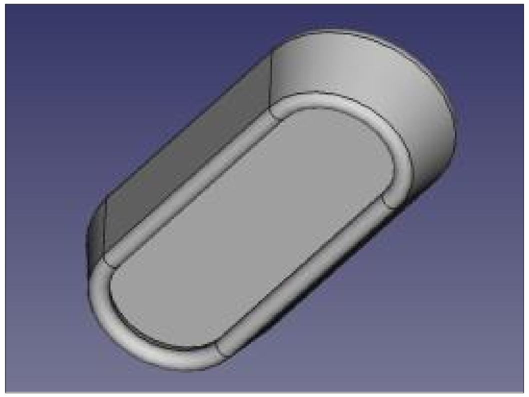


Nice! Now we have an irregular shape as a path. Now, be careful with the paths you make, as a wonky angle or something similar will cause your shape to also go a bit wonky. So, if your shape doesn't close properly, it's probably down to a wonky path.

Let's finish up by adding a crude bottom to our makeshift bath shape.

Click the sketch for the path (you may need to click the arrow beside AdditivePipe in the left panel to see the sketches), and click the Pad tool. As ever, edit the thickness of the padding.





You can, of course, double-click the sketch (in the left panel) to edit the path to then edit the object. Give it a try.

Inkscape - Part 135

Correction: This article includes a fix to address an issue with the code published last month which failed to work if Inkscape inserted the <script> tag near the top of the SVG file.

```
function createViewElements() {
  const inkNS = 'http://www.inkscape.org/namespaces/inkscape';
  const svgNS = 'http://www.w3.org/2000/svg';
  const pages = document.getElementsByTagNameNS(inkNS, 'page');

  Array.from(pages).forEach((page, idx) => {
     const x = page.getAttribute('x');
     const y = page.getAttribute('y');
     const w = page.getAttribute('width');
     const h = page.getAttribute('height');
     const viewBox = `${x} ${y} ${w} ${h}`;

     const view = document.createElementNS(svgNS, 'view');
     view.setAttribute('id', `page-${idx + 1}`);
     view.setAttribute('viewBox', viewBox);
     page.appendChild(view);
  });
}

window.addEventListener('load', createViewElements);
```

Last time I demonstrated how to write a small chunk of JavaScript that can be added to a multi-page Inkscape file in order for all the pages to be viewable in a web browser by appending '#page-2' to the URL (without the quotes, and replacing '2' with the relevant page number). That's a useful step forward, but Inkscape lets you explicitly name pages, so it would be nice if we could use those names in the URL, rather than a generic string. This month I'm going to extend the existing script to add exactly this capability. But before I do, I need to address an issue that was discovered by 'dwhall' on the official Inkscape forum.

Although I tested the code extensively before writing the previous article, dwhall was unable to get it working in their own files. With a bit of digging, I was able to track the problem down : my test files had the <script> tag inserted at the end of the file, whereas theirs was inserted near the start. I'm not sure why this difference occurred – we had both used various Inkscape releases, and I'd also made some manual edits while testing the code, so the difference could potentially have been introduced in a number of different ways. Having the <script> tag near the start meant that the JS was being executed before the SVG content had finished being read and rendered, so the call to getElementsByTagName(), for example, was failing to return the correct elements. That same code worked fine when it was at the bottom of the file.

