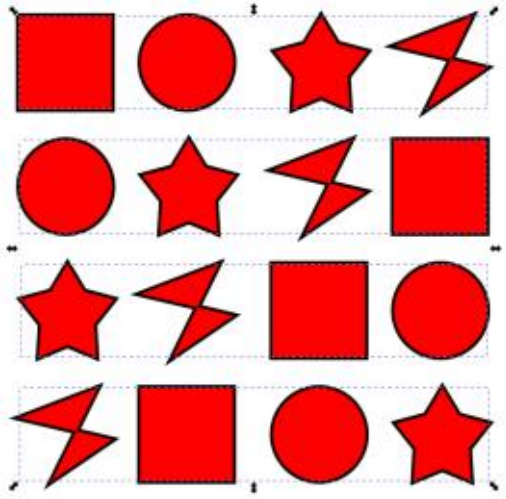
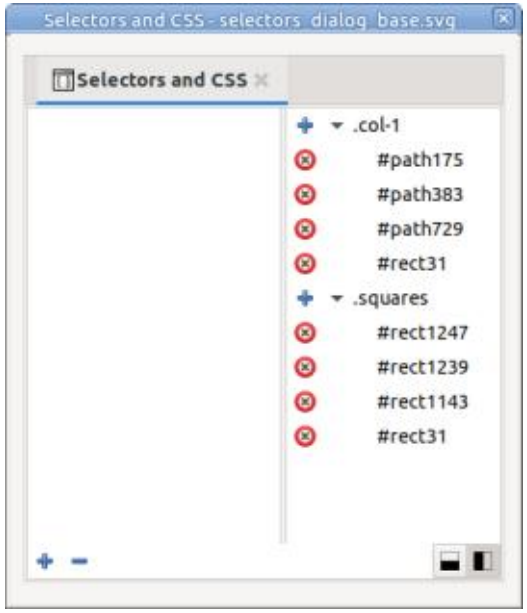
Inkscape - Part 115

This month is the last part of our exploration of the new Selectors and CSS dialog ("Selectors dialog" for short). As with the previous parts, we'll be working with this collection of shapes arranged as four groups of objects, one for each row.

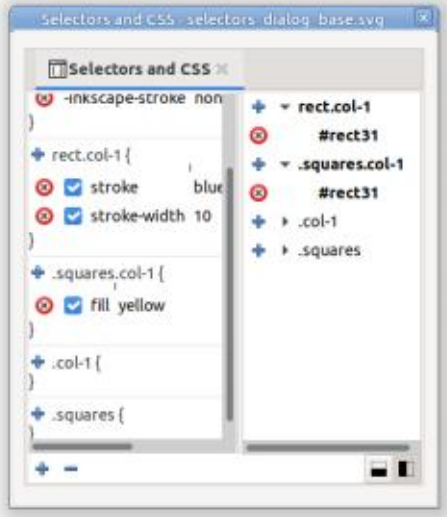


We've previously looked at using class and element selectors in the dialog, but, as you may recall from part 112, CSS allows for a variety of other selectors, and different ways to combine them. Inkscape's CSS parser doesn't support all the various possibilities, but it does manage some of them. Whether or not they're actually useful to you is another question entirely, but, in this instalment, I'm going to take a look at some advanced selectors that do work, as well as some that unfortunately don't.

Starting with a blank Selectors dialog, I've created a class called "squares" that includes all the squares in our sample image, and a second class called "col-1" which contains all the elements in the first column of objects. As you can see, the "#rect31" element appears in both, as you would expect.



Deselecting everything in the drawing, and clicking the plus button at the bottom of the dialog, triggers the dialog for adding a new selector – pre-filled with ".Class1" as usual. As we saw last time, it's also possible to enter an element name here, but, this time, we'll create something even more complex: a selector that targets multiple classes. If we enter a value of ".squares.col-1" as a single string, with no spaces, the selector will target only those elements that have both the "squares" and "col-1" classes applied. In our case the only thing that matches that combination is the square at the top left.



Alternatively, we can combine an element selector with a class selector. Entering "rect.col-1" for example, will only match those <rect> elements that also have a class of "col-1" applied. Again, the only thing that matches in this case is the square at the top left.

Using these selectors, I've added rules to change the color of any

object where both classes match, and the stroke of any <rect> with the "col-1" class. The effect is that the square at the top left has both the new rules applied, but no other elements are affected.

The ability to combine classes in this way could potentially be useful, especially if you want to use class names to categorize your elements. When producing game assets, for example, you might have multiple images in a single document, covering different types of landscape in different seasons, and with different assets. Need to quickly find the image for a wintery forest tile with a mine? Add a new selector for ".winter.forest.mine" – assuming you've already set the right classes on your images.

Combining elements and classes is probably less useful, particularly given how many of Inkscape's primitives are actually just <path> elements in the underlying SVG, and are therefore indistinguishable from each other via a simple element selector. If you can think of a good use case for this, however, it's nice to know that Inkscape already supports the format.

The fact that Inkscape piles its own internal attributes onto <path> elements in order to support some of its basic shapes does lead onto another type of CSS selector that's worth further examination: the attribute selector. In the world of CSS it's possible to select elements that have a particular attribute, or which have an attribute set to a particular value. The table below covers the main selectors that will work in a web browser.

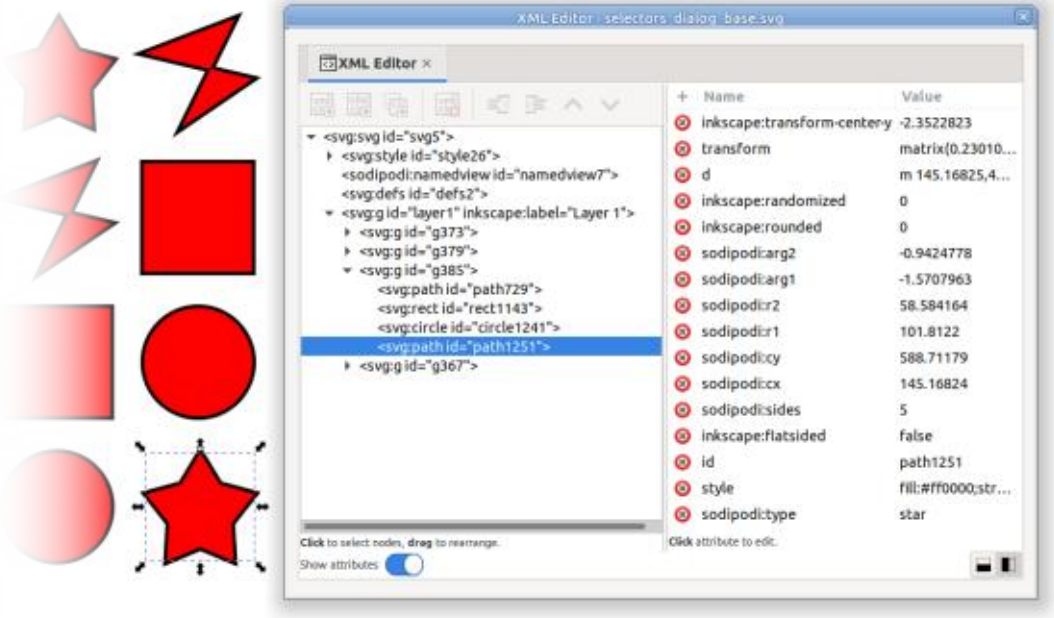
Selector	Description	Example
[class]	Match any element with the "class" attribute.	<rect class="squares" … />
[class="col-1"]	Match any element whose "class" attribute *exactly* matches the string "col-1".	<rect class="col-1" … />
[class~="col-1"]	Match any element whose "class" attribute is a whitespace separated list in which any of the values match the string "col-1".	<rect class="squares col-1" … />
[class*="col"]	Match any element whose "class" attribute contains the string "col" anywhere within it.	<rect class="col-1" … /> <rect class="my-col-2" … />
rect[class]	Attribute selectors can be combined with other selectors – in this case to find any <rect> which has a "class" attribute.	<rect class="squares" /> <rect class="col-3" />

There are other variations to find substrings only at the start or end of the value, or which force case sensitivity when matching, for example. If you're a web developer who wants to know more about attribute selectors, I recommend looking them up on the Mozilla developer site (link at the end of the article).

For our purposes, these should allow us to distinguish between different types of primitive in Inkscape. Let's look at one of our star shapes in the XML editor (shown bottom right).

That's a long list of attributes, but the one we're most interested in is the last one: "sodipodi:type" with a value of "star". As I've discussed previously in this column, the "sodipodi" part is the namespace for this attribute, which is required because it's not part of the SVG specification. In practical terms, however, I usually just refer to this as the "type" attribute.

Suppose we want to target just the stars with a CSS rule. Based on the table of attribute selectors you might expect [type="star"] to do the job, but it doesn't. Not even a more basic selector of just [type] is accepted by Inkscape.



Perhaps it's that pesky namespace. How about [sodipodi:type] or [sodipodi:type="star"]? No, they don't work either. In fact namespaces in CSS are a bit of a pain, requiring you to redefine your prefixes in the CSS in addition to the definition in the XML itself. In the selector rule, the namespace is then separated from the value by a pipe character, not a colon. So, in theory, manually adding an "@namespace" rule to the stylesheet, then using [sodipodi|type="star"] should do the job. But not in Inkscape.

I've tried many, many different combinations of attribute selector, both with and without namespaces, but can't get any of them to work within Inkscape. This is a real shame as it makes it impossible to target specific Inkscape primitives, or elements with other proprietary attributes.

If you're a web developer, then you may wish to know that these selectors do work as advertised in web browsers, provided you include the CSS namespace declaration and use a pipe separator. This <style> block, for example, will cause all the stars in the test document to appear with an orange fill within a web browser, but it doesn't work when the same file is loaded into Inkscape.

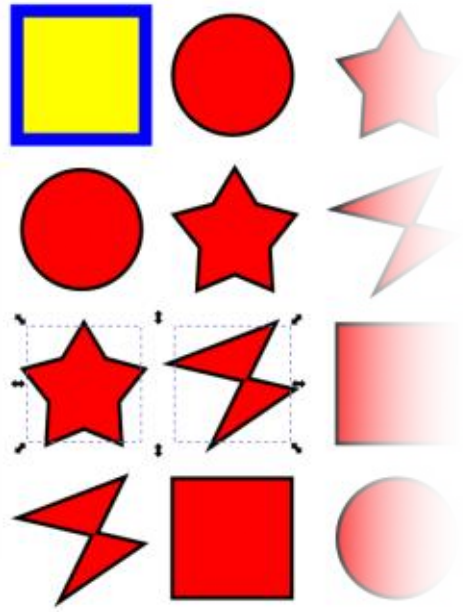
```
<style id="style258">

   @namespace sodipodi
url(http://
sodipodi.sourceforge.net/DTD/
sodipodi-0.dtd);

  [sodipodi|type="star"] {
    fill: orange !important;

  }

</style>
```

Although Inkscape doesn't work with attribute selectors, there are a few other useful CSS rules that it does seem happy with. First, we have the descendent selector: simply enter two selectors with a space between them, and the rule will match only if the element matching the second selector is some descendent (in the XML structure) of an element matching the first selector.

For example, in our test file, each row of objects is in a separate group, and I've set the ID for each group (using the Object Properties dialog) to "row-1", "row-2" and so on. If I want to select all the <path> elements in row 3 (i.e. the star and Bézier path objects), then I can create a selector with the string "#row-3 path" to find every <path> element, but only if it's a descendent of the element with the ID "row-3".

Note that "descendent" in this case means any child, grandchild, great-grandchild and so on down the tree. If you want to select only immediate children, then use a ">" character between the individual parts of the selector (with optional white space around it for readability).

Sometimes you might want a single set of CSS rules to apply to several different selectors. In that case, you can comma-separate the individual items. So, to create a rule that matches three specific elements by their IDs, use something like "#rect31, #rect1247, #path729". The easiest way to achieve that is just to select them all before you press the "+" button at the bottom of the dialog: the field that opens for you to enter a selector will be pre-populated with a comma-separated list of the IDs. You can use commas to mix any types of selector, so creating a rule to match any immediate child of the second row plus any circle, would result in this: "#row-2 > *, circle". Notice from



this example that you can also use an asterisk ("*") to match any element, regardless of its type or class.

Comma-separated lists of selectors can easily become long and unwieldy when you need to match lots of different things. CSS has a couple of features in the form of the ":is()" and ":where()" pseudo-classes that can simplify many such rules. Although Inkscape does let you enter them into the dialog, unfortunately they don't actually work in the program.

Also in the list of useful CSS rules that don't work is the ":not()" pseudo-class. This should allow you to select elements that don't match a particular rule. E.g. ":not(path)" for selecting all the non-path elements. Instead, Inkscape just swallows the new rule when you enter it – without it then appearing in the dialog at all. Prefixing it with a class selector (e.g. ".row-1:not(path)") allows it to appear, but it certainly doesn't work as it should.

There are some pseudo-classes that do sort-of work with Inkscape, but not well enough to be genuinely useful. The ":first-child", ":last-child" and ":nth-child()" selectors work, but only if they're applied to a class or ID selector. For example, ".squares:first-child" will select any element with the "squares" class that is the first child of its parent. In the example file that will match the square in the top left, as it is the first child of the group element that holds the row. In theory you should be able to use just ":first-child" or "*:first-child" to match the first element of any parent but, in practice, that doesn't work at all. This is a real shame as it makes it practically impossible to use the powerful ":nth-child()" pseudo-class to select all the odd children of a group, or every fourth one, for example.

A related set of selectors are ":first-of-type", ":nth-of-type", plus some other "-of-type" strings. Trying to use these will actually cause Inkscape to crash entirely, so definitely steer clear of them!

To summarise my findings, the Selectors dialog is effective when used with simple class, ID or element selectors, including combining them in a comma-separated list. But most of the more powerful CSS rules either don't work at all, don't work as expected, or might even kill the program. It's probably best to see this dialog for what it is: a replacement for the Selection Sets dialog that also lets you set some CSS rules in a stylesheet, should you wish to. If you're enough of a developer to specifically need a stylesheet in your document, then you're probably better off managing it outside of Inkscape for now. If you want to manage and store just some simple selections, on the other hand, then this dialog should serve you well enough.

Over time, it's likely that Inkscape's CSS capabilities will improve, and perhaps some of the more complex rules will be supported. But, for now, you're best to either keep it simple, or to do it by hand.

