Spiff up your website with KImageMapEditor

A brief tutorial to a small application that can save you a large amount of time

Terry Hancock

One of the things I love about using a large free software distribution, especially on a suitably large harddrive, is that you can sometimes just go exploring in your applications menu. It seems like there's always something there I haven't looked at yet. Jan Schäfer's KImageMapEditor was one of these discoveries—and what a gem it turned out to be! Oddly, you probably won't find this in the "Graphics" menu, but rather under Debian→Apps→Editors, where you usually find text tools (of course, your distribution may have it in a different place). What it does is to make HTML "map-area" tags (which define clickable "hot spots" in an image) completely simple to use, by providing a graphical editing environment for laying them out.

You probably already have KImageMapEditor if you use KDE. If not, you can find it as a DEB package [1], an RPM package [2], and of course, as source code from the project site [3].

The problem

I've long avoided the map-area tags, because they just seemed too complicated. You have to tinker with coordinates within an image, measure out where points should go, and create a complex set of them in order to get them to work. Besides, I already had a tool for doing similar effects—I was a real wiz at slicing up images and arranging them in tables.

Recently, I also began using HTML to provide in-package documentation for a software package "Universe" [4], which is based on the "Python Universe Builder" (PUB) [5]. I wanted to mingle my pre-written design documentation with the automatically generated API documentation from epydoc, and there's a nice hook which allows me to link them by putting some content on the "index.html" page that epydoc generates.

You probably already have KImageMapEditor if you use KDE

I decided a diagram would be the perfect way to quickly communicate the relationship between the modules in the package, and give potential developers a quick way to get started on the project. So, naturally, I whipped out my HTML table skills, made a zillion (okay, just fifteen) images of connecting elements and assembled them with the appropriate text in a table, resulting in the diagram in figure 1.

This took, as I now see, 183 lines of table code, and was extremely tedious to maintain (as I discovered, for example, when I added the "sound" module!). What a waste of time! Fortunately, I have discovered KImageMapEditor, which I can now use to greatly simplify this page, making it much easier to maintain, as well as much more visually appealling.

First, you need an image

With the KImageMapEditor approach, the first requirement is a good starting image. For a diagram like this, I highly recommend using a vector-graphics tool like Dia [6], Skencil [7], or Inkscape [8]. I've had success with all three, and used Inkscape to create the drawing for this project, which is in figure 2 (online readers can simply use the figure image to follow along—but feel free to start right away on your own image).

As you can see, I've gone beyond my original concept, by adding images, descriptive of what each module is responsible for (this package is mostly graphics and presentation, so this is a natural choice for this project). I just used polybezier lines to lay out the relationships, which obviously sidesteps the complexity of the arrow images I used in my original design.

I made a backup copy of my HTML file and then deleted the enormous table from my original, replacing it with a single image tag, embedding the above image into it. At this point, it's just a dumb image, without anything clickable. The tag itself looks like this:

```
\<img src=''../image/universe\_plan.png''
     alt=''Universe Modules''
     width=''597'' height=''656'' border=''0'' />
```

Figure 1: The "old way", with HTML tables. Dull and too hard to maintain.