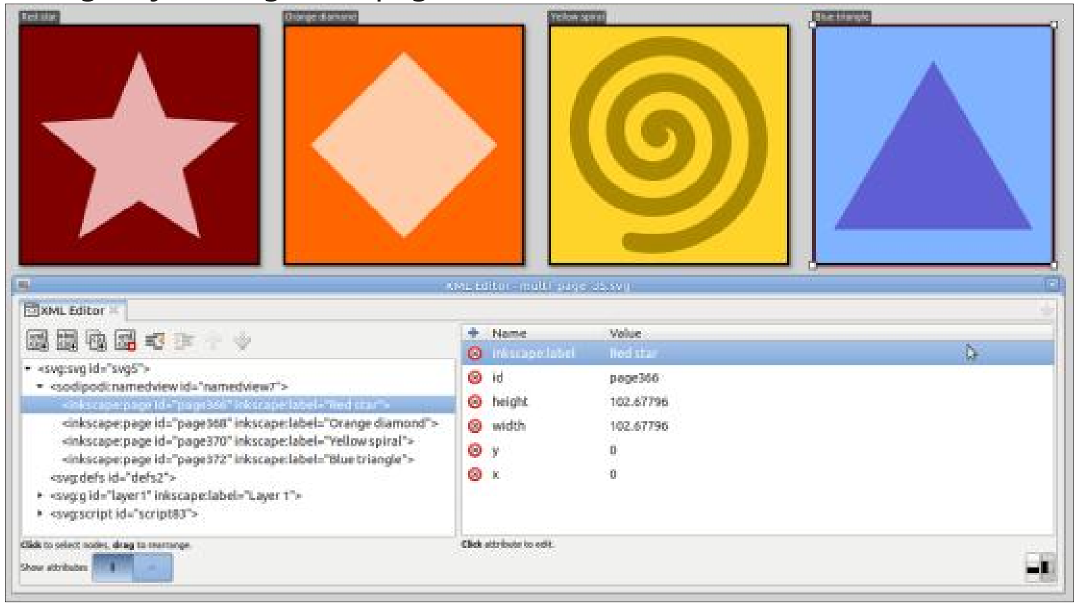
One solution to this is to manually move the <script> tag to the end of the file – but that involves editing XML in a text editor, which is not a very user-friendly approach. Instead I've modified the code by wrapping it all in a function, named 'createViewElements()'. That stops the code running immediately as the JavaScript is parsed, but we still need some way to make sure it still runs once the page has loaded. The answer to this is to add a line at the end, outside the function, which listens for the browser's 'onload' event, and responds by calling the function.

With these changes in place, the code now looks like that shown above.

I've added line numbers to make it clearer which parts of the code I'm referring to in the rest of this month's article, but I'm using the convention from my BASIC days back in the 80s of incrementing the numbers in tens so that I can squeeze new lines of code in-between as necessary. Do not include any line numbers when entering this code into Inkscape! I've also removed the console.log() calls that I'd previously left in, for brevity. If your own code still contains them it won't cause any problems.

If we're going to make this code support named pages, we'll need to begin by naming some pages in Inkscape, then open the XML editor to see how those values are stored in the file. Using the same demo page as last time, I've given each page a name based on its contents – 'Red star', 'Orange diamond' and so on.

It may not be very obvious by the time this screenshot has been resized and compressed to fit the magazine's PDF file, but if you do this yourself you'll quickly find that the page names are stored as part of the <inkscape:page> element (no surprise there!), in an attribute called 'label' which is also in the 'inkscape' namespace. The use of the namespace on the attribute seems a little unnecessary to me, given that the whole element is



```
▾ <svg:svg id="svg5">
  ▾ <sodipodi:namedview id="namedview7">
      <inkscape:page id="page366" inkscape:label="Red star">
      <inkscape:page id="page368" inkscape:label="Orange diamond">
      <inkscape:page id="page370" inkscape:label="Yellow spiral">
      <inkscape:page id="page372" inkscape:label="Blue triangle">
    <svg:defs id="defs2">
  ▸ <svg:g id="layer1" inkscape:label="Layer 1">
  ▸ <svg:script id="script83">
```

Name	Value
inkscape:label	Red star
id	page366
height	102.67796
width	102.67796
y	0
x	0

```
160      page.appendChild(view);
161
162      if (label !== null) {
           // More code to go in here…
169      }
170    });
```

non-standard and in a custom namespace already, but we have to work with the format we've got, not what we think it should be.

