Port your widgets to GTK4

Matthias Clasen

>> Is this better?

>> Yeah, that's pretty good.

>> Okay. I think we can restart. This is GUADEC 2020. Our first day, first track. And we continue with Matthias Clasen, port your widgets to GTK4. Please.

Matthias Clasen: Thank you, yes, welcome back, everybody. My name's Matthias. I'm supposed to talk about GTK4 today. Or more specifically about porting your custom widgets from GTK3 to GTK4. As I was preparing for this, I took a bit of a look back at our GTK4 development and I was kind of astonished to see that it's been almost four years now that we started working on GTK4. The release came out in November 2016. And since then, we've racked up well over 16,000 commits in the GTK master branch. And we've had did a couple of fearless early adopters come by at various time to port some applications to snapshots. There was a port around 396, and GNOME to do around 3.98. And we were lucky to be able to observe the attempt to use our new drag and drop API life just as we had committed it. We learned a lot and our drag and drop is a lot better as a result. Thanks a lot to all the early adopters.

But now GTK4 is almost here. In fact, I was hoping to have a 3.99 release done by GUADEC. That did not quite happen. We are waiting for the accessibility occupy to land that Emmanuele was talking about before the break. But that 3.99 release will happen very soon. Either during GUADEC or shortly after. And our plan for the 4.0 release is still to get it out the door before the year is over. It's a great time to take a detailed look at what it will take to port all of your applications from GTK3 to GTK4.

So, in this talk, I will focus on the basic aspects of custom widget functionality and how it changed in the move from GTK3 to GTK4. It will also give us a chance to take a look at the core aspects of GTK such as rendering and layout and input handling. And see what changes have happened there.

But before I dive into details, you wanted to take a short amount of time to explain a little bit the principles that are guided our work on GTK4 so you have some idea what the motivations are behind some of the changes you will see later.

And the first part I want to mention here is the first general direction API changes have been to delegation over sub classing. Part of the motivation behind that, hopefully makes writing custom widgets easier and less error prone. And as a result, you will see auxiliary objects that take over core widget functionality. And you will see a lot of widget transfers are now final since you're actually expected to derive directly from GTK widget. I think you could maybe summarize this principle as lessons learned from clutter, if you wanted.

Another dell trend in our API is we want to treat everything as a widget as far as possible. That started already in GTK3 with -- when we slowly broke up complex widgets. First we introduced CSS nodes and then gadgets. And now in GTK4, for example, the GTK scale widget has a trough and a slider, not just gadgets. But are fully-formed sub-widgets that can maintain their own style and state and can handle input just like every other widget.

A big loser in the transition from GTK3 to GTK4 is GTK container. As a base class, it's become less important. Any widgets can have them now. And properties have been replaced but layout children and their properties. And the focusing has moved from GTK container up to GTK widget. So, there's really not much use for GTK container at all. In fact, it's lost so much that it doesn't exist anymore. We have gotten rid of it.

Another big loser is GTK window. In GTK3, all the pop-ups and all the different kind of top levels such as anti-completions, menus, tooltips or combo box drop downs all had -- were using a GTK window underneath somewhere. And we've replaced most of that with pop overs over GTK pop-up services. And in addition, we have untangled the popover implementation from GTK window. And we've also moved some top-level specific GTK window functionality out into separate interfaces such as GTK rule or into subwidgets such as GTK one doe controls. So, overall, GTK window is a lot less scary than it used to be.

And on the GTK level, we've moved from modeling our abstractions on X11 concepts to following Wayland. As a consequence, we now have surfaces instead of windows. And things like screens or visuals, explicit grabs are gone. And we are moving to -- move away from the complex input device hierarchy that was characteristic for X11 input too. Overall, this change should not affect applications too much because they don't directly necessarily use these objects. But it does affect GTK back-ends. And hopefully makes it easier and straightforward to write new back-ends. In fact, just yesterday, we merged the new beginnings of a new OS spec that needs more work before it's fully fleshed out. But I think it's a hopeful sign for thing to come.

Moving on from principles to surprises. With any change of this size such as moving from GTK3 to GTK4, there will always be some surprises and alliances, unexpected changes that you did not see coming. And that will hold up much longer than they should. And that'll probably leave you angry and annoyed. So, I figured