Links

Overview of CSS resources and tutorials on MDN:
https://developer.mozilla.org/en-US/docs/Web/CSS

Attribute selectors on MDN:
https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors

The @namespace rule on MDN (works in browsers, not in Inkscape):
https://developer.mozilla.org/en-US/docs/Web/CSS/@namespace

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**

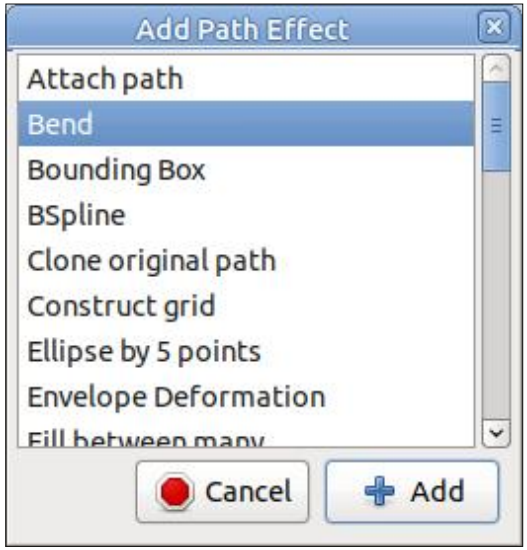
Inkscape - Part 116

This month, we're moving on from the Selectors and CSS dialog, to the last of the big changes that were made with the release of the 1.0 and 1.1 series of Inkscape versions: Live Path Effects (LPEs).

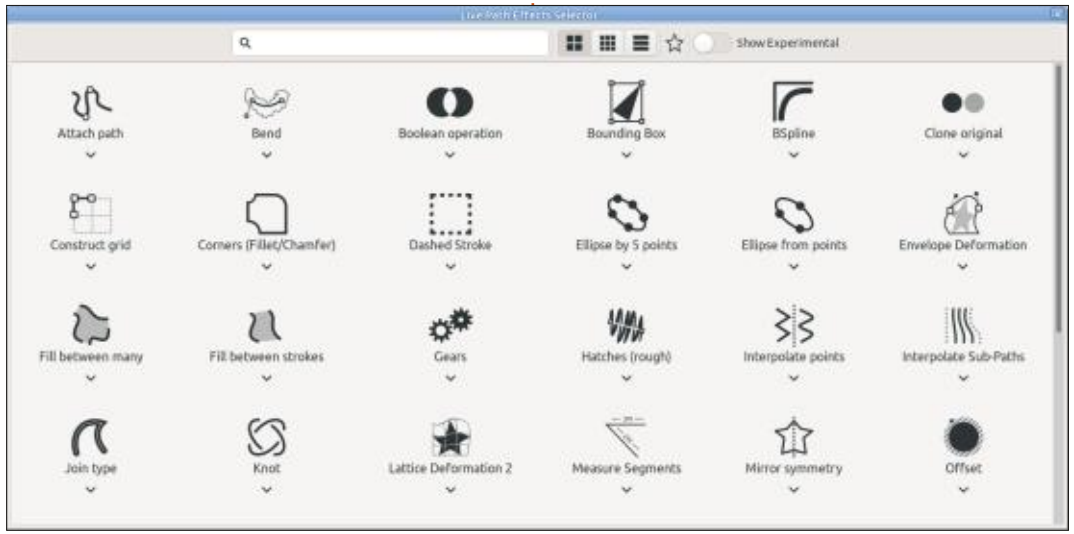
LPEs themselves are not new, of course. They've been a staple of Inkscape since version 0.46, way back in 2008, but have seen considerable improvements with every release. With 1.0, the user interface was radically overhauled, so, this month, I'll be concentrating on those changes. The following months will then take a deeper dive into the new effects that arrived with 1.0 and 1.1.

If you're new to LPEs, you may want to take a look at parts 42 – 47 of this series for a general introduction, and the effects that were available in v0.48; then parts 65 – 69 for the effects that were added with v0.91 and v0.92.

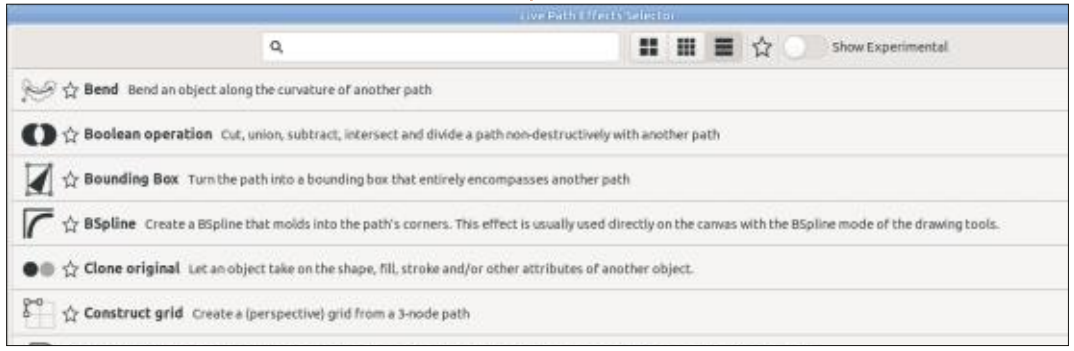
One thing that hasn't changed much with the new releases is the initial LPE dialog, opened via Path > Path Effects… (or Ctrl-Shift-7). The content of this will remain disabled until a path is selected, at which point you're presented with a rather empty dialog. Just about all you can do at this point is to click the "+" button at the bottom in order to add your first LPE to the effects chain. On 0.92, the available LPEs are displayed in a list, like this:



It does the job, but it is somewhat utilitarian. As the total number of LPEs grew, it became clear that something more functional was required. Version 1.0 takes that requirement and hits it out of the ballpark, with a vastly more powerful dialog (see above).



Immediately you can see the biggest change is the switch from a simple list of titles, to a grid of icons that represent what each LPE does. This alone is a huge improvement, as it's generally much easier to find the effect you're looking for with the aid of the icons rather than by title alone. If you really prefer a list view then you can select this using the buttons at the top of the dialog. These allow you to choose between two densities of grid, or a list view that is still more useful than the old one, as it also includes a smaller version of the icon plus some descriptive text (see below).

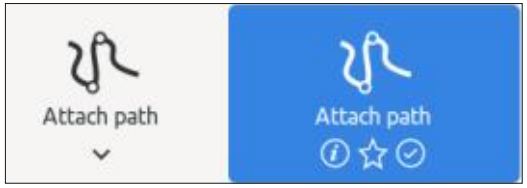


You won't be at all surprised to hear that the search box at the top of the dialog can be used to filter the list of effects based on a simple substring search that looks at both the effect name and the description. This applies even in the grid view, when the descriptions aren't so obviously visible.

In the list view, you'll notice that each entry has a small star between the icon and the effect name. Clicking this will mark (or un-mark) that effect as a "favorite". The visible list can then be restricted to show only the favorites using the star icon in the toolbar. It's important to note that, when showing just the favorites, any text typed into the search box will be tested only against favorites, not against the other hidden effects.

It's also important to note that the clickable areas on each row are a little non-standard in some respects. Hovering over the star doesn't change the cursor to indicate it is clickable, and doesn't produce a tooltip to that effect either. Clicking it does toggle the state, indicated by a filled or open star, but has no other side-effects. The rest of the row, on the other hand – whether that's the icon, title or description – changes the cursor to indicate that it is clickable. If you do click the mouse button, it will add the effect to the main LPE dialog and immediately close this one. Take care, therefore, when trying to (un)mark a favorite, as a slight mis-click could easily lead to the effect being added to the effect chain by mistake.

Similar care needs to be taken in the grid view. Clicking on an effect's icon or title will, again, immediately add it to the effect chain and close this dialog. Below each entry, however, is a small downward-facing chevron: clicking this does not add the effect to the chain, but rather selects it and displays three icons, as shown in this before/after screenshot:



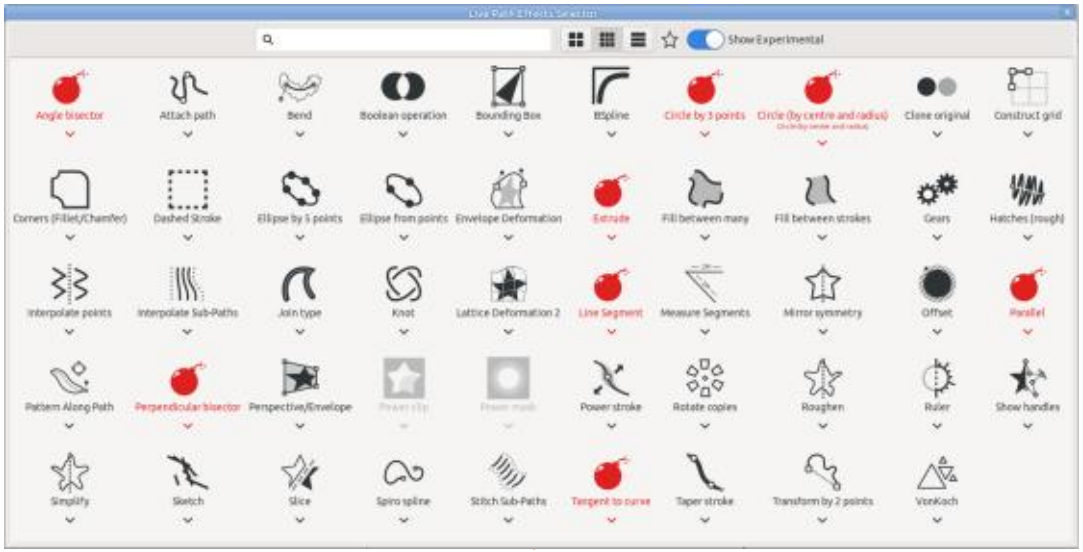
The three icons all behave quite differently. Hovering over the first will display a pop-up showing the icon, title and description. This is the only way to view the description in grid mode – unfortunately the developers haven't exposed them via tooltips on the main icons or titles. There is no change in the cursor when hovering over this icon, but clicking it will add the effect to the chain and close the dialog.

The second, star-shaped icon toggles the favorite status of the effect, as you might expect. As with the list view, there's no change of cursor, nor a tooltip to describe this behaviour, and clicking here will not add the effect to the main dialog.

Finally, the third icon (a tick in a circle) seems a little redundant. It appears to be there as a means for you to confirm your selection, causing the effect to be added to the chain and the dialog to be closed. Given that clicking almost every other part of this widget has the same effect, however, it seems unnecessary. It is worth noting, however, that the clickable area doesn't cover the entire size of the colored background: the large, empty spaces to the left and right are not clickable (and do not change the mouse cursor) which I find a little misleading, but not a huge problem in practice. As I'm being picky about the UI, though, I do think the developers should nudge the favorites toggle up a couple of pixels. I'm sure it's perfectly aligned numerically, but the difference in visual weight between a circle and a star does make it look like it's sitting a little low compared to its siblings.

The final part of the UI for this dialog is the slide switch at the right of the toolbar, labelled "Show Experimental". Clicking on the switch itself (the label isn't clickable – a classic UI mistake) reveals or hides any LPEs which are included in your Inkscape release but still considered experimental by the developers. Unfortunately, these all get the same "cherry bomb" icon, which indicates that they are risky to use but doesn't provide a quick indication of what each effect actually does, as a normal icon would. I would much rather see them distinguished by having the cherry bomb as an additional tag or emblem attached to the main icon.

The exact list of additional effects that are exposed by this switch will vary depending on your Inkscape release, but could be substantial. On my 1.1.1 version, for example, an additional eight effects become available, which is quite a percentage of the 49 that are present in total. I've colored



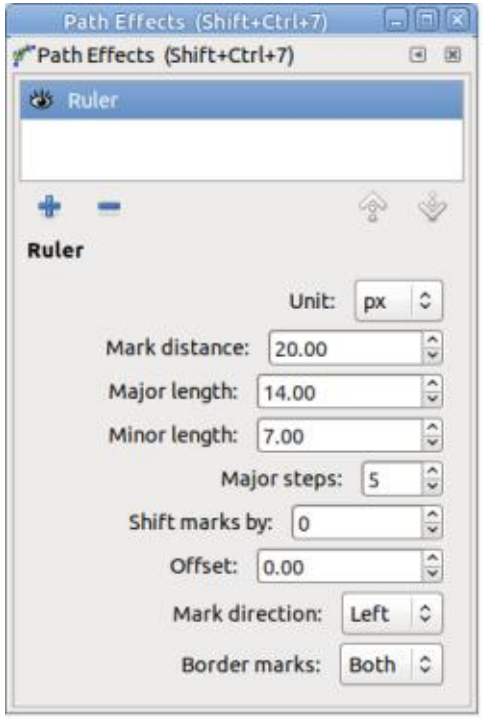
them red on the screenshot above to make them stand out a little, but Inkscape itself presents them in the same color as the other icons. The effects appear in alphabetical order, with no means to sort them; I would prefer an option to group all the experimental effects at the end of the list, perhaps with a divider, so that it becomes more practical to leave this option enabled without them cluttering up the list of "safe" effects.

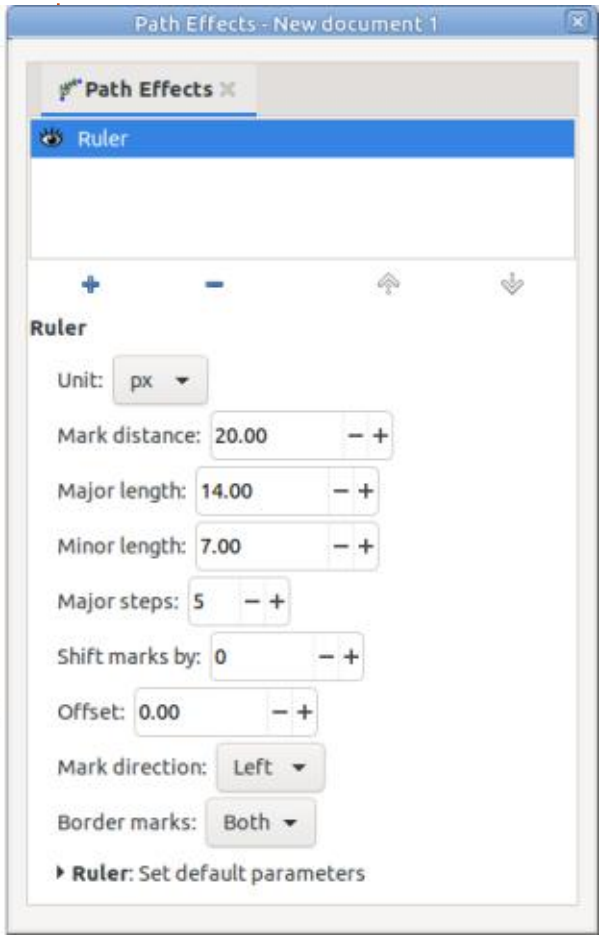
As you might expect, using any of these experimental effects is entirely at your own risk. Don't be surprised if doing so results in crashes, and even if they appear to work fine there's no guarantee that your files will continue to be compatible with future versions of Inkscape. For this reason I don't intend to delve into these in any detail until they are promoted to supported effects in future – though I won't rule out a quick overview if I run out of other topics to write about before the next release!

One other thing to notice from the previous screenshot is that there are two effects which are disabled: "Power clip" and "Power mask". These require that there's already a clip (or mask) on the path that you're adding the effect to. When a suitably clipped/masked element is selected, these will also be enabled alongside all the other LPEs.