We'll need to extract the page names into JavaScript variables for further processing. In reality our code already works on one page at a time, courtesy of the forEach() loop, so we actually only need to get the name from the attribute in the page that's currently being processed. We've already used the getAttribute() function to extract attribute values, but this time we've also got that pesky namespace to deal with. That requires the getAttributeNS() function which is exactly the same, but takes a namespace as the first parameter. Despite this small difference, we'll keep all the attribute-getting code together, resulting in the insertion of line 105:

```
105   const label =
page.getAttributeNS(inkNS,
'label');
```

It's entirely possible that the label attribute doesn't exist on a particular page. You should be free to just add labels to some pages and not others, without the code breaking. Fortunately, getAttributeNS() won't just fall over in that case, but rather it will return a JS null value. So if 'label' ends up holding a string, we know that the page had a label applied, whereas if it is null, there was no page label. We can use that to write an 'if' statement containing code that should only run if a label is present. This goes inside the end of the existing loop, so the file looks like this (top right).

Note the jump in line numbers after 162, as we'll need more than just a single line of code in this section. The first thing we'll need to do in here is to ensure that the label is valid for use as the ID for a <view> element. IDs have pretty strict rules about what characters they're allowed to contain, whereas Inkscape page labels are

contents ^

far more lax. We'll take a rather basic approach to this problem: we'll just replace every character that's not alphanumeric with an underscore.

```
163  const idLabel =
label.replace(/\W/g, '_');
```

This calls the JS replace() method on our string, passing it two parameters. The second is pretty obviously the replacement string we want to use, but what about that first one? That's a JavaScript regular expression – similar to those that you may be familiar with from grep or sed on the command line, but with some significant differences in syntax. In JS a regular expression is delimited by slash characters ('/') in much the same way that a string is delimited by quotes. So the actual regular expression itself is just the '\W' between those slashes. What does that mean?

In the command line world you might be familiar with character classes in regular expressions – special values such as [:alnum:] which match against the range of alphanumeric characters (equivalent to [a-zA-Z0-9]). JS has its own set of character classes,

```
130     const view = document.createElementNS(svgNS, 'view');
140     view.setAttribute('id', `page-${idx + 1}`);
150     view.setAttribute('viewBox', viewBox);
160     page.appendChild(view);
```

with the closest match to [:alnum:] being \w (lower-case 'w'). This actually matches not only alphanumeric characters, but also the underscore character. No, I don't know why they decided to throw an arbitrary underscore into the mix, but that's the way it works. Each of these character classes has an inverse, using the upper-case version of the letter. So the '\W' used in our code matches any character that is NOT an alphanumeric character or an underscore.

The 'g' at the end of the regex is a flag that turns this into a 'global' regular expression. All that means is that it will apply to all the substrings that match, not just the first one. Without this a label such as "Red star #1" would be transformed into "Red_star #1". With this one extra character the transformation becomes "Red_star__1" instead. Note that both the second space and the hash character are transformed, resulting in a double-underscore in the final string. If you prefer to collapse sequences of multiple matching characters down to a single underscore, replace the regex with /\W+/g instead. The additional '+' will cause the '\W' to not just match a single character, but any consecutive sequence of one or more characters, replacing entire spans of non-alphanumerics with a single underscore. I think this is usually a little more user-friendly, so I'll include the '+' in the final code.

Now that we've got an ID-friendly name, we can create the new <view> element, just like we did last time. In fact, the code will be virtually identical, which makes it a prime target for moving out into a separate function. For our purposes we want a function that takes the strings for the ID and viewBox as parameters, as well as a reference to the current <page> element. It will create the new <view> and append it to the <page>, as before. The current code looks like this (above).

Let's extract lines 130 to 160 into a function, which we'll put between the end of the existing function (line 180) and the event listener (line 200). The main change in this code is that the ID that we previously generated on line 140 is now going to be passed in to the function call, so on line 184 below we use the parameter name. We'll also move the svgNS variable (previously on line 30, now on line 183) as it's only used by the createElementNS() function and no longer needs to exist in the main function (next page, top right).

If you've kept up so far, then well done! It's time to take a break and test the new functionality. You should be able to access each page either by appending '#page-n' to the URL (e.g. file.svg#page-3) or by appending a hash followed by the page label, with non-alphanumerics replaced by underscores. With my test file, appending '#Orange_diamond' to the URL produces this (image shown next page, bottom left).

