

Self-Modifying Help Patches

10th April 2011

1 The trouble with help patch templates and the lack of them

Having followed the PDDP help patch template committee back in 2006, I have noticed several things. The template is a file meant to be copied so that it gets imitated, and then the elements in it need to be copied, imitated and customised, kept aligned, etc ; this was claimed to be the new standard for help patches, that everybody should follow in order to have consistently clean-looking documentation. There was very little adoption of this template when making help patches for new externals and abstractions, and there was also very little adoption of this format when updating existing help patches. This led me to think that all that the template is doing, is creating more work to be done by volunteers who completely choose to do one thing or another and end up consistently kicking the PDDP template down the priority ladder, because why spend time following a help template when you can spend time coding and fixing bugs. I've been one of them.

Yet, at the same time, the structure of a template suggests some goals to reach when making help patches. PureData help is often in the paradigm of putting an object of the class to be documented, surrounding it by things connected in its inlets and outlets and ask people to play with it until they understand what's going on. This is especially bad for knowing when it is that the object class has a bug in it : If the output values of the object are its own documentation, an object can never really be found to be wrong. In contrast, a template with fields to be filled suggests some expectations that some text has to be written. It's also very possible to make thorough documentation by not using a template, but I am talking about incitations, friendly reminders and frames of mind. That's only one big advantage of using a template.

So, this suggests to me that PureData help needs to both use a template and not use it. It's not really a contradiction. There are some assumptions about templates that have to be defenestrated for the good of the community so that there are no longer big incentives to not using the template.

2 Copy-Paste to encourage Big Design Up Front

If ever you feel like stretching a committee meeting, ask them to decide important things up front, such as colour. Colour is one of the things that people feel strongly about. Choice of colour was the first negative criticism I received about my new help patches. However, regardless of justification, it amounts to personal choice, therefore people can argue about them for long time if they are told that they have to decide that up front.

In a system in which you have a template decided in advance, that gets copied and pasted and recopied and repasted, you need to figure out what's the best up front, even though you don't know yet what is best, because you learn that by trying the system over extended periods of time. It's possible to search-and-replace in hundreds of helpfiles after changing your mind, but it isn't a very good solution and can cause problems of its own.

To get out of the infinite loop of colour choice, the PDDP template reserves a receive-symbol so that the colours can be configured using a [s], although there was no talk of how people would make this happen (a hidden patch that continually sends the colour settings...? that's the only way I could think of).

But it does not stop there. We might want to change fonts, adjust fonts because they still differ from platform to platform (and between Tcl/Tk 8.4 and 8.5), and then change some spacings and amount of padding around the text and fitting box sizes of [cnv] objects to the text we write over them and perhaps change the wording : what if Arguments is now to be called Constructor Arguments, or Creation Parameters ? And now what if we want to write that in Russian. But we might also want to turn those words into hyperlinks to about_arguments.pd, or add mouse-over explanations, etc. Some of those things might look like complicated decisions when we choose to not use the appropriate tools to support our efforts.

Then one ought to wonder why we aren't using three of Pd's greatest gifts : Abstractions, Graph-on-Parent, and Dynamic Patching.

3 Help the help

I don't want to do work for the template. I would rather have the template to do work for me. Therefore I introduced the following things in my help patches :

3.1 Abstract out the nodes

Instead of putting a comment on a [cnv] to say "Creation Arguments", I create a [doc_c] object, which does the rest on its own. Similarly I have [doc_i] and [doc_o] for inlets and outlets, and I have [doc_cc] [doc_ii] [doc_oo] for individual arguments, individual inlets and individual outlets. Each method selector (or creation argument type) gets a [doc_m] object.

Then the [doc_m] object gets attached to comments. This is allowed by an external that modifies PureData at load-time so that comments have an inlet

that is both invisible and connectable to.

[doc_also] is the “See Also” section.

[doc_h] represents a header bar, whereas [doc_f] represents a footer bar.

That way, the appearance can be controlled by editing the abstractions in a centralised place. But there are many more possibilities that arise from using abstractions.

3.2 Each node knows its own aggregate size

Abstractions such as [doc_bottom] [doc_below] and [doc_layout] are used so that each help-node can query the sizes of the help-nodes that it is responsible for, but also so that it can report its own size to their superiors. But this does not work in the case of comments. Comments couldn’t become just abstractions instances because then we wouldn’t be able to edit them in the usual way. This is why i attach them to the right outlet of [doc_m]. Then an external class is responsible for querying PureData’s internals to find out the total size of the comments.