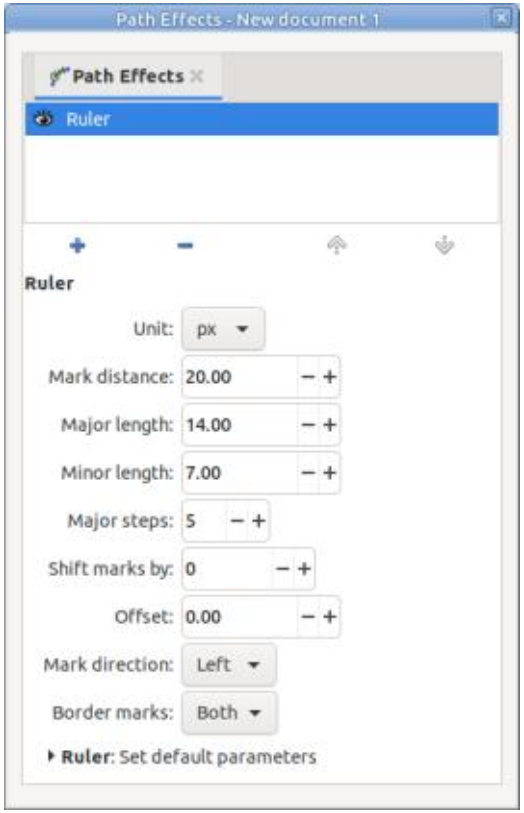
The UI changes aren't limited to the "Add Effect" dialog. Once an effect has been added to the chain, the corresponding parameters section of the main LPE dialog will also show some additional options. This can be seen with the Ruler LPE, for example, as shown in this comparison between v0.92 and v1.1.1 (below).

The height difference between the dialogs can be explained by the "tab" at the top of the panel, which can be used to dock it in v1.x, combined with the generally larger input fields used throughout the UI in newer releases. The parameter rows themselves have also changed from right-aligned to left-aligned. Of the two, my personal preference is for the older style where at least the +/- buttons are vertically aligned. In reality, however, I don't really like either approach. A better

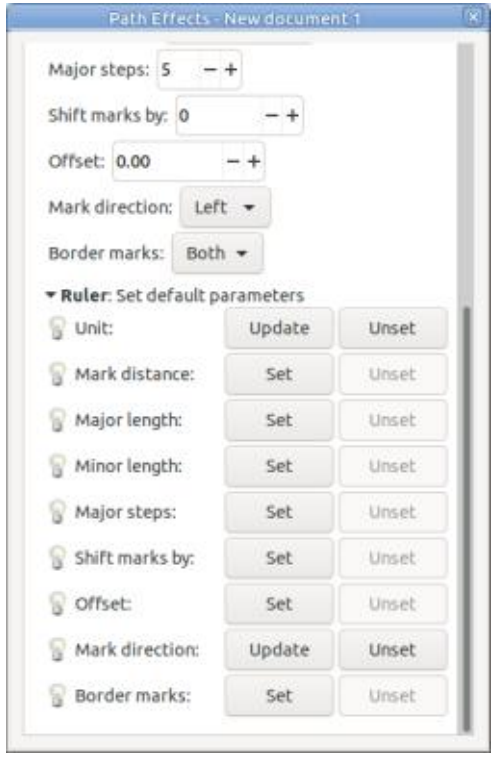




option, in my opinion, would be a more tabular style in which the labels fall neatly into one column and the input fields into another, as in the following mock-up. I am aware, however, that this may not be possible to achieve with the current widget toolkit, so consider this wishful thinking rather than a serious proposal.



One new feature that is common to all the LPE parameter screens is the "Set default parameters" section at the bottom of the dialog. Expanding this will display a list of Set (or Update) and Unset buttons, one pair for each parameter the LPE offers. These allow you to set the default values that will be used when the LPE is first added to a path, by entering the value into the parameter field and clicking the corresponding Set button. The button label will then change to Update, allowing you to modify the stored default by changing the value in the field before clicking the button. The Unset button will clear your saved preference and revert to using the LPE's own default values.



For a little more information about any parameter, hover the mouse over the lightbulb icon at the left of the row: a pop-up will display the name of the field, any tooltip associated with it, and the default value – or the value override, if you've set one. This can be useful for confirming the value that is currently being used, but it would be nice if it still showed the system default when a custom value is set, to give the user a bit more information about what will happen if they click the Unset button.

As well as this new set of buttons, you'll find that many of the long-standing LPEs have gained a few additional parameters. I don't intend to revisit these at this time, as the changes are generally small enough not to present either a problem or a significant opportunity. From next month, however, I will start to take a detailed look at the completely new LPEs that have been added.

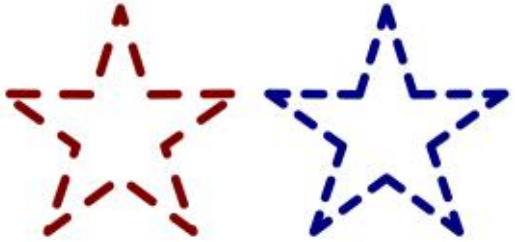
Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

Inkscape - Part 117

Over the next few instalments I'll be looking at the new Live Path Effects (LPEs) that have been added with Inkscape 1.0 and 1.1. Earlier articles in this series provide a general introduction to LPEs and what they are (part 42), as well as a deeper dive into the LPEs that arrived with earlier Inkscape releases (parts 42 to 47, 65 to 69). The previous instalment detailed the more general changes to the LPE dialog that took place with version 1.0, but this month I'm going to look at the first of the new LPEs, plus an old one that has had something of a rebirth.

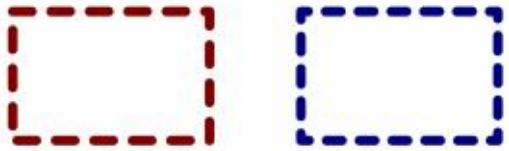
Dashed Stroke

At first glance there doesn't appear to be an obvious need for an LPE that renders the stroke of an object as dashes. After all, the Fill & Stroke dialog already offers various dash patterns which form part of the native SVG format (remember, LPEs are an Inkscape-specific extension). But although the standard SVG dashes are often sufficient, they do lack some nuance in the way they're distributed along a path which can give a less than aesthetically pleasing result. This is the niche that this new LPE aims to address. As a quick example, look at these stroked stars, the red one on the left using standard SVG dashes, and the blue one on the right using the Dashed Stroke LPE.



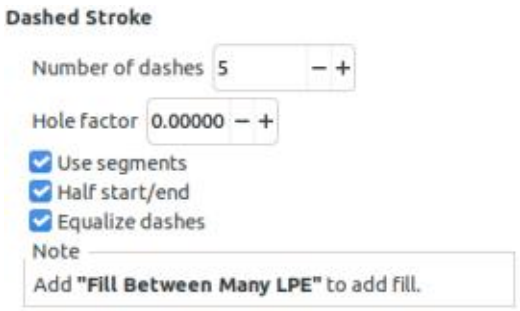
Pay close attention to the way that the corners – both concave and convex – are rendered. The SVG version is mirror-symmetric along the vertical axis, but only because I adjusted the dash offset to give that effect. Without that manual intervention there was no symmetry to the dashes at all. Even with that change, however, the corners differ as you move around the star: for the tips of the shape we would probably like them all to look like the point at the top, and not like the remaining four. None of the inner corners are really what we would like in most cases. Notice that the LPE version achieves exactly the right look, with the corners all appearing pleasingly similar and symmetric.

Let's look at another example. Dashed outlines are commonly used around simple rectangular boxes in flow charts and other diagrams. Which do you think looks better: the red SVG version (left) or the blue LPE version (right)? Once again, focus on the corners.



The reason for this difference is that the SVG stroking spec uses a very simple algorithm to determine how to draw the lines and spaces. It simply starts at the beginning of the path and draws a series of alternating dashes and spaces, based on the pattern described in the stroke-dasharray attribute or CSS property. It doesn't care about curves or corners, it just plods along from start to finish rendering a repeated series of strokes and dashes, regardless of the underlying shape. You can control the position of the first dash, using the stroke-dashoffset attribute or property (which is exposed via the Fill & Stroke dialog in Inkscape), but all that does is shift the entire pattern along. It doesn't affect the length of each stroke or space, so you are still likely to end up with unbalanced dashes as they wrap around the corners of your shape.

The LPE, on the other hand, works a little differently. The biggest change is that it can work on each segment of a path individually, rather than treating the entire path as a single stretch to be dashed as one. This is the secret of those better looking corners – drawing half a dash at each end of a segment results in dashes that are pleasingly symmetric as the path turns a corner. Let's look at the options available in this LPE, and the settings I used for the blue rectangle.



I'm going to describe these parameters out of order, as this is the best way to explain what each option does. I'll start with the "Use segments" parameter: when unchecked this results in the other parameters applying to the entire path (much like the native SVG dashes). When checked, each segment of a path is treated separately. In most cases you'll probably want this checked.

Going back to the top, the "Number of dashes" parameter defines the number of dashes that will be rendered along the length of the whole path, or along each individual segment. But the actual count will also depend on the "Equalize dashes" option, as we'll see shortly. This parameter is at the heart of the fundamental difference with the LPE dashes, though: SVG dashes don't have a count or limit, they'll simply keep rendering as long as there is any path left to fill; the LPE dashes, on the other hand, aim to fit a specific number of dashes into each path or segment, subdividing the available length according to this parameter and then distributing the dashes and spaces evenly within.

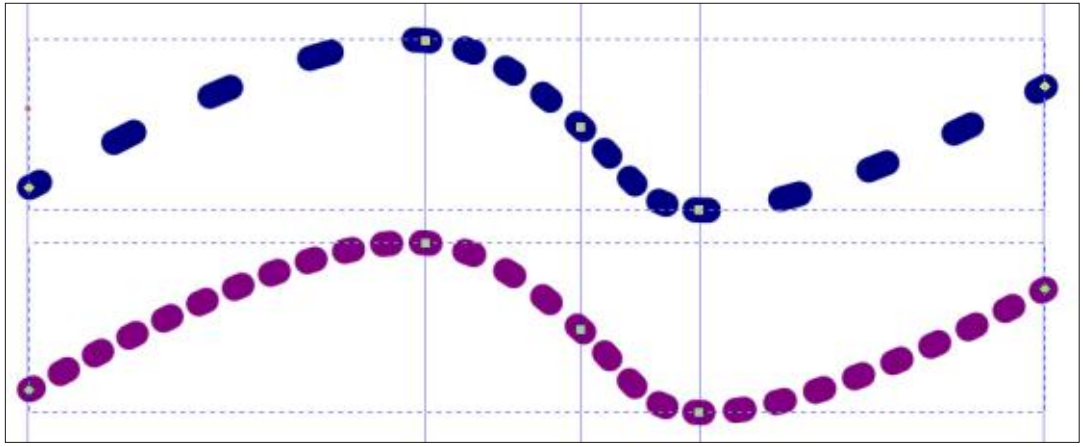
The relative lengths of the dashes and spaces can be adjusted using the "Hole factor". Leave it at zero to have them the same size, increase it (up to +0.99999) to increase the size of the dashes and reduce the spaces, or decrease it (down to -0.99999) to adjust the balance in the opposite direction. Reducing it to its lowest value results in each dash appearing as nothing more than a pair of line caps, as set in the Fill & Stroke dialog: a circle (for round caps) or a square (for square caps). Watch out if you use the "Butt cap" style, however, as that effectively causes the dashes to disappear completely at the lowest hole factor. Note, however, that using a single ratio like this means that the LPE can't produce the sort of dash and dot combinations that the stroke-dasharray allows for in normal SVG strokes.

The "Half start/end" parameter determines whether to only draw half a dash as the start and end shapes (checked), or to draw a full dash at the start and end if possible (unchecked). Usually this is best left checked in order to gain the aesthetic benefits of symmetry and even spacing. Each half dash still contributes to the "Number of dashes" count, so a count of 5 with this parameter enabled actually means 3 whole dashes and two half dashes, rather than the 4 whole dashes (plus two halves) that you might expect if you were adding the parts up numerically.

Finally, the "Equalize dashes" parameter has the potential to upend the "Number of dashes" count entirely. When this is checked, the algorithm first creates the desired number of dashes for the shortest segment in the path. The length of each dash in that segment is then used when rendering all the other segments, fitting more than the actual count in if there is space. A demonstration might make this a little clearer.

In the image below, the two paths are the same, but the top one has "Equalize dashes" unchecked, whereas the lower one has it checked. I've positioned some vertical guides to make it clearer where the nodes of the path are – i.e. where each segment starts and ends. The top path honours the "Number of dashes" count completely: each segment has 5 dashes (3 whole, 2 half). But this leads to different spacing between the dashes across the segments, and even differently sized dashes in the middle two segments where they've had to be squeezed into a smaller space.

The lower path, on the other hand, clearly shows that all the



dashes and spaces are even across all the segments. But it does this at the expense of the "Number of dashes" value. That parameter is used when calculating the smallest segment (the third one), but then the resultant dash and space size is simply used for all the other segments, regardless of the count. As you can see, the end result looks better, and is probably what you are likely to want, but the first and last segments have way more than 5 dashes each.

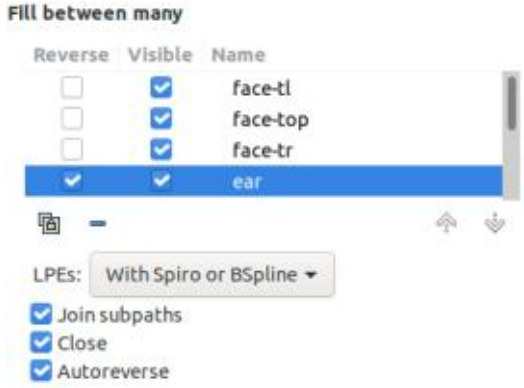
There's one additional part of the UI in the screenshot from the LPE dialog: not another parameter, but a note in a box, which says 'Add "Fill Between Many LPE" to add fill'. What on earth does that mean, and why is it necessary?

Fill Between Many

Remember that the output from an LPE is just an SVG path, so all the clever things that LPEs can do must ultimately be rendered using normal SVG capabilities. As we've already seen, raw SVG can't produce the sort of dashes that we're getting from the Dashed Stroke LPE, so what actually are we seeing in our rendered output?

The result is actually a new complex path, made up of lots of individual subpaths, one for each visible dash. Trying to add a fill to this will actually just fill the subpaths, not the whole shape. Because most of the subpaths only have two nodes, even that fill isn't generally visible. The exception is the corners, where three nodes are used in a triangular configuration. Sure enough, adding a fill to a Dashed Stroke path does result in a web of colour in the corners, but not the filled shape we're looking for. As an example here's our star from earlier, but with the stroke width reduced for clarity and an orange fill applied.