I've opened the developer tools and switched to the Inspector in order to see what's actually happened in the document. Sure enough, each of my <inkscape:page> elements now contains two children, both <view> elements, with an ID corresponding to the page number, and another corresponding to the label. Either can be used as a fragment identifier in the URL to give exactly the same result.

You can stop there if you want. We've achieved the goal we set for ourselves, of being able to access pages by their label. But with little effort we can make this script even more useful. There's a way we can apply those additions to the file,

```
180   }
181
182   function insertSVGView(page, id, viewBox) {
183     const svgNS = 'http://www.w3.org/2000/svg';
184     const view = document.createElementNS(svgNS, 'view');
185     view.setAttribute('id', id);
186     view.setAttribute('viewBox', viewBox);
187     page.appendChild(view);
188   }
190
200   window.addEventListener('load', createViewElements);
```

Now we can replace lines 130-160 with a single call to the new function, as follows:

```
130     insertSVGView(page, `page-${idx + 1}`, viewBox);
170   });
```

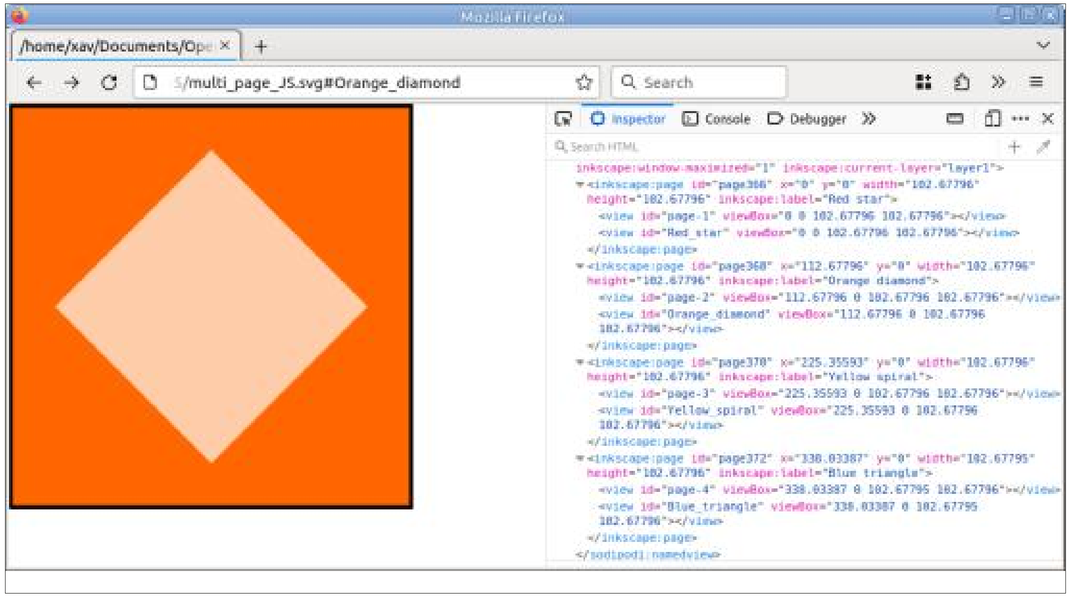
Save the file and make sure it still works as it did previously. Assuming it does, then our last step is to call the new function inside our 'if' block, creating another <view> element if the page has a label. This is as simple as it sounds – just adding a new line after we've created the ID:

```
162   if (label !== null) {
163     const idLabel = label.replace(/\W/g, '_');
164     insertSVGView(page, idLabel, viewBox);
169   }
```

permanently.

Currently our SVG file has a significant limitation. The <view> elements are created dynamically when the JavaScript runs – but JS only runs when we load the SVG file directly, or via an <object> tag, neither of which are common in today's web. More often our files are pulled into a page via an HTML <img> tag, or a CSS url() function, and in those cases we can't use fragment identifiers unless we manually edit the file to add <view> elements as described a couple of articles ago.