Thus we now have a way to figure out whether any two objects overlap, or are lacking any amount of spacing that we want them to have.

3.3 Each node can push itself leftwards, rightwards or downwards

[doc_layout] and friends also help the help-nodes move around in the patch. This means that [doc_h] forces itself to go at position (0,0), while [doc_c] forces itself at x=0 and any y below [doc_h] (plus some spacing). Then [doc_i] forces itself to be at x=0 and sufficiently below [doc_c], including all the subnodes of [doc_c].

3.4 Each node can push itself upwards if told to

Currently, help-nodes never move automatically upwards. This is because some elements are not taken into account by the help system, and they need to be before any node can be sure that it can go upwards. When all the elements will be taken into account automatically, there will be no “hand”-positioning of objects anymore, and thus the layout will have been separated completely from the content and structure. In the meanwhile, there is a “roll up” toggle in the [doc_h] abstraction to cause the nodes to go upwards as much as they are allowed to, but it can’t be turned on all of the time, because of the exceptional elements.

3.5 Non-nodes have to be moved by other nodes

The comments can’t move on their own, therefore [doc_m] is both responsible for moving itself and moving its comments so that they don’t overlap with each other. This can also apply to most non-comments. [doc_also] uses the

same trick. Non-nodes can always and will always be moved in all four directions. [doc_m] places its non-nodes vertically, whereas [doc_also] places them horizontally.

3.6 Edit-mode reveals the abstractions' arguments

This allows to edit the arguments of a GOP that would otherwise hide its arguments, while not needing property dialogues. The GOP flag looks like it's getting automatically toggled whenever switching from run mode to edit mode and back (though in practice it is not the case). Abstractions that need act like that are using the [doc_editmode] abstraction to achieve this result.

3.7 Automatic node creation

[doc_h] knows that [doc_c] [doc_i] [doc_o] [doc_also] [doc_f] need to exist, therefore it creates them, with simple dynamic patching, on the condition that the [doc_exist] abstraction says that they don't exist already.

[doc_c] [doc_i] [doc_o] take an argument to specify the number of subnodes. At loadbang-time, a counter goes through all the subnodes that should exist and creates them unless [doc_exist] advises otherwise. This uses the [doc_make] abstraction.

[doc_cc] [doc_ii] [doc_oo] have a small button that when pressed creates an automatically-associated [doc_m] and clicks in it to start edition of its arguments. This uses the [doc_add] abstraction.

[doc_m] has a small button that creates a comment, connects to it, and starts edition of its arguments.

3.8 Links

[doc_link] are hyperlinks to any other help patch of that manual, and unlike [pddplink], it does not need an external server with a 5-second timer on it. When that help system will go beyond the scope of one manual, [doc_link] will be adapted to search for help patches.

3.9 Screenshots

[doc_h] contains a subpatch that takes multiple screenshots of the help patch, while moving the scrollbar, and then joins the screenshots together, to make a long vertical screenshot, which is saved as a single PNG. This allows help patches to be seen directly on the web, without any additional tools, and with the exact same look as on the screen (on **my** screen, anyway). This is automatable and driven from another patch called shoot.pd which opens each help patch and tells it to take a screenshot of itself.

3.10 Varia

- [doc_same 0] stands for the phrase “Same as argument 0”.
- [doc_accolade] draws a big brace.
- [doc_section] looks like [doc_c] [doc_i] [doc_o] but has customisable text and manual placement.
- [doc_iemfont] corrects the platform-dependent aspects of IEM font sizes. The difference between Tk 8.4 and 8.5 (that exists on Linux) is currently not corrected, but can be corrected by just editing [doc_iemfont], which is in turn used by any GOP abstraction that displays any IEM-style label.
- [doc_demo] makes a [doc_h]-lookalike bar that is meant to go in non-help patches only.
- Segmented wires indicates that a [doc_m] is connected to a subordinate (comment or other).
- Those wires, as well as [doc_also] wires, are hidden while in run-mode.
- dashed bounding boxes can be drawn by the nodes, but this is turned off, as it was mostly only useful during the development of the system.
- Window size is enforced to a certain fixed width, and a minimum height so that small help patches don’t have a scrollbar. This minimum height itself has a maximum, so that long help patches use scrollbars so that their windows can fit on most screens.

4 Portability

This system needs a library that is not present in Pd-Extended, and if it were present in Pd-Extended, it would not work in distributions of PureData that don’t allow themselves to bundle the externals and abstractions that are needed to make things happen.