This has long been an issue for many LPEs, not just the Dashed Stroke, so the Inkscape developers addressed it head-on a long time ago, by adding the "Fill Between Many" LPE back in version 0.92. I covered this LPE in some detail back in part 67 (FCM issue #127), though the UI has expanded a little since then. In older versions you only had the ability to add paths to the LPE, flagging some of them as needing to be reversed. The new UI, when used with the same "Frankie" image I used in part 67, looks like this.



The basic functionality remains the same: you have to create a sacrificial path on which to apply this LPE, then add each of your source paths by copying each one to the clipboard and adding it to the list in the LPE, as described in the earlier article. It can be a time-consuming and difficult process when dealing with lots of paths, though it's not too bad for adding a fill to a shape with the Dashed Stroke LPE as there's only one path to add in that case. These are the steps needed to add a fill to our rectangle, for example:

- Draw a sacrificial path (usually just a simple two-node line)
- Add the Fill Between Many LPE to the sacrificial path
- Select the path which has the Dashed Stroke LPE applied (the rectangle) and copy it to the clipboard
- Re-select the sacrificial path in order to bring up the UI for the Fill Between Many LPE
- Click the "Link to path in clipboard" button to add the Dashed Stroke path to the list
- Adjust the fill and stroke values to suit your needs

With luck you'll now find that your rectangle has a fill, but things don't always go so smoothly. In my own experiments, trying those steps with a star rather than a rectangle results in either no fill, or an oversized fill object that is wrongly positioned and can't be moved. There are definitely some bugs in this LPE that have yet to be ironed out.

Compared with v0.92, the newer version of this LPE also provides some additional parameters to tweak. There is a "Visible" checkbox

for each path, allowing you to temporarily remove it from the filled shape, perhaps to test whether or not it is contributing anything useful prior to completely removing it from the list. The "Join subpaths" checkbox lets you fill each subpath individually (unchecked), or use the older behaviour of joining the subpaths to create a single shape to fill (checked). The latter is almost always going to be what you want. Another checkbox ("Close") now lets you leave the new path unclosed between the first and last paths in the list – probably more useful if you are using this LPE to add an extra stroke rather than a fill and, again, usually something you would want to leave checked. Finally the "Autoreverse" option overrides the individual "Reverse" checkboxes for each path: with this checked the algorithm will try to join paths based on the proximity of their endpoints, rather than strictly following the direction of each path. Usually this does a good job, and is best left checked, but you do have the option to turn this off and manage path reversal on a per-entry basis as before, should you wish to.

The pop-up menu is also a new addition, choosing how the source paths should be interpreted. Usually leaving this as "With Spiro or BSpline" is a good option: this will essentially use the shape you originally drew, whether it was created using simple SVG paths, or you used the Spiro or BSpline options that Inkscape exposes in some drawing tools. In practice these are implemented as LPEs, so this option tells Inkscape to use the output from those LPEs as the source, if they exist, or to use the plain path data otherwise. Alternatively you can select "Without LPEs" to only use the original path data, regardless of any LPEs applied. Conversely the "With all LPEs" option will use the path data that comes out of whatever series of LPEs has been applied to the shape. Be aware that this can quickly lead to very complex shapes if you're not careful, so isn't often the choice you want.

Looking back at the number of steps needed to add a fill to a path with the Dashed Stroke LPE applied, you may feel it's not worth the extra effort and confusion, preferring to stick to SVG dashes or to draw the fill as a separate object. The "Fill Between Many" LPE can certainly be a tricky feature to get your head around, and in other use cases where you need to add multiple paths to the dialog it can be a time consuming pain. Luckily the Inkscape developers have realised that this complexity gets in the way of an otherwise useful feature, so with version 1.1 they've added a new menu entry, Path > Fill between paths, which will silently create a sacrificial path and add the "Fill Between Many" LPE to it, already populated with any paths from your drawing that were selected at the time. This makes it trivial to use this LPE in most cases: just select the path or paths that need to be filled and select the menu option. You can then select the newly added fill in order to access the LPE parameters if you need to (e.g. to reverse specific paths).

Note that the sacrificial path added by Inkscape is of zero length: its "inkscape:original-d" attribute just consists of an "M 0,0" command, which doesn't actually draw anything. As such, be careful not to either hide the LPEs visibility, or that of all its listed paths, otherwise you won't be able to select it on the canvas. In that case you'll have to find it in the XML editor (look for a path with that "M 0,0" value) in order to select it for further editing or deletion.

This new menu entry is a great addition for working with LPEs, as it helps to get around one of the most fundamental problems most users will come across as they begin to use them. For this reason alone it may be worth upgrading to version 1.1.x if you haven't already done so. It's a shame, however, that the "Fill Between Many" LPE, even when added using this menu entry, can still be rather buggy, even for simple examples. Hopefully future releases will make it more robust, which will help to make LPEs in general a far more useful tool than they already are.

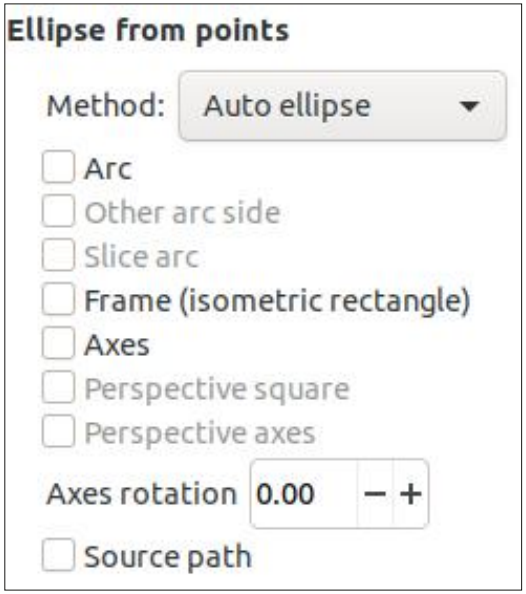
Mark uses Inkscape to create comics for the web (www.peppertop.com/) as well as for print. You can follow him on Twitter for more comic and Inkscape content: **@PeppertopComics**

Inkscape - Part 118

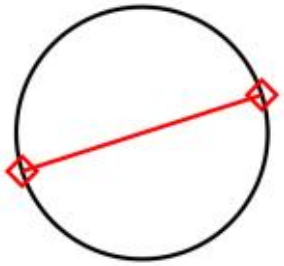
This month, I'll be continuing to look at the new Live Path Effects (LPEs) that were added in Inkscape 1.0.x and 1.1.x.

Ellipse from points

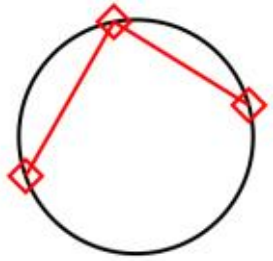
If you're a frequent user of LPEs, you may already be familiar with the "Ellipse by 5 points" effect (covered in part 69 of this series). As the name suggests, this draws an ellipse that passes through the first five nodes of a path. This new LPE does the same thing, and much, much more. In fact, the name really doesn't do justice to the capabilities it offers, as it not only allows for the creation of ellipses, but also circles, arcs, and segments. Whereas the old LPE provides no parameters to control its output, this new one comes with quite a few, not all of which are enabled at the same time.



Despite all these extra controls, however, the basic functionality is still pretty intuitive, and benefits hugely from applying any changes to the parameters or path shape interactively, making it fun to play around with all the different options. Your starting point will always be a path to which the LPE is applied, although this effect cares about only the positions of the nodes, not the shape of the path segments. For demonstration purposes, however, all my examples will use straight line segments, and I'll show the original path as a red line with diamonds marking the nodes (courtesy of the "Clone original" and "Show handles" LPEs). The black lines are the output from the LPE. Let's start with the simplest case: a two-node line using the "Auto ellipse" method.



In this case, the effect draws a circle using the two nodes in the path as points at either end of the circle's diameter. Drag one of the nodes around, and the circle will scale and rotate accordingly. Let's see what happens if our source path has three nodes, rather than two.



Again we have a circle, but this time it circumscribes the triangle created by the three nodes. Once more, dragging the nodes around the page will give you a good idea of how the size and position of the circle relates to the node positions.

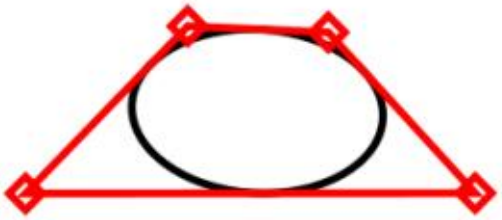
With three nodes, some of the LPE parameters start to become useful to us. When enabled, the "Arc" checkbox draws an arc connecting the three nodes, rather than closing the whole circle. Enabling the "Other arc side" checkbox instead draws the "other" arc which forms the remainder of the original circle. "Slice arc" can be used with either type of arc to render it as a segment (i.e. a pie-chart "slice") rather than an arc, by adding straight path segments that join the end nodes to the center of the circle.

With three nodes, more of the options in the "Method" pop-up menu will also work. The first two ("Auto ellipse" and "Force circle") just produce the result we've already seen. "Isometric circle" treats the path as having straight line segments, even if it hasn't, and uses the first two segments to define the edges of an isometric rectangle into which it fits an ellipse that appears as if a circle is rendered in that isometric

projection. That sounds complex, but if you do much work with isometric or oblique projections, you'll know exactly what this is for: in short, draw your path with suitable angles (e.g. 120° for isometric, 135° for oblique), and it will render "circles" that are appropriately distorted for the projection.

The last two options in the pop-up are thankfully easier to describe: "Steiner ellipse" draws an ellipse that circumscribes the triangle created by the three nodes, while "Steiner inellipse" draws one that inscribes it. The image below shows the "Isometric circle" output, followed by the two ellipses, for the same path that I used earlier.

Adding a fourth node to our path is required for the remaining entry in the pop-up: "Perspective circle". This treats your four nodes as defining a square in a perspective view, and renders a "circle" that fits within that square. This is perhaps most clearly demonstrated using a closed path arranged to give a classic perspective view.

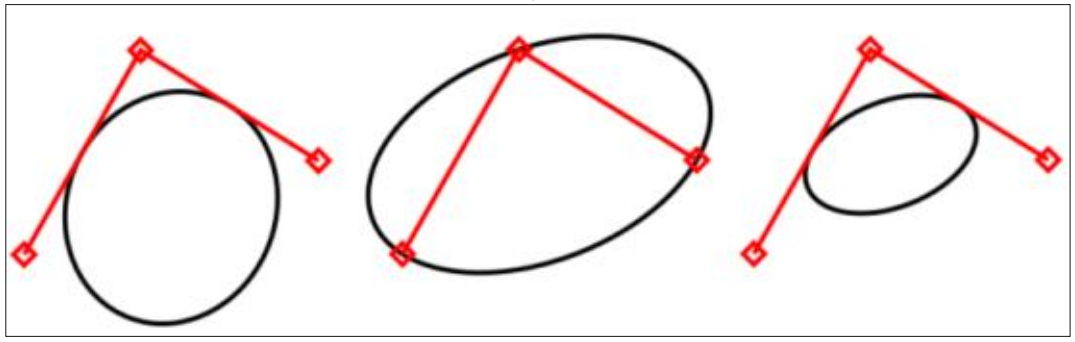


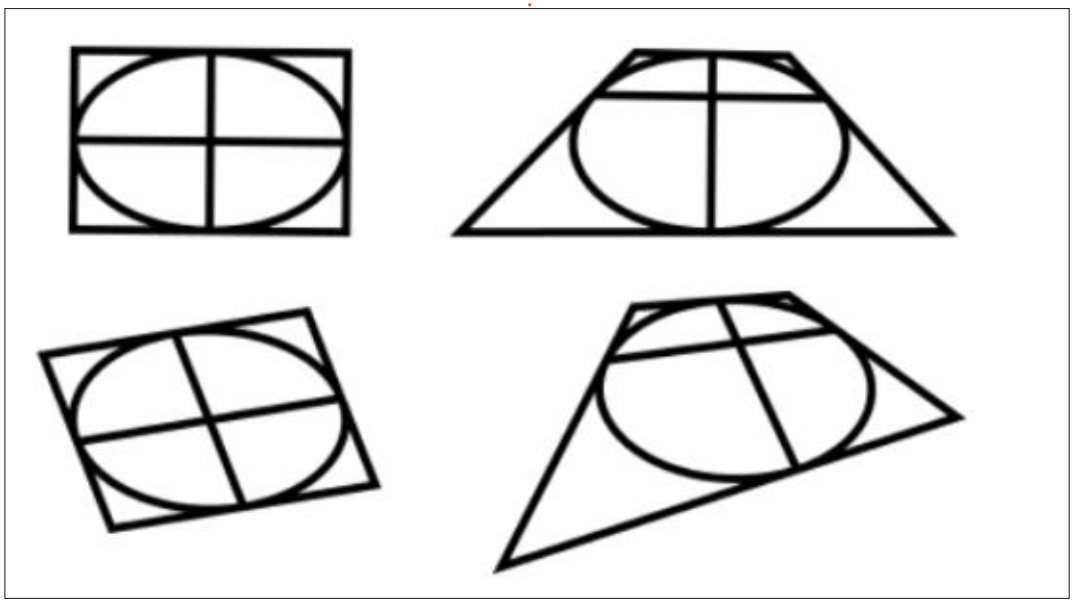
With the red lines removed, we can now also see what the remaining checkboxes do. The "Frame (isometric rectangle)" option will draw a bounding box around your circle or ellipse. By default this will be a rectangle, defined by the size of the major and minor axes of the ellipse, but you can use the "Axes rotation" spinbox to rotate the box around the ellipse, resulting in it becoming a parallelogram if the ellipse's axes aren't perfectly aligned with the global x and y axes. The "Axes" checkbox simply adds two lines, joining the mid-points of opposite sides of that bounding box in order to divide it into four equal areas.

When the "Perspective circle" method is used, however, two alternative options become available. The "Perspective square" checkbox draws lines marking the "square" in perspective space that the "circle" is inscribing: essentially this draws a shape connecting all of the four nodes, even if the original path wasn't a closed shape. The "Perspective axes" renders a pair of axes as they would appear in perspective, leading to a rather different outcome compared with the plain "Axes" option, especially if you rotate them using the spinbox.

The image below compares the two types of bounding box and axes when used on the same perspective circle. The left shows the result of the "Frame" and "Axes" checkboxes, while the images on the right show the corresponding "Perspective" versions. In both cases, the top image shows a rotation of 0° whereas the bottom image shows the result of increasing that value to 15°.

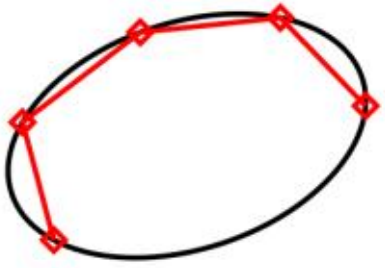
To complete our tour of the checkboxes, the "Source path" option does what you might





expect: it renders a copy of the original source path as part of the output. Due to the nature of LPEs, the source path is drawn in the same style as the ellipse (and axes and bounding box, if used), so if you want it to appear differently – as I did in my examples – you'll need to use a "Clone original" or "Fill between many" LPE on a sacrificial path in order to render it as a different object for styling purposes. For general use, however, enabling this option can make it a lot easier to see what's going on with your ellipse as you interactively tweak it, even if you then turn it off again once things are positioned correctly.