One workaround to this is to add the code from this article, load the SVG directly into a web browser, and then save the page (with a new name) from within the web browser. That creates a new file that contains the current version of the document – i.e. the one which contains the <view> elements that the code created. We've just automated the task of adding the new elements to the



```
inkscape:window-maximized="1" inkscape:current-layer="layer1">
▾<inkscape:page id="page366" x="0" y="0" width="102.67796"
 height="102.67796" inkscape:label="Red star">
   <view id="page-1" viewBox="0 0 102.67796 102.67796"></view>
   <view id="Red_star" viewBox="0 0 102.67796 102.67796"></view>
 </inkscape:page>
▾<inkscape:page id="page368" x="112.67796" y="0" width="102.67796"
 height="102.67796" inkscape:label="Orange diamond">
   <view id="page-2" viewBox="112.67796 0 102.67796 102.67796"></view>
   <view id="Orange_diamond" viewBox="112.67796 0 102.67796
   102.67796"></view>
 </inkscape:page>
▾<inkscape:page id="page370" x="225.35593" y="0" width="102.67796"
 height="102.67796" inkscape:label="Yellow spiral">
   <view id="page-3" viewBox="225.35593 0 102.67796 102.67796"></view>
   <view id="Yellow_spiral" viewBox="225.35593 0 102.67796
   102.67796"></view>
 </inkscape:page>
▾<inkscape:page id="page372" x="338.03387" y="0" width="102.67795"
 height="102.67796" inkscape:label="Blue triangle">
   <view id="page-4" viewBox="338.03387 0 102.67795 102.67796"></view>
   <view id="Blue_triangle" viewBox="338.03387 0 102.67795
   102.67796"></view>
 </inkscape:page>
</sodipodi:namedview>
```

file, avoiding the need for our error-prone manual editing.

That works, but leaves us with a small problem. The new file already contains the additional <view> elements… but it also still contains the code that creates new <view> elements. If that file is loaded directly into a browser, all the new <view> elements get created again, resulting in twice as many as we wanted. Here's how it looks in the Firefox developer tools (bottom left).

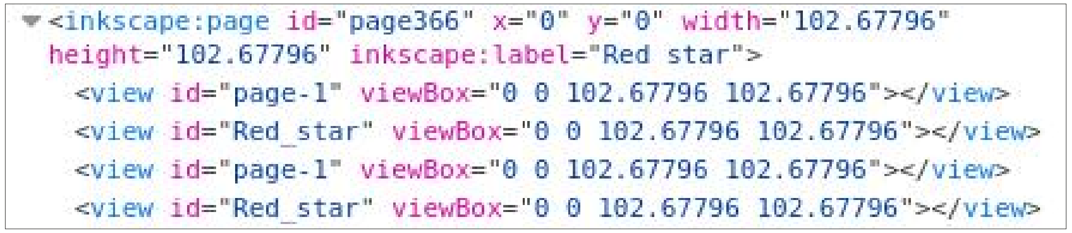
To fix this we need to modify the function that creates the <view> elements to test for the existence of the ID in the page. If it already exists then the function just exits before creating any new elements. Otherwise it carries on as before. We can achieve this with a couple of lines at the top of the function. They go after line 182, but as I've already used the line numbers after that, I'll show them without numbers in the code below (it was the same in the 80s – the idea of numbering in tens falls down as soon as you need to insert 11 new lines!). Shown top right.

The first line assigns null to the 'foundID' variable if the ID does NOT already exist in the page. Otherwise it assigns a reference to the element. The second line then tests 'foundID' and returns from the function if it's not null (i.e. if the element already exists).

With that in place, our enhanced function is complete. Here's a breakdown of how to use it:
• Create a multi-page Inkscape document.
• Optionally add labels to some or all of the pages.
• Add this entire JS script via the Document Properties dialog.
• Save the file.
• Load it directly into a web browser. That triggers the creation of the extra <view> elements.
• Save it from the web browser, fixing those new elements into the file.

Now you can load it any way you want, and access the additional pages with a fragment identifier based on the page number or label.