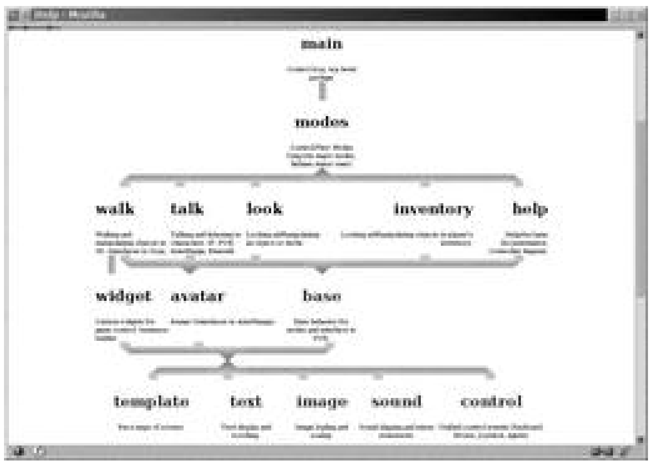
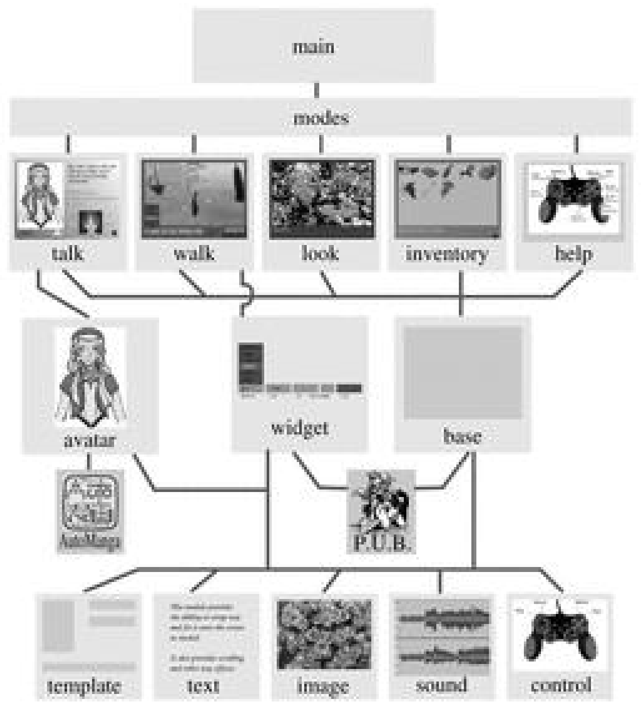


Figure 2: My grand plan for the universe, as an image.



Start KImageMapEditor

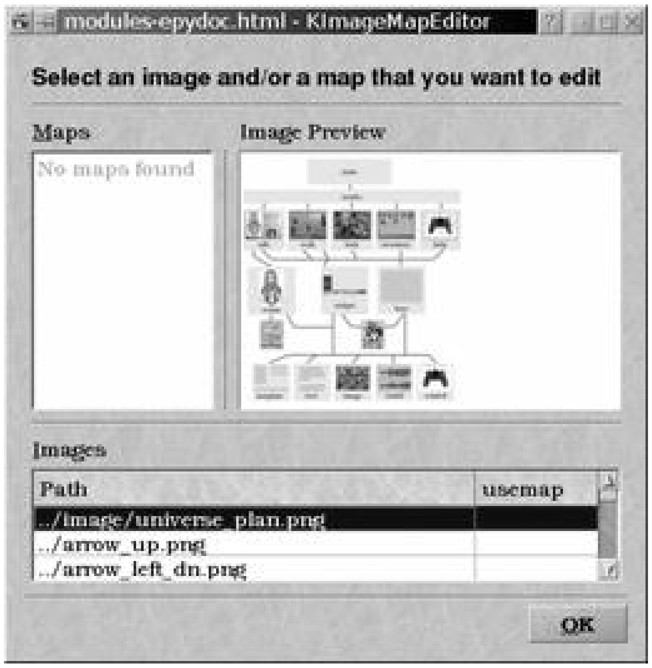
The next step is to start the KImageMapEditor. Normally, if you haven't used it before, it will simply bring up an empty screen. Unsurprisingly, you will click on File→Open to open the HTML file (You can also start by just loading an image, in which case KImageMapEditor will create a new HTML file). The editor will analyze the HTML and let you choose from the images that it finds (figure 3).

In this case, there's not a lot else in the file, so the first image is what is wanted. So, select that, and click "Ok". This will load the image into the editor, as shown in figure 4.

You will note, looking at the screen, that there is a canvas area on the right and three tabs on the left. First, you will need to create a map to edit. So, click on the "Maps" tab; then right-click in the empty window; and select "New map…". You can choose any name you like. I picked "modules" for this project.

Next, you will need to attach this to your image. Clicking on the image tab will show you the image you are working on. Right-click on the image you want to use and select "Edit usemap…". This will allow you to specify a map for

Figure 3: Selecting the correct image from the HTML source.



modules-epydoc.html - KImageMapEditor

Select an image and/or a map that you want to edit

Maps

No maps found

Image Preview

Images

Path	usemap
../image/universe_plan.png	
../arrow_up.png	
../arrow_left_dn.png	

OK

the image using a combo box based on the "Maps" tab, so we can just select "modules" off of the list at this point.