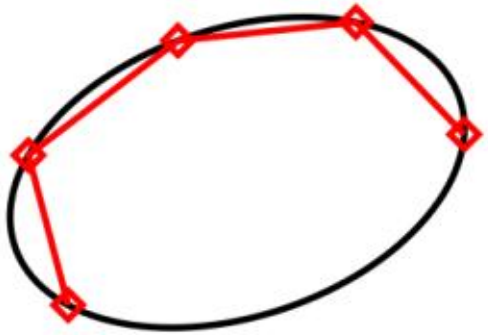
Lastly, if we add a fifth node to our path (with Method set to "Auto ellipse") we end up with the same result as the old LPE: an ellipse that circumscribes the five nodes. If you want full control over your ellipse, this is probably a better option than either of the Steiner methods.



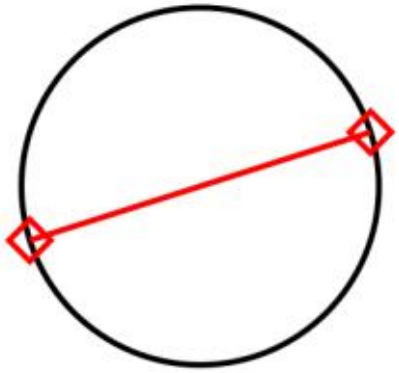
There's one final thing to mention regarding this LPE: the developers should be commended for putting the effort in to produce really useful tooltips. For example, if you can't remember how many nodes your path needs for each different method, just hover your mouse over the pop-up for a useful reminder.

Offset

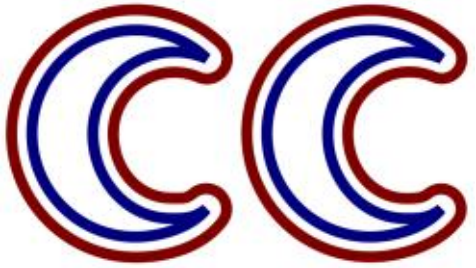
The Offset LPE is pretty straightforward, and does what its name suggests. You may be familiar with the Path > Dynamic Offset feature which puts a handle on your path that you can drag to adjust the amount of offset, letting you create



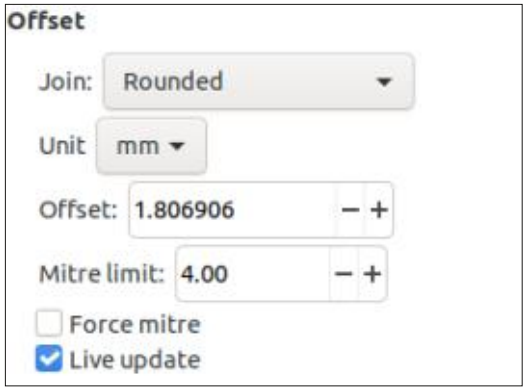
a shape that insets or outsets the original path. In doing so, it modifies the original, unlike the Path > Linked Offset feature that creates a second path which maintains a live linkage to the original. The LPE falls somewhere between these two: there is a live link to the original path shape, but that path is not included in the final output, so, despite this link, the result still leaves you with only one path rather than two. In practice, therefore, this LPE is closer to the Dynamic Offset feature, only with more options.



Let's look at an example. Here I've created a crescent shape by performing a Boolean difference operation between two circles. I've also adjusted the nodes of the bottom point very slightly in order to demonstrate some aspects of the LPE later on. In both these cases I've made a copy of the original shape in blue, but applied an offset to generate the red version. The left-hand image shows the result of the Path > Dynamic Offset feature, while the right-hand version shows the LPE equivalent. As you can see, they look identical.



If that was all there was to this LPE, it might still be useful as part of an effect chain, but not so much as an effect in its own right. But once we consider the various settings that it offers, it quickly becomes clear that the LPE offset is a far more powerful beast than what went before it. Let's look at the available parameters.

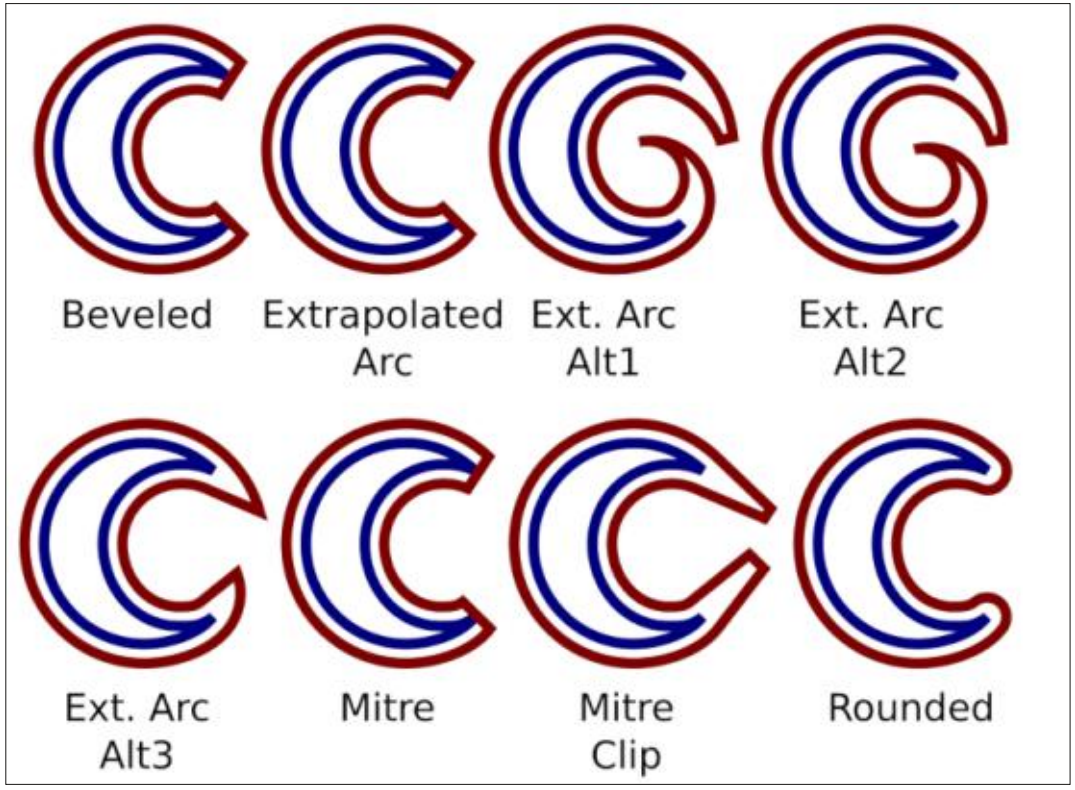


Dealing with these out of order, the "Unit" pop-up should be pretty self-explanatory, setting the type

of units used for the "Offset" parameter which, in turn, is used to set the amount of offset that is applied to the path. It can be a positive value for an outset, or a negative value for an inset – but, in practice, it's usually more effective to switch to the Node tool (F2) and drag the small, red, circular handle on the canvas, to adjust the offset by eye. The "Force update" checkbox determines whether or not the path updates live as you drag the handle, or updates only when you release the mouse button. Usually you should leave this enabled, unless you have a slow machine or a complex path which makes the updates jerk and stutter.

The "Join" pop-up has the most effect on the shape of the path. In the previous image it was deliberately set to "Rounded" to reproduce the effect of the Dynamic Offset feature, but here's a demonstration of how each entry appears with this particular shape.

It's worth noting that the result you'll see is extremely dependent on the shape of your source path. In particular, look at the difference between the two pointed corners in the extrapolated joins, after making only minor tweaks to the nodes of the bottom point. As this shows, tight corners are a particular issue and increasing the "Mitre limit" value will allow some corners to appear that would otherwise be cut off. In the previous examples, increasing this value to 10, for example, allows most of the shapes to extend to give far more pointed ends. The main exceptions to this rule are the Beveled and Rounded types, which don't take the mitre limit into account. The best option is usually to try each join type, and adjust the mitre limit and/or the individual nodes to get the result you want.



The Extrapolated Arc join types are particularly interesting. These try to follow the curves of your path to form a more natural join, rather than just projecting straight lines as a mitre does. When working with curved paths, these are well worth trying. If, however, you really want to project the pointiest of mitred corners regardless of the mitre limit, choose any join type other than Beveled or Rounded, and check the "Force mitre" option.

Finally, it's worth noting that this LPE also works with open paths, whereas the Dynamic Offset feature automatically closes them when you try to use it.

These two LPEs both offer features that are head-and-shoulders above the options that Inkscape provided previously, and the developers should be applauded for continuing to push the boundaries of what path effects are capable of.

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**

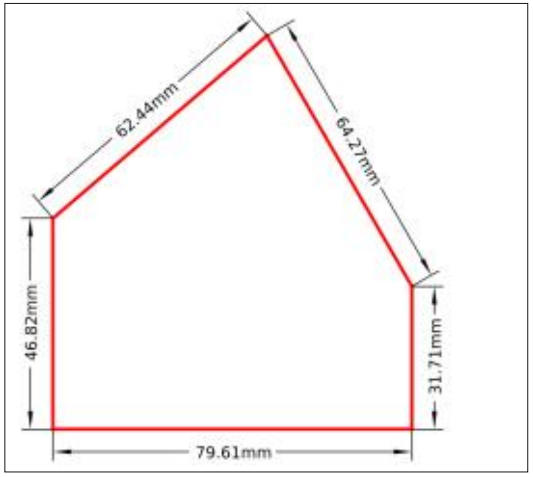
Inkscape - Part 119

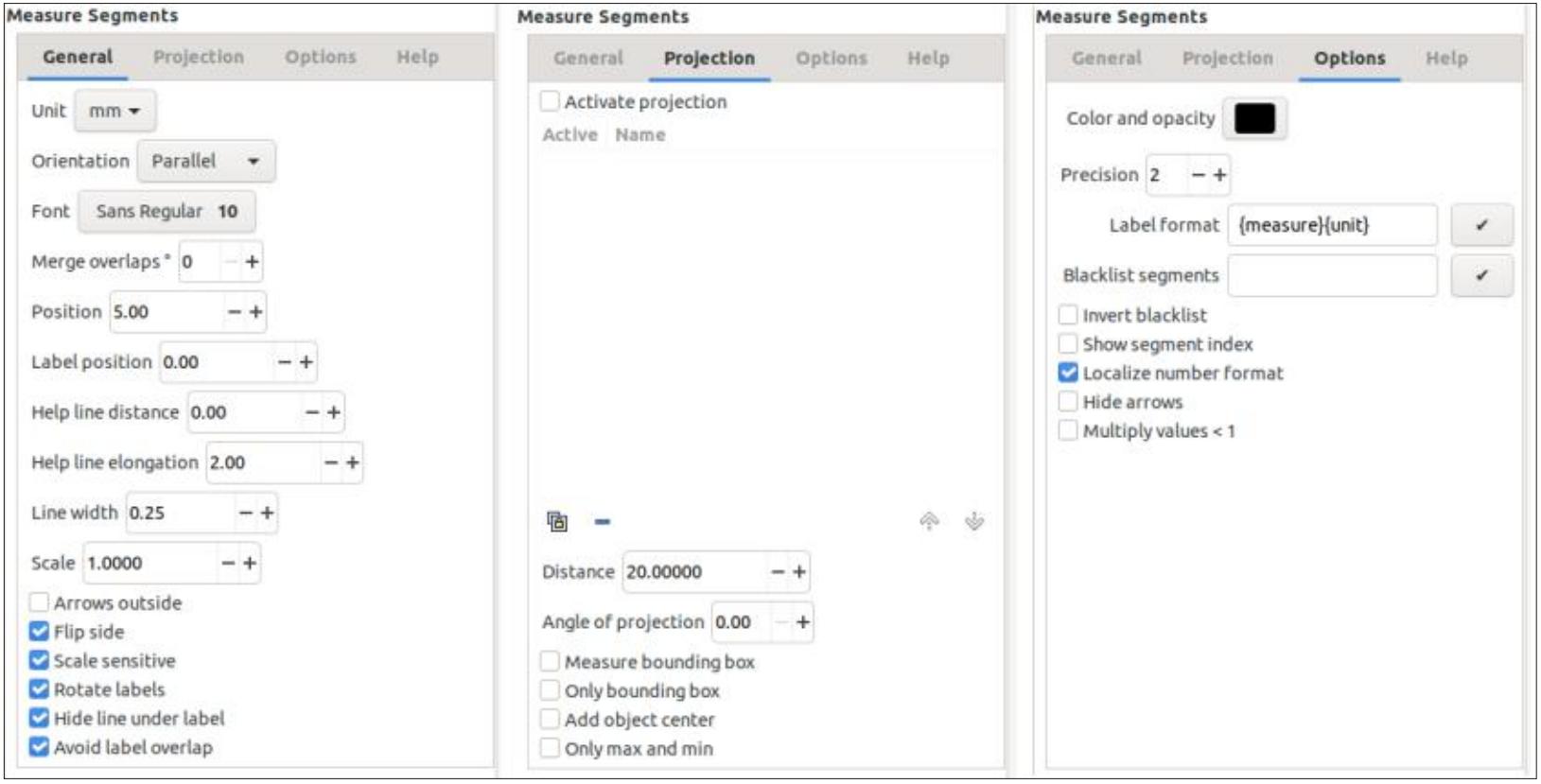
They did it again! Literally the day after the deadline for last month's article, the Inkscape developers released a new version. It's only a maintenance and bugfix release (version 1.1.2) with no new features, so I won't be covering it in any more detail, but if you've already upgraded to the 1.1 series, it's probably worth installing this latest version for improved stability. Alongside this release was an alpha version of Inkscape 1.2. If you have the time and inclination, I do recommend giving this release a try and reporting any issues you find, particularly in the new features. The more that users report problems with the alpha and beta releases, the more stable the final release is likely to be. Information about both these releases can be found on the official Inkscape news page: https://inkscape.org/news/

This month, I'll be looking at just one Live Path Effect (LPE) that was added in version 1.0: Measure Segments. This LPE operates so differently to most that I won't have enough space to cover everything in one article. This time, I'll look at the practical functionality of the effect, and next month, I'll dig into some of the more technical details of how it works.

Measure Segments

At first glance, the Measure Segments LPE seems fairly straightforward – albeit with a lot of parameters to consider. It measures the straight line distance between the nodes of your path, and annotates your image with those dimensions. This is definitely a case where a picture is worth a thousand words: in this image my original path is the red line, and the rest of the content has been generated by the LPE.