Finally, next page, here's the complete code we ended up with. No line numbers this time, to make it easier for you to copy and paste it into Inkscape.



```
▾<inkscape:page id="page366" x="0" y="0" width="102.67796"
 height="102.67796" inkscape:label="Red star">
   <view id="page-1" viewBox="0 0 102.67796 102.67796"></view>
   <view id="Red_star" viewBox="0 0 102.67796 102.67796"></view>
   <view id="page-1" viewBox="0 0 102.67796 102.67796"></view>
   <view id="Red_star" viewBox="0 0 102.67796 102.67796"></view>
```

```
function createViewElements() {
  const inkNS = 'http://www.inkscape.org/namespaces/inkscape';
  const svgNS = 'http://www.w3.org/2000/svg';
  const pages = document.getElementsByTagNameNS(inkNS, 'page');

  Array.from(pages).forEach((page, idx) => {
     const x = page.getAttribute('x');
     const y = page.getAttribute('y');
     const w = page.getAttribute('width');
     const h = page.getAttribute('height');
     const label = page.getAttributeNS(inkNS, 'label');
     const idLabel = label.replace(/\W+/g, '_');
     const viewBox = `${x} ${y} ${w} ${h}`;
     insertSVGView(page, `page-${idx + 1}`, viewBox);

     if (label !== null) {
       const idLabel = label.replace(/\W/g, '_');
       insertSVGView(page, idLabel, viewBox);
     }
  });
}

function insertSVGView(page, id, viewBox) {
  const foundID = document.getElementById(id);
  if (foundID !== null) return;

  const svgNS = 'http://www.w3.org/2000/svg';
  const view = document.createElementNS(svgNS, 'view');
  view.setAttribute('id', id);
  view.setAttribute('viewBox', viewBox);
  page.appendChild(view);
}

window.addEventListener('load', createViewElements);
```

Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

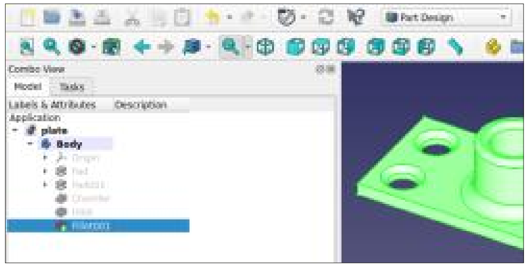
FreeCAD - Part 7

OK, so you've got a thing, but how do you get it to your printer? Well, I'm glad you asked.

First things first: FreeCAD is only for making the object. It won't prepare or send an object to your printer. For that, you need something like Ultimaker Cura (https://ultimaker.com/software/ultimaker-cura). It will take the object and 'slice' it. It will then export a gcode file that tells your printer how to print each slice.

Export

I'm pretty sure we covered saving a FreeCAD thing, but if not: File > Save As. Exporting is different. It will make a file that has your thing as a finished model. If you need to go back and edit something then you need the FreeCAD file.



First, make sure you have the last thing in the left panel selected. Otherwise, FreeCAD won't export.