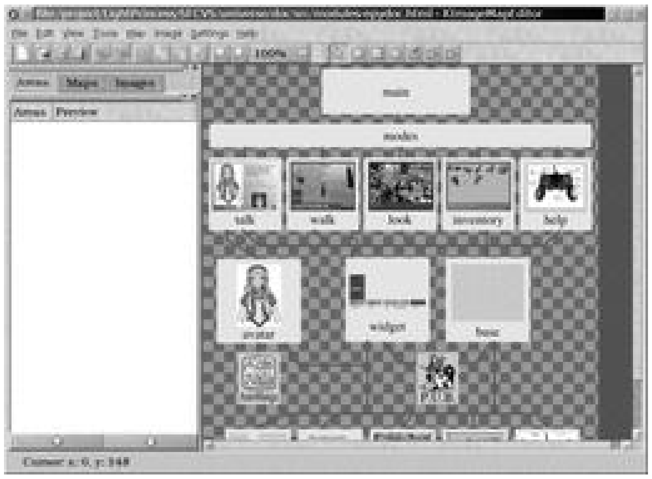
That takes care of the "boilerplate code". Next we'll click on "Areas" to get started defining the hotspots in this image.

Defining areas

If you look at the top of the canvas part of the screen (right side), you will see a palette with the area-definition tools. The first tool (arrow icon), is used to select already-defined areas. You can't do that yet, but you'll get a lot of use out of it later. The next tool is the circle tool (circle icon), which makes circle-shaped areas, as you would expect; though, if you are experienced with vector graphics programs, you might be surprised that it cannot make ellipses. The next tool, which will be using a lot of, is the box tool (square icon), which defines rectangular box-shaped areas.

The next four icons can be used to draw more-or-less arbitrary shapes. The first makes (irregular) polygons, allowing you to click for each vertex (if you click close enough to the first point, the polygon closes). The next icon provides "freehand" definition of polygons. I recommend you alway use the first polygon tool, which allows you to explicitly specify points, because the freehand tool will generally

Figure 4: What you'll see after you've loaded the image.



create a lot of points, making your HTML a bit untidy. You should realize that there is no need to draw with the precision you would ordinarily use in creating graphics: the viewer will never actually see what you draw, you're just defining active regions in your image. Hence, a couple of pixels of misfit is just not going to matter that much anyway, and I think it's better to keep it simple.

The viewer will never actually see what you draw, you're just defining active regions in your image

In fact, since this project is a simple block diagram, I'll only be using the "box" tool.

My first block was on the lower right, so I scrolled the canvas up; selected the box tool; and clicked and dragged it over the "control" box in my diagram. As soon as you let go of the mouse button, a dialog will pop-up prompting you to edit the properties of the area (figure 5). This is just a time-saver, you can actually click past this, and come back to it later, so don't panic if you miss something. For now, I just put in the correct module link (this is a URL that's automatically generated by epydoc in my project, but you can use any URL appropriate for your web page), and a brief descriptive label "control".

You'll also note that the area appears on the area tab at left, with both the URL and a thumbnail of the image in

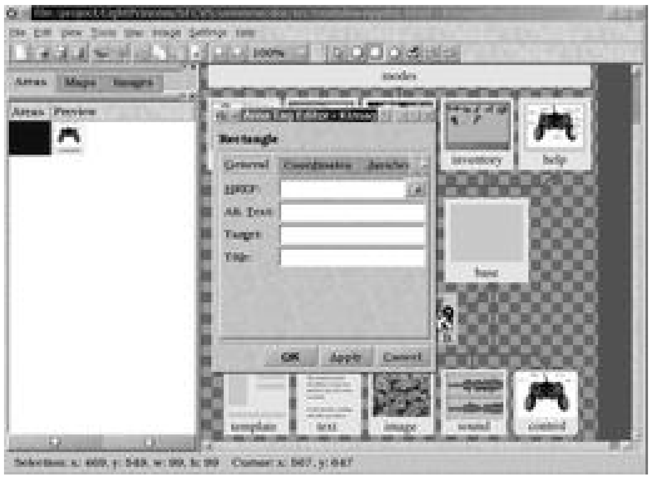


Figure 5: The properties dialog will come up as soon as you finish defining the area.

the area. Defining the rest of the areas in the image is just as easy, so I repeated the process for all of those, ending up with what you see in figure 6. You'll also note that the descriptive labels are shown in the graphics canvas area.