The most interesting thing about this image is that I was able to color my original path in red without the need to use the Clone Original LPE, or any equivalent technique or workaround. The styling of the original path is distinct from the styling of the dimensions. Long-term readers of this column will know that this flies in the face of everything we know about LPEs. Historically, the output from an LPE has been a single path – albeit often a very complex one – meaning that all the different parts of an effect would have to adopt the same styling. Clearly there's something very different going on in this case, but I'll get back to that next month.

For now, let's just take the effect at face value, and have a look at some of the parameters we can tweak in order to adjust its output. For this LPE, there are so many that they have been split across three tabs – plus a fourth "Help" tab that doesn't really provide any more information than can be gleaned from the tooltips.

Starting with the General tab, the Unit pop-up is pretty self-explanatory. It's restricted to the units that Inkscape uses throughout the software, so if you're trying to create a scale drawing in miles or microns you might think you're out of luck. In practice the format of the numeric labels is defined over on the Options tab, so you can get around this limitation by replacing the "{unit}" placeholder with a fixed string of your own. For example you could set the Label Format field to "{measure} miles" or "{measure} µm" to mark the dimensions in units that Inkscape doesn't support.

Back on the General tab, there is another field that should go hand-in-hand with the Unit pop-up, but which has been counter-intuitively put towards the bottom of the dialog: the Scale field. This acts as a multiplier for all the numeric values, so if your original drawing is half-size, you should set this to 2; conversely if your drawing is double-size, set it to 0.5, and so on. You can even enter a negative value here, though I'm not sure why you would want to.

Most of the remaining fields on this tab simply adjust the specific appearance of the dimension lines and labels. You can use the Font pop-up to change the font and size; the Position field to adjust how far away the dimension line is from the path being measured; the Flip Side checkbox to select which side of the path the dimension is drawn; the Label Position field to change the position of the numeric value relative to the dimension line, allowing it to sit on top of or below the line (in which case you might also want to uncheck the "Hide line under label" option). To be honest, the best approach is just to play around with these fields in order to see what they do – using the tooltips if you need a hint – but in my experience the default values tend to give pretty good results.

The one widget that I don't understand is the Merge Overlaps field. No matter how I draw my paths, whether with long or short segments, or with tight or wide angles, I can't get this field to have an effect. If anyone can provide some insight into what this field does, please let me know.

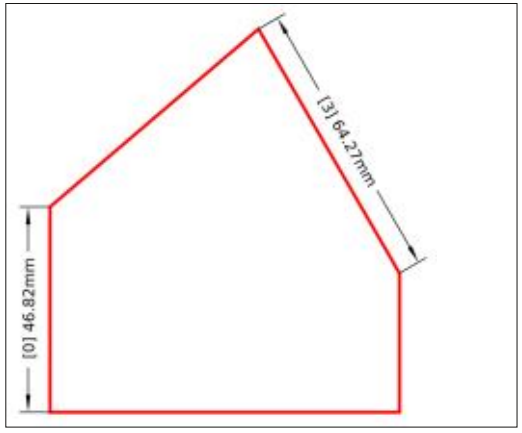
I'm going to skip the Projection tab for now, and go straight to Options. This is a mish-mash of fields, some of which would seem to relate closely to those in the General tab, but which have been hidden away here instead. The Color and Opacity picker, for example, seems to me to be a sensible companion to the font picker. This widget sets a single color that is used not only for the dimension lines, but also the leader lines ("Help lines" as they are named in this extension), and the dimension text.

In the same vein, the Precision and Label Format fields surely deserve to live alongside the Unit and Scale widgets. Note that the Precision field just sets the number of decimal places in the text labels. There's no way to use significant figures rather than decimal places, and there's no facility for engineering or scientific notation.

Similarly misplaced is the "Multiply values < 1" checkbox, which is used to better display very small values by multiplying them by 100 and omitting the auto-inserted units from the text (but it won't omit the units if you've used a fixed string in the Label Format field, as I described earlier). This should arguably also live with the Scale and Units fields. The Hide Arrows checkbox is just as deserving of a place on the General tab as most of the checkboxes at the bottom of that pane.

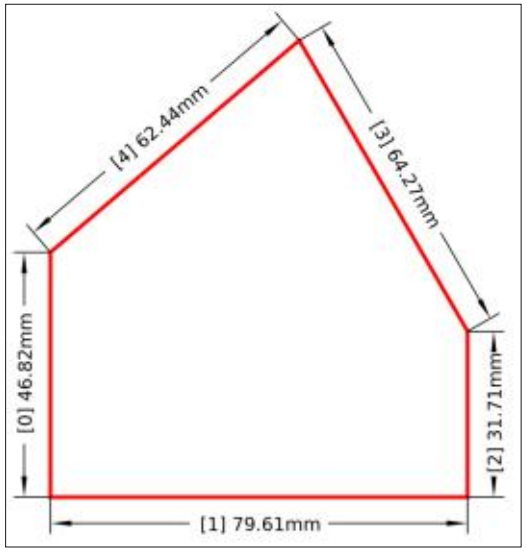
None of this arbitrary placement of widgets is a deal-breaker, but it's useful to be aware that the Options tab provides these features. Perhaps a later release will tidy up this effect's UI, and better group the controls into more logical sections.

Also on the Options tab are what I consider to be the "sneaky" controls: three fields that radically increase the capabilities of this effect, albeit at the expense of a little extra effort on the part of the user. The most important of these is the Blacklist Segments field, which takes a comma-separated list of numbers, and uses those values to suppress the output of the LPE for specific path segments. Here's how the earlier image looks when the string "1,2,4" is entered into this field and the tick button is clicked.



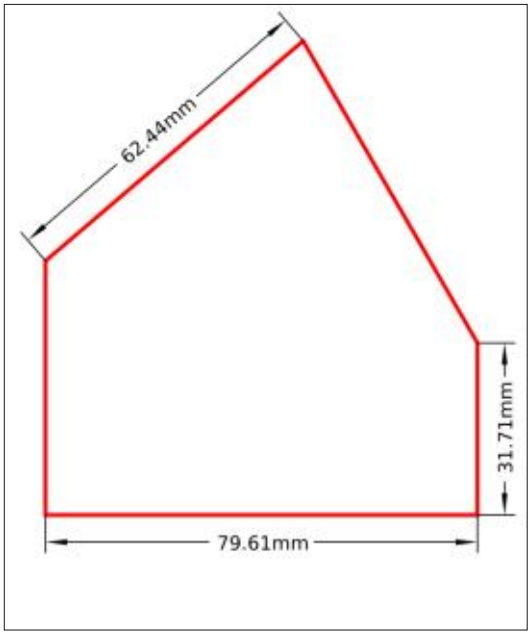
You can see that there are no longer any dimension lines rendered for three of the path segments. The three segments are not, however, the first, second and fourth ones in this path. Rather they are the second, third and fourth: the values start at zero for the first path segment in a classic example of a programmer exposing the internal indexes that the software uses, rather than adjusting them to be more user-friendly to the layman.

As you might imagine, working out which segment index you need to use to target a specific part of the path can quickly get tricky with complex shapes, but this LPE does offer a feature to help. Enabling the "Show segment index" checkbox will prefix each dimension with the segment's index, in square brackets. Be aware that it unfortunately doesn't show the index for any segments that are already listed in the Blacklist field, so you may want to enable this option first, while that list is still empty. With no blacklisted segments, and this checkbox enabled, you can see that indexes 1, 2 and 4 do indeed correspond to the omitted dimensions on the previous image.

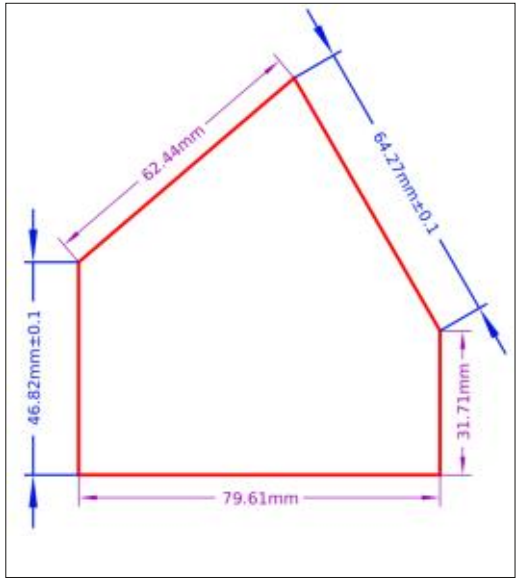


One thing to be very conscious of is that the segment indexes are based on the direction in which the path is drawn. If you use the Path > Reverse menu entry, you'll find that the indexes run in the opposite direction, probably requiring you to adjust the blacklist. Similarly, if you add or remove any nodes then the indexes of some of the segments will also change.

Due to the political sensitivity around certain terms used in computing, there's a good possibility that the "Blacklist segments" field might be renamed in future. Often "blocklist" is used instead, but I think in this case calling it "Skip segments" or something similar would give a better idea of its functionality. I'm mentioning this because of the last of the sneaky controls: Invert Blacklist. Checking this turns the blacklist into a whitelist – or, more descriptively, turns the "skip segments" list into a "draw these segments" list. This may save you having to enter a long list of segment indexes when you only want the LPE to render a small number of segments from a complex path. With this applied, you can see that my list of "1,2,4" actually results in only those segments being drawn, rather than those segments being omitted.



So why do I consider these three fields (Blacklist segments, Invert blacklist, Show segment index) to be "sneaky"? It's because they allow you to apply this same LPE to a path multiple times, each using different parameters, using the blacklist fields to ensure that each copy of the effect targets a different subset of the path segments. Perhaps you need to color-code different measurements, alter the label format for one segment, or simply move the position of some dimensions so that they don't clash with other parts of the image. Using these fields will let you achieve all that and more.



Here, for example, I've used two copies of the LPE. The first colors three of the dimensions in purple. The second uses the same blacklist, but with the Invert box checked, in order to target the remaining dimensions. These are then drawn in blue, with a thicker line width, arrows on the outside of the extension lines, and a tolerance value added by manually altering the Label Format field.

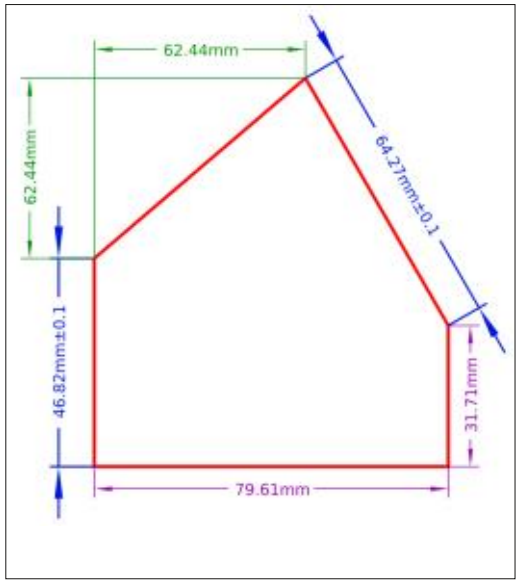
Now let's return to the Projection tab that we skipped earlier. I'll admit that this one has me a little stumped. When the "Activate projection" checkbox at the top is enabled, every node in your path is projected in an invisible straight line along the specified "Angle of projection", with the final dimension lines showing the distances between those projected lines. As you can see from this screenshot, however, the default behaviour may not be terribly useful, depending on the shape you're trying to measure, and the angle of projection you use.

There is a section in this tab to which you can add other objects, by copying them to the clipboard and using the Link to Item button, as happens in other LPEs. According to the tooltip, the nodes of those objects should then be projected onto your path in order to produce datum points for additional measurements. In practice, I haven't been able to get this feature to work at all, regardless of what I tried. If anyone has some insight into how to use this facility, please do get in touch.

The Blacklist Segments field can be used with projection mode, but the numbers in there are no longer the indexes of the path segments. Instead this allows you to turn off individual projected measurements. Once again the "Show segment index" checkbox can be used to identify the correct numbers to use. Also once again, this allows you to combine multiple copies of this LPE, with different projection angles, or mixing both projected and non-projected dimensions to produce the result that best suits your needs.

While the Projection tab might initially look useful, in cases where you only want to project dimensions along the horizontal or

vertical directions you may well find that the Orientation pop-up on the General tab lets you achieve a better result – though you're likely to need to use multiple instances of the LPE, each targeting specific segments, in order to get things exactly as you want them. Here's another copy of our dimensioned path, this time using the Orientation pop-up to produce the green dimensions at the top left. In this case I had to add two more instances of the LPE, one for each orientation, with both targeting the same single segment.



This really is a very powerful LPE, with a lot of flexibility built into it. It's a shame that there's not also a corresponding LPE for measuring angles, but perhaps that will come in future. Although this may give the impression of adding more CAD features into Inkscape, I see it more as a means of annotating simple sketches, rather than producing production ready designs. I've always maintained that Inkscape is a primarily artistic program, and if you want real CAD capabilities then you're better off learning to use FreeCAD or some other dedicated application.

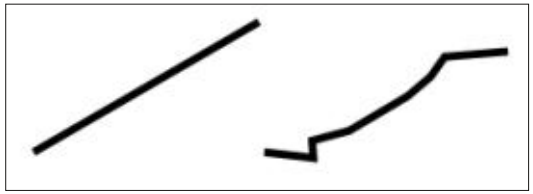
In this article, we've seen what this LPE is capable of – including different fonts and multiple colours that aren't the same as the source path. The way it achieves this is radically different to the way most LPEs work, and I'll be looking at the details of that – together with the problems it brings – next time.

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**

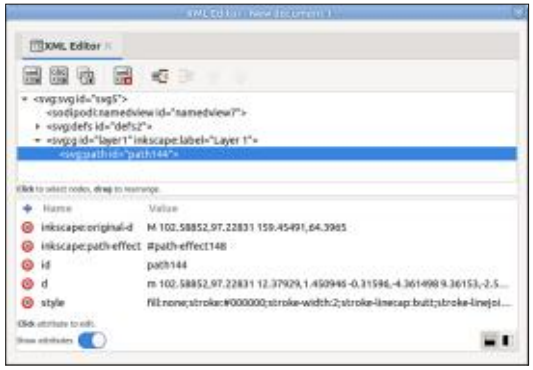
Inkscape - Part 119

This month doesn't just mark 15 years of Full Circle Magazine – I also celebrate a full decade of writing these Inkscape columns! Many thanks to everyone who has read any of them over the years, and I hope you've found them useful. One thing I've always tried to do is to explain the underlying reasons for some of the oddities and limitations in the way Inkscape operates, and this month is no different. Having described the operation of the Measure Segments LPE last time, in this instalment I'm going to look behind the curtain at how this effect differs quite radically from those that came before it. Please note, however, that this is just for information and education – you don't actually need anything in this instalment to simply use the LPE in the way it was intended.

First, a quick reminder of how Live Path Effects worked historically. An LPE was applied to a single path, and produced a single path as its output. The output path would replace the source path in the image. Here's a very simple example: the Roughen LPE, when applied to the two-node path on the left, produces the multi-node path on the right.