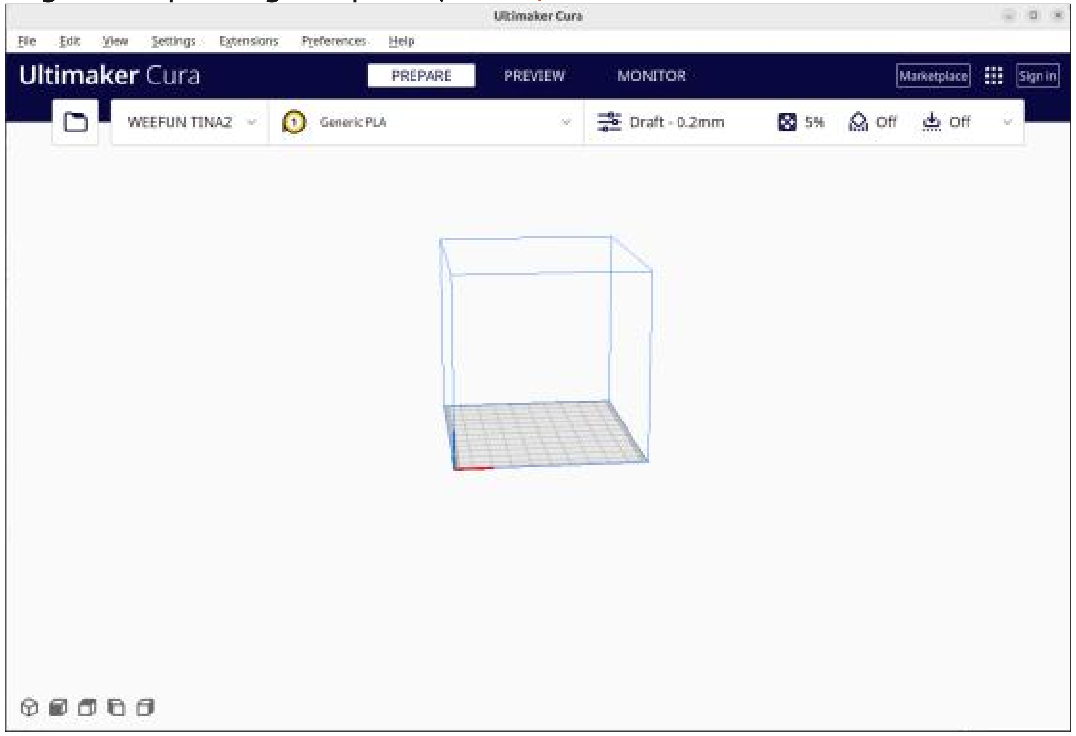
Do a File > Export and save as an STL file. This is pretty much the standard for 3D printing models.

Slicing

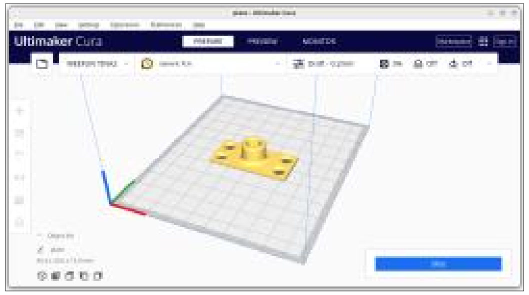
As I mentioned previously, We're now done with FreeCAD with regards to printing this plate (as I've called it).

Load up Cura. I'm using the AppImage version. So a double click on the downloaded file will start it.

The next part is outwith this series, but you need to give Cura your printer settings. Cura comes with a huge list of printers that it supports. If your printer isn't on the list then maybe contact the makers and ask if they have a Cura settings file. The makers of my Tina2 supplied a Cura settings file which was nice.

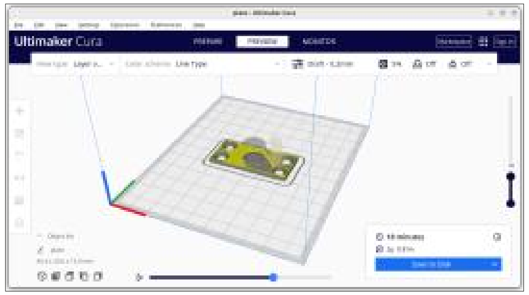


Head to the Cura menu and click File > Open File(s) and select the



STL file.

Last thing is to click the 'Slice' button to have Cura analyse the model to see how it should be printed. It'll give you a print estimate time (which is a nice feature), but click Preview to see



what the printer will do.

In this window you can view each slice and using the sliders show exactly what the printer will do.

Last thing to do is click 'Save to Disk' and save the sliced model as a gcode file.

Now it's a case of getting the gcode file to the printer. In my case I copy it to a memory card and pop that into the printer, choose the file on the little screen, and click print. Then it's a case of making a cuppa and coming back after, in this case, about 15 minutes.

There's more to Cura

I've greatly simplified the Cura process here as this series is focusing on FreeCAD. With Cura you can have different adhesion for the printing process (skirt, platform, etc.), different infill amounts (for saving PLA by making the object hollow, but still sturdy), supports (for overhanging parts). The list goes on.

Well, that's about it for this series. I hope you enjoyed it. I have to be honest and say I don't know enough about FreeCAD to write more parts. If you know more about FreeCAD and would like to continue the series, feel free to email me: ronnie@fullcirclemagazine.org