Testing and cleaning up

At this point, you can use File→Save, and the changes will be saved to the HTML file. This will allow you to load it immediately and test it in your favorite web browser, which will show you the final result, as in figure 7 (of course, this

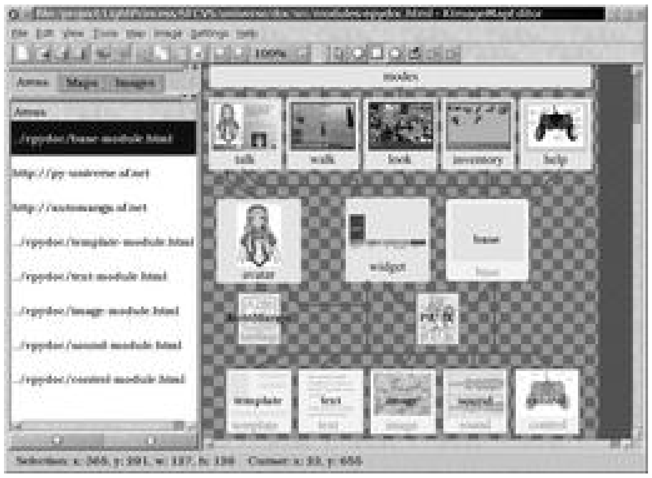


Figure 6: My session, after all of the implemented areas have been set.

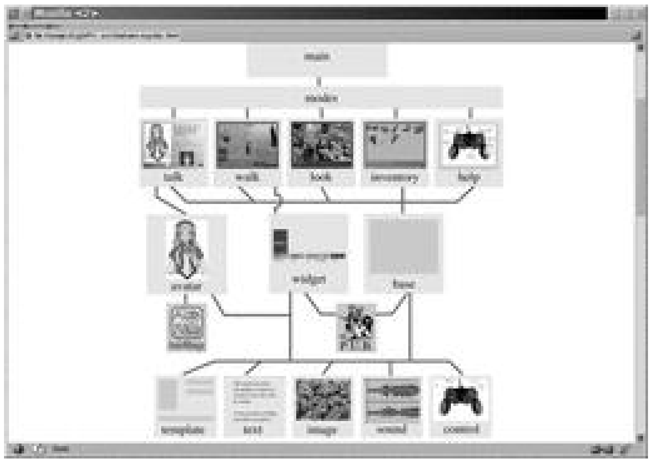


Figure 7: What it looks like in the browser (unfortunately, it's hard to show that it is now clickable).

looks just like the HTML with the plain image, but it's clickable now!).

You can be a lot more daring with the layout of links

KImageMapEditor doesn't know where to put the map element in the HTML document, so it just sticks it on the end. I immediately cut and pasted this section to a position immediately after the image tag, which I find more intuitive. KImageMapEditor is unfazed by this and will keep the map element wherever you put it (I have noticed that it may scramble the attributes on the image tag, which is a little annoying, but you can still find everything). The result is quite compact compared to my original solution, taking up less than 19 lines of code (which I don't even have to manage manually). This, combined with the ease of editing the original graphic in Inkscape, makes this a *huge* improvement in terms of maintainability!

Going further

Universe, like most projects is a work in progress, and although this diagram outlines my grand development scheme, only parts of it are complete. So far this is only apparent because the unimplemented parts of the image aren't clickable. But I'd like to be a bit more explicit. I could modify the image to indicate this, but I can also use KImage-MapEditor to actively alert the reader that those parts aren't implemented.