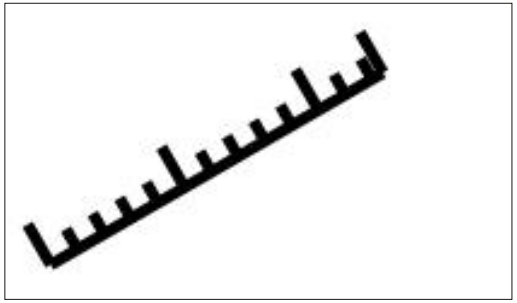


Looking at this in the XML editor, we can see that there's still only a single path object, but as well as a multi-node "d" attribute, it also contains an "original-d" attribute (in the "inkscape" namespace") which has only the two nodes of the original path.

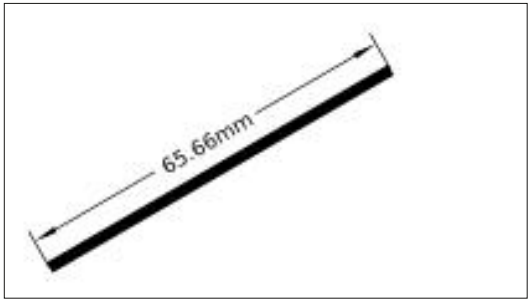


This is a pretty clever way to implement LPEs. Inkscape understands the extra attributes in its own namespace, so is able to treat the effect as a live, editable feature, while other SVG renderers, such as web browsers, will still show the result of the LPE since it's just a normal "d" attribute like you would find on any SVG path object.

This approach does, however, come with one big limitation. Because the output is just a single path, it can be given only one style. Even if that path appears to be multiple separate shapes, it's actually just a single SVG path element, with gaps in the shape described by the "d" attribute (i.e. with sub-paths). If we look at the same two-node path with the "Ruler" path effect applied instead, you can see that the result gives the appearance of numerous small paths. While it would be nice to be able to style the ruler's tick marks separately from the main spine of the shape, that simply isn't possible because, despite appearances, the output is still just a single path, with a single style.

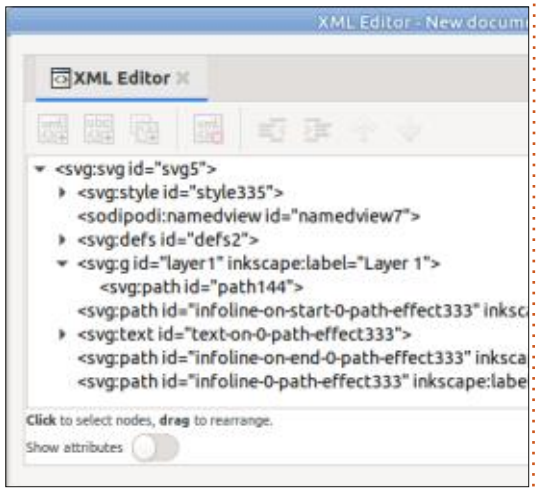


With the release of version 1.0, Inkscape has added the ability for path effects such as "Measure Segments" to break this historical limitation. No longer is an LPE limited to one path in, one path out. Let's apply "Measure Segments" to the same two-node path:



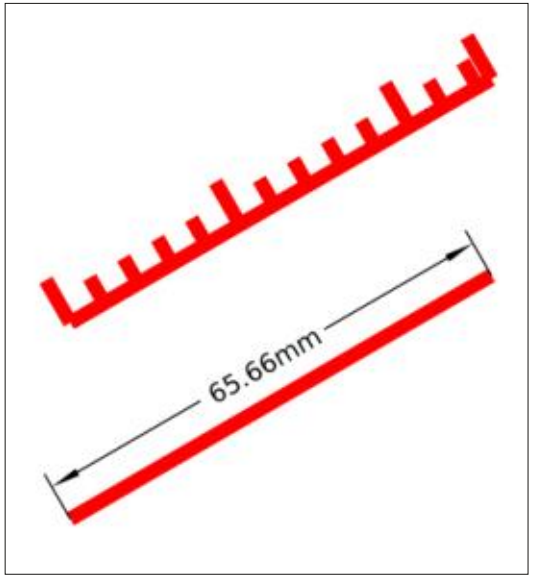
Immediately we can see that there are multiple styles being applied here. Our original path maintains the thicker style we used when drawing it, but the lines added by the LPE are significantly thinner. How is this possible? Quite simply, the lines added by the LPE are no longer just sub-paths in a "d"

attribute, but are additional SVG <path> and <text> elements in their own right. A quick look at the XML editor shows the difference. You might like to refer back to the earlier screenshot of this dialog, where the top section shows we just have a single layer with a single path in it. Now take a look at the document structure after applying this LPE:



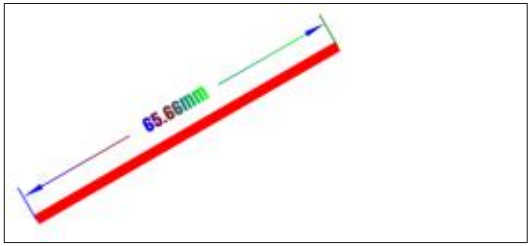
In addition to our original path, we now have three extra <path> elements (two leader lines and one measurement line), plus a <text> element to hold the measured value. Because these are separate SVG elements, you can obviously select them individually in order to style each of them differently… can't you? The answer to that question isn't the straightforward yes or no you might expect, so let's delve a little deeper still.

Intuitively, you might try to click on one of the new elements on the canvas in order to select it, but you'll find that your clicks are in vain. Dragging a rubber band selection box doesn't work either. The only thing you can select is the original path. Having selected that path, you can change its style as normal. As you've surmised by now, however, doing so will modify only the original path itself, not any of the elements added by the LPE. For example, note the difference in behaviour between the Ruler LPE and the Measure Segments LPE when I set a red stroke color on the original path.



As you'll know from last month's instalment, the color, font, line thickness, and other aspects of the Measure Segments LPE are set as part of the effect's parameters, split between the "General" and "Options" tabs. Should we wish to make the dimensions match the color of the original path, for example, we'll need to manually set it via the "Color and opacity" control in the "Options" tab. There's no means of linking or inheriting styles, though, so if you subsequently change the stroke color of the original path, you'll have to also remember to manually alter the LPE parameters to suit.

If we can't select the new elements using the mouse on the canvas, is there another approach we could use? Selecting individual elements within the XML editor does still work, and selects the corresponding item on the canvas when you do so. Even with that selection made, mouse interactions are restricted: you can drag the resize handles, but still can't drag the object itself to move it (though using the cursor keys will work); you also can't click on the object in order to switch to the rotate/skew handles or other modes now available with the selection box. You can change the style though, as demonstrated by this image of a multi-colored dimension line, complete with gradients and a different font.



There's just one problem with this approach, and it's something of a deal-breaker. The "L" in LPE stands for "Live" because the output from an LPE is calculated dynamically whenever the original path changes, or the parameters are adjusted. This means that any manipulation of the original path – even just nudging the position of one of the nodes – or any changes to the LPE parameters, will cause the output to be recalculated and all your manual changes to be discarded. You might think that this is okay, so long as you do your changes last, and then don't touch the object again, but the LPE output is also calculated when your file is loaded from disk: save the file, and reopen it later, and your manual changes are gone. There's

simply no way to manually edit the parts of the LPE such that Inkscape won't throw your changes away at some point.

The reason that these new elements are not selectable on the canvas is that they're all created in a "locked" state. The ability to lock objects has been in Inkscape for a long time, but has generally been a poor substitute for keeping objects arranged in suitable layers and locking the whole layer instead. This is because a locked object is difficult to unlock again – after all, you can't select it with the mouse to indicate which object you want to unlock. This situation improved with the release of Inkscape 1.0, which added an "Unlock Objects Below" entry to the context menu (see part 101 of this series for more details). Perhaps we could use that to allow easier editing of the individual components of our dimension line?

Sure enough, right-clicking on the dimension, and selecting the Unlock option from the context menu, does make the individual elements selectable with the mouse. Now they can be individually styled, and can even be clicked on to switch to the rotate/skew handles and other selection box modes. Internally, what has happened is that the "sodipodi:insensitive" attribute has been removed from each element's SVG node, which allows Inkscape to treat these elements like any normal selectable, movable, and editable objects… right until you edit the original path, alter the LPE parameters, or save and load the file. Unfortunately, just unlocking these objects isn't enough to break their connection to the Live effect.

So what's the solution? Is there a way that we can style the individual parts of the dimension lines beyond the limited options provided in the LPE parameters? Well, there is… but only in a way that removes their link to the original path. For example, do you want to style the leader lines as dashes, or with a different thickness to the arrowed dimension line? It's possible, but only by also losing the live update of the text value when you move or modify the path.

The way to achieve this is to use the Path > Object to Path menu entry. Historically, this has been the mechanism used to "fix" the output of an LPE, collapsing all the "live" parts of the effect chain to produce just a plain and simple SVG path that has the same appearance as the final LPE output. With the Measure Segments LPE, you can still use this same menu entry to "fix" the LPEs output, except this time the command's name becomes something of a misnomer: you are no longer converting the object into a <path> element, but rather breaking the link between the original path and the various generated <path> and <text> elements. In other words, choosing this option doesn't actually convert your object into a path, but it does convert it into separate editable objects. Naturally, this means that the elements are no longer "live", so you do lose all the auto-updating that is so useful in an effect like this.

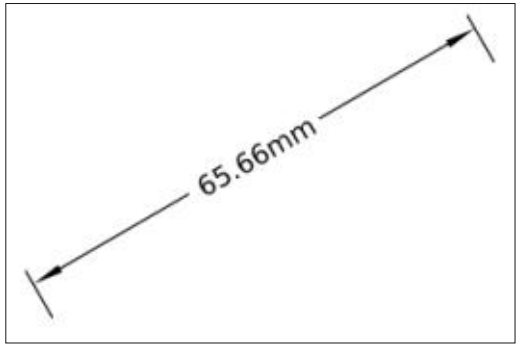
For most people, all this talk of styling parts of the Measure Segments LPE will be somewhat academic. In the vast majority of cases, the normal output from the effect will be sufficient, and the parameters it provides will give enough flexibility to style the new elements well enough. If more complex adjustments are needed, then using Object to Path will usually suffice, even if it does mean sacrificing live updates of the dimensions. It would be great if Inkscape offered a means to indicate that an element has been manually styled, but that you still want the position and text content to update, but perhaps that's too niche a requirement to warrant the development time.

Even if you don't want to style the dimension parts, however, there's one significant aspect of this LPE's approach that you should be aware of, because the behaviour is quite surprising, and could easily catch you unawares. The behaviour of Measure Segments with regard to layers is, in my opinion, broken.

Let's take another look at the new elements in the XML editor. This is the same content as the earlier screenshot, but I've cropped it to just show the relevant detail.

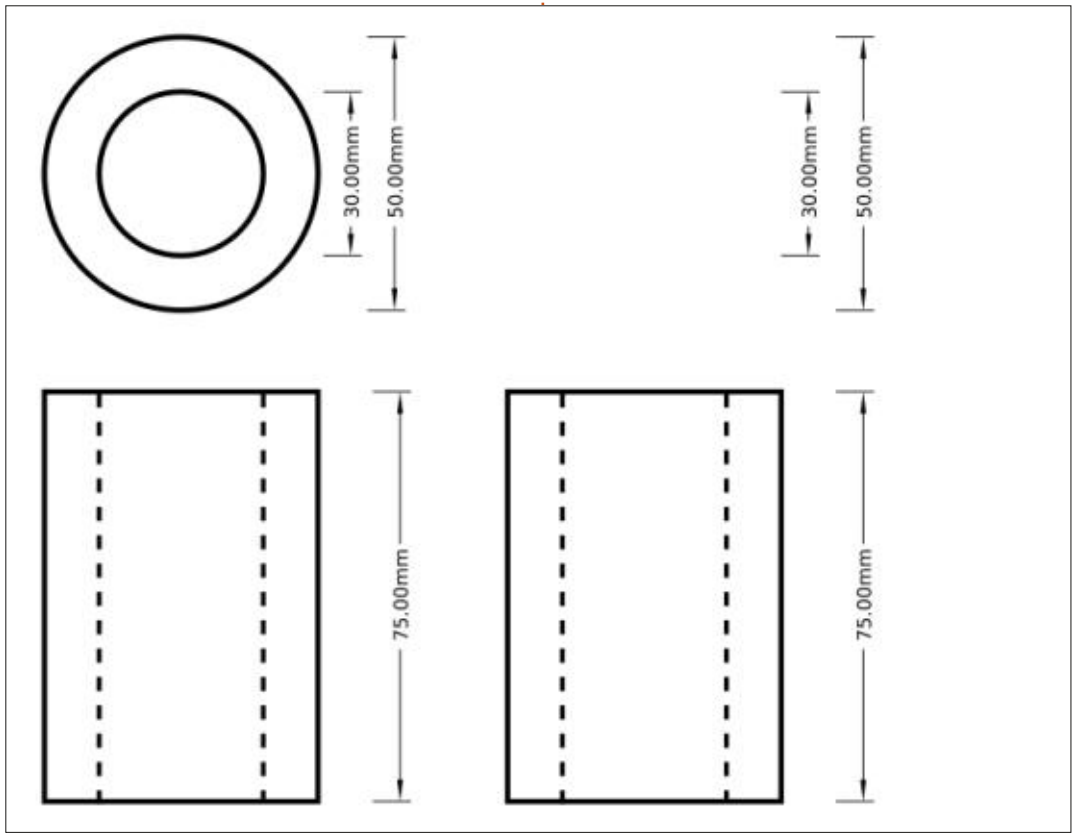
```
▾ <svg:g id="layer1" inkscape:label="Layer 1">
    <svg:path id="path144">
  <svg:path id="infoline-on-start-0-path-effect333" inkscape:label="dinhelpline">
▸ <svg:text id="text-on-0-path-effect333">
  <svg:path id="infoline-on-end-0-path-effect333" inkscape:label="dinhelpline">
  <svg:path id="infoline-0-path-effect333" inkscape:label="dinline">
```

Notice that "path144" is indented compared with the rest of the elements? That's the original path to which we've applied the LPE, and it's indented because it is a child of the Inkscape layer (the <g> element above it). The newly created <path> and <text> elements, however, are not indented because they are siblings of the layer. This means that they live alongside the layer, not inside it, in the XML structure. Now let's see what happens when we hide the layer.



The original line is hidden, but the dimension elements are not. They all live at the top level of the SVG, not within the same layer as the path they're associated with, so aren't affected by hiding the layer itself. This happens regardless of how deeply nested your original path is. Consider trying to create a technical drawing showing different views of an object: common sense would tell you to put each view in a separate layer so they can be turned on and off individually, but doing so will still leave the dimensions visible. In the following example the left hand image shows a simple technical drawing of a cylinder, while the right hand one shows the result of hiding the "Top View" layer. It's not exactly what most people would expect.