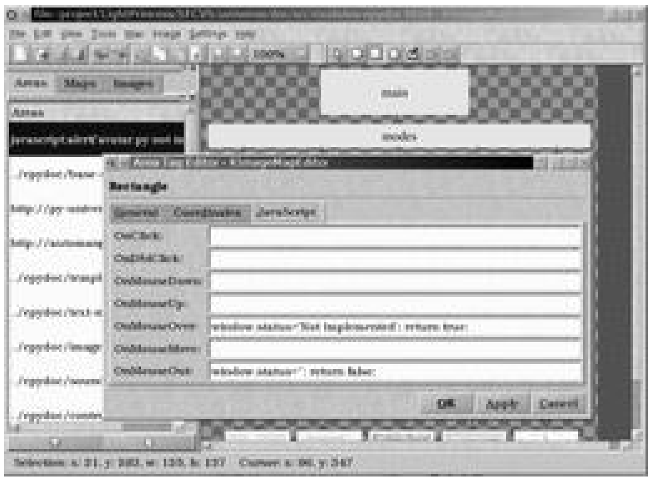
For this, I used the editor's ability to manage Javascript tags. First of all, I defined the extra area tags for the unimplemented modules. Then I used a Javascript URL as the link:

```
javascript:alert('Not implemented.')
```

This will make a Javascript dialog box pop-up if you click on it. Of course, that could be sort of annoying since I've now made a "do-nothing" link. I needed to add a brief note for what each module is supposed to do, so I used the `onMouseOver` and `onMouseOut` tags to control the status line, so that the words "Not Implemented" immediately appear there as soon as you put the cursor over those modules.

To do this in KImageMapEditor, you select the area you want to use (from either the area list or the canvas), right-click, and select "Properties...". This takes you back to the dialog you used in creating the areas in the first place. Now, though, click on the "Javascript" tab to get to the various event trigger hooks that you can use, including `onMouseOver` and `onMouseOut`. I then defined these with simple tabs (figure 8).

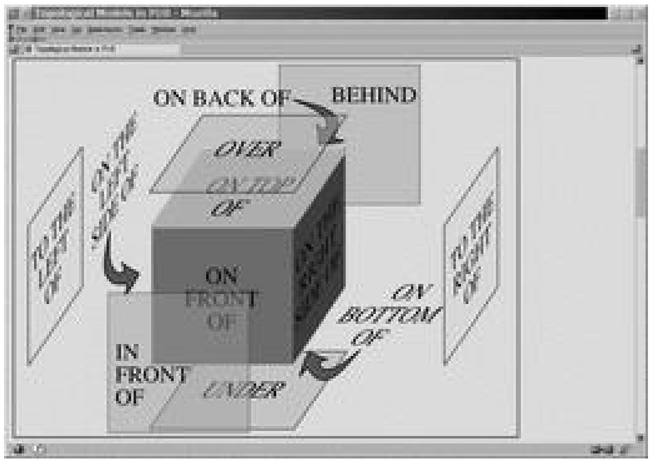
Figure 8: Managing javascript hooks.



OnClick:
OnDblClick:
OnMouseDown:
OnMouseUp:
OnMouseOver: window.status='Not Implemented'; return true;
OnMouseMove:
OnMouseOut: window.status=''; return false;

Selection: x: 21, y: 103, w: 135, h: 137   Cursor: x: 86, y: 347

Figure 9: Much more complex 3D diagram, which uses polygon-areas (from PUB [5]).



More complex drawings

Obviously, it is possible to go a lot further with KImage-MapEditor. You can be a lot more daring with the layout of links. With the polygon tool, it's possible to make diagrams with some three-dimensional effects (figure 9), and complex artistic designs that would be nearly impossible with tables.

That's it! I hope you find KImageMapEditor as useful as I have. It's a cool graphical tool that solves a problem simply and efficiently, and it's definitely going to be a regular part of my web crafting software from now on.

The result is quite compact compared to my original solution, taking up less than 19 lines of code (which I don't even have to manage manually)

Bibliography

[1] Debian Packages (`http://packages.debian.org/kimagemapeditor`)

[2] RPM Packages (http://rpmfind.net/linux/rpm2html/search.php?query=kimagemapeditor)

[3] Source Code (http://www.nongnu.org/kimagemap)

[4] Part of my game project, The Light Princess (http://light-princess.sf.net)

[5] Joe Strout's Python Universe Builder (http://py-universe.sf.net)

[6] Dia (http://www.gnome.org/projects/dia/)

[7] Skencil (http://www.skencil.org)

[8] Inkscape (http://www.inkscape.org)

Copyright information

About the author

Terry Hancock: Terry Hancock is co-owner and technical officer of Anansi Spaceworks (http://www.anansispaceworks.com/), dedicated to the application of free software methods to the development of space.