There is a solution to this issue, but it's not pretty. You can unlock the generated dimension content (right-click > Unlock Objects Below) – though you may need to do this multiple times for each part of the content. Then you need to select all the parts. Finally you can move them into the right layer using the Layer > Move Selection to Layer… menu option. Doing this will cause Inkscape to re-run the LPE, locking the objects again, but they will now be on the correct layer. The good news is that, once they've been moved, they tend to stay put.



Further changes to the path or the LPE parameters won't suddenly break them back out to the top level again. It would be much better, though, if Inkscape simply created them in the same layer as the original path by default.

Last month we looked at how to use this effect in practical terms. This time we've examined some of the technical details behind it. Now that the genie is out of the bottle, it's likely that future LPEs will also create new elements rather than just single paths, so understanding what's happening, and how they're different from older LPEs, might be a useful skill to add to your Inkscape repertoire.

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**

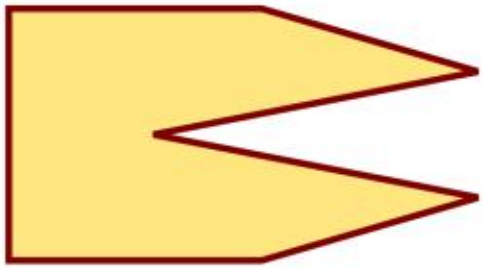
Inkscape - Part 121

Although it's not yet out as I write, by the time you read this article Inkscape 1.2 will have been released. This version is a major update that contains a huge number of changes and additions, and will likely provide plenty of topics for this column for many months to come. But let's not get ahead of ourselves – we haven't even finished looking at all the new Live Path Effects (LPEs) that were added to 1.0 and 1.1 yet!

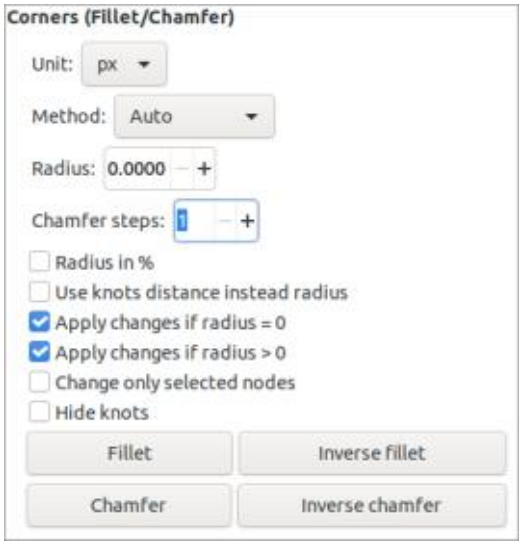
Corners (Fillet/Chamfer)

Consider two perpendicular lines meeting to form a 90° corner. Often that sharp transition from one line to the other is exactly what you want. But sometimes a design calls for something more gradual: an intermediate straight-line segment at 45°, or a rounded section that seamlessly joins the two lines. Creating such gradual corner transitions is known as chamfering or filleting, respectively, with the newly inserted path being referred to as the chamfer (for straight path segments) or fillet (for curved lines). Unsurprisingly, the "Corners (Fillet/Chamfer)" LPE is the tool to use when you want to quickly add such shapes to your paths.

As always, let's take a look at an example of this effect in action. Of course that means we'll need a suitable path to work on, such as this shape which has a selection of obtuse, acute and right angles so you can easily see how Inkscape applies the LPE in these different cases.

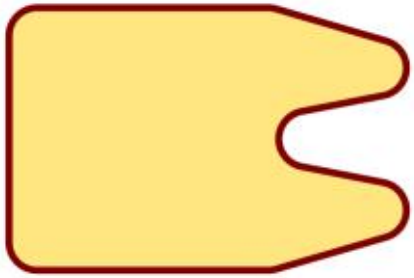


Adding the effect to a path like this probably won't produce an immediately obvious result, but that's just down to the values the parameters have by default. Let's take a look at the UI and examine each of the parameters individually, as usual.



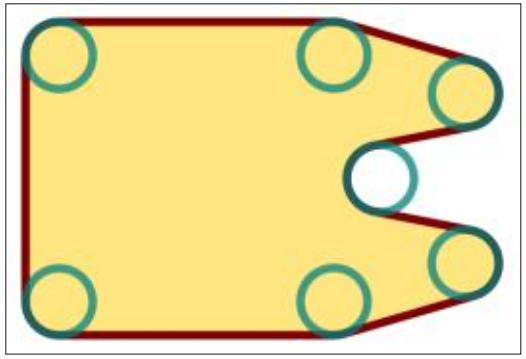
The Unit pop-up is pretty self-explanatory, though there is one omission that we'll come back to shortly. The Method pop-up allows you to explicitly determine whether fillets are rendered as arcs or Bézier curves, should you care about that distinction. Most people won't, and should probably just leave this as "Auto".

It's the Radius field that is the first really important one. While this is set to zero you won't see any filleting or chamfering effect, so the first thing to do is to crank this up to a suitable value for the result you want. If you're using a mouse with a scroll-wheel (which I contend is the best control device for serious Inkscape work), click in the field to focus it then roll the wheel to adjust the value in integer steps. By doing this you can watch the effect change the path on the canvas in real-time, making it easy to adjust the strength to the value that gives the right appearance. Here's our test shape with a radius of 15px.

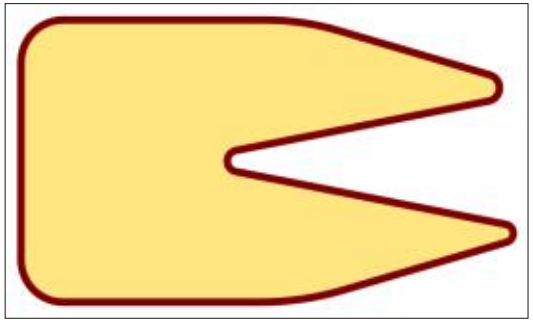


The filleting effect is obvious on the right-angled corners at the left, and completely rounds off the acute angles at the right. The obtuse angles at the top and bottom right, however, are barely rounded at all. This makes sense due to the radius being set to a specific value: as the two sides of the angle approach 180° so the tangent points get closer and closer, and the curving effect is less

pronounced. This can be seen more clearly by adding circles with a radius of 15px to each corner of our shape, to demonstrate how the fillet shapes are constructed.



If you want a smoother transition in your obtuse angles there are several solutions. The first is simply not to use a radius defined in explicit units at all, but make it vary based on the geometry of the path around each corner. This is what the "Radius in %" checkbox is for. Enable this and… nothing happens. A common problem with this and other LPEs is that some controls don't immediately trigger an update on the canvas. In this case the easiest option is to nudge the Radius value up by one, then back down by one, using the plus and minus buttons at the end of the field. With that refresh forced, our shape now looks like this.



Clearly the obtuse angles are a lot smoother now, though the acute ones have become sharper, with a smaller radius. This raises the obvious question as to what "Radius in %" actually means. Percentage of what? It's clearly not the same thing being used for all the corners, so it can't be "percentage of bounding box width" or anything like that. Is it based on a percentage of the segment length? The angle at which the lines meet? The price of Bitcoin? Who knows!? I certainly don't, and the tooltips aren't giving any insight.

One thing I do know, however, is that the percentage option tends to be more resilient to design changes. Consider what happens when you scale your shape up or down: if you've set a specific radius in pixels or millimetres, the LPE will change the output path in order to maintain that defined size. In the case of our example shape this causes the "prongs" to become longer or shorter. Conversely, when using the percentage option you'll find that scaling the path results in no significant changes to its shape. For this reason alone, unless you have a specific requirement that demands a fixed value radius, I suggest enabling the "Radius in %" checkbox.

Remember that I said that there was an omission in the Units pop-up that I would come back to? It's simply this: why isn't there an entry for "%" in the pop-up, instead of also having this checkbox? With the UI as it stands, it would be very easy to misread the parameters as indicating a fixed radius rather than a percentage, by overlooking the checkbox. As often seems to be the case with Inkscape's LPEs, some of the parameters and and their positions do rather leave me scratching my head.

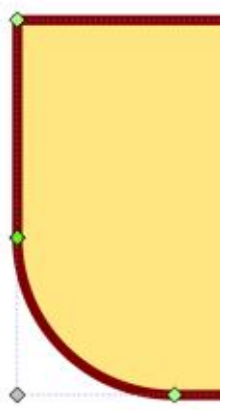
If you switch to the Node tool (F2) while your path is selected, you'll see a pair of handles for each node. On my setup they are rendered as particularly small shapes, so you may want to increase the size of the handles throughout the whole Inkscape UI via Edit > Preferences > Interface. These specific handles are referred to as "knots" in the Corners LPE interface. They can be hidden by enabling the "Hide knots" checkbox, so if you don't see them when switching to the Node tool, double-check to see if that box is enabled. These are also, presumably, the knots referred to in the label for the "Use knots distance instead [of] radius" checkbox – though I'm not certain because the behaviour of that control is less than obvious, and I'm not even convinced that it's needed at all!

Checking that box (and nudging the Radius control up and down) modifies the shape once again. The positions of the knots move, and with it the curvature of the various fillets change. The thing is, you can actually drag the knots on the canvas in order to manually adjust the curvature for each fillet – and this works regardless of the state of the checkbox. I'm therefore at a loss as to what this checkbox is meant to achieve, so my advice is to simply leave it un-ticked, enable the "Radius in %" option, and manually adjust any fillets you need to.

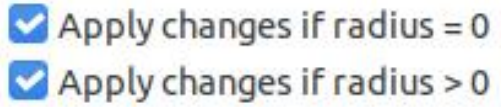
It's worth reiterating the fact

that you can change each pair of knots individually. This is the first LPE to support storing per-node data, allowing different parameters to apply to different nodes within the path. This allows you to not only set a different radius or knot position for each corner, but also to mix-and-match between fillets and chamfers, as we'll see later, all with just a single instance of the LPE. Compare this with the need for multiple LPEs each with its own blacklist or whitelist that we had to use to achieve something similar with the Measure Segments LPE, and I'm sure you'll agree that the new method is a lot more straightforward to use.

Now that you know how to set the fillet radius on a per-node basis, you should be able to reproduce a pair of corners like these.



The top corner has a radius of zero, whereas the bottom corner's radius is set to a much larger value simply by dragging one of the knots. It's important to note the distinction between a zero radius node and one with a radius greater than zero as we take a look at the next two checkboxes in this LPE.



Here we have another pair of controls that are, in my opinion, practically useless. When we first began to apply fillets to this shape we increased the radius value and all the corners responded. Suppose, however, you've manually reset some of them to a radius of zero, and don't want them to be affected by further changes. Unchecking the first box will mean that any changes you make in the LPE interface won't affect those zero radius corners. In other words, if you want to keep your square corners square while adjusting all the others then uncheck this box.

The problem is what happens if you do want to adjust the square corners as well. Obviously you need to have this box checked, but that's not really enough. As soon as you nudge the radius parameter up, those corners cease to be zero radius corners, so that checkbox no longer applies. This is where the second checkbox comes in: with this checked your changes also affect non-zero corners. Unchecking this would mean that your changes only affect the zero radius corners, which is almost never what you want – especially if it's the radius parameter you're playing around with. My advice, therefore, is to always leave these two checkboxes ticked. If you want to protect your sharp corners from changes, there's a better way to achieve that which I'll describe shortly.

Personally I think these two parameters should be collapsed into a single checkbox labelled "Protect zero radius corners". When checked, the tight corners would be left unmolested by any changes to the parameters, but in its unchecked state your changes would affect all of the corners, as usual. In reality even this probably isn't required, given the next checkbox in this dialog.

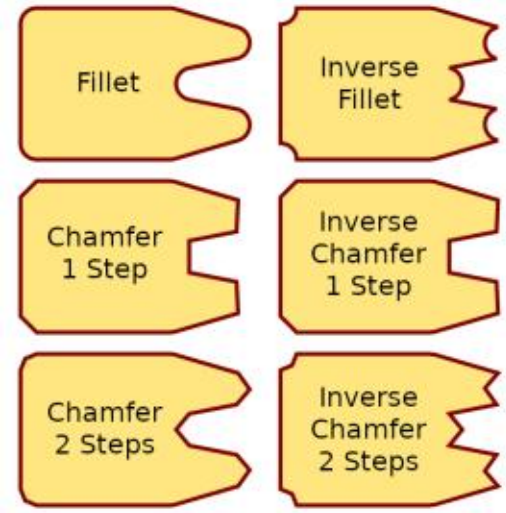
What if you don't want to affect all of the corners, but perhaps the ones you want left untouched already have a non-zero radius? We've seen that the radius can be adjusted on a per-corner basis using the knots on the canvas, but what about the other parameters? The checkbox labelled "Change only selected nodes" is the option for you. With this enabled any changes you make to the LPE's parameters will only be applied to corners that you've selected. This renders the previous checkboxes rather redundant. If you want to modify all the corners then just ensure that all of them are selected. Want to leave the zero radius ones untouched? Just make sure they're not selected (but the other corners are) when you make your modifications. Importantly, however, you can also choose any subset of corners to adjust at once, regardless of their current radius.

In the unlikely event that you're not familiar with selecting nodes in Inkscape, here's a quick recap. First, you need to be using the Node tool (F2). You can click on individual corner nodes to select them, or on a path segment to select the nodes at either end. You can also drag the mouse over multiple nodes to select them (a so-called "marquee" or "rubber band" selection).

Holding the Shift key will let you add nodes to, or remove nodes from, an existing selection. Most usefully, Ctrl-A will select all the nodes in the path, providing a quick way to alter all the nodes at once. The Escape key will deselect them all, should you wish to start again with a fresh selection. If you have difficulty seeing the path nodes due to the placement of the LPE's knots, don't forget that you can check the "Hide knots" control while you make your selection. With those basics at your fingertips you'll soon find that it's quick and easy to select exactly which corners should be affected by your LPE changes at any time, without having to consider their existing radius or other attributes.

Now we know how to apply parameter changes to specific nodes, but so far all the examples we've looked at have been fillets. What about chamfers? It doesn't take a genius to figure out that's what the buttons at the bottom of the LPE parameters are for.

Depending on the state of the "Change only selected nodes" checkbox, clicking on one of these buttons will change either the selected nodes, or all nodes, to the appropriate type of join. For chamfers and inverse chamfers the "Chamfer steps" parameter also plays its part, dictating how many straight line segments should be used to make up the connecting shape. Note that when this is set to 1 there is no visual difference between a chamfer and inverse chamfer. Here's an example of how the different types of join are rendered with our test shape.



In conclusion, I think this is a very capable and useful LPE that is only let down a little by offering too many non-intuitive options in the UI which don't seem to really provide much benefit. My advice is to enable the "Radius in %" checkbox, both the "Apply changes…" checkboxes, and the "Change only selected nodes" checkbox. That will give you an LPE that behaves predictably when you resize your objects, and which allows you to trivially alter all of the nodes, or just a subset of them, depending on what you select on the canvas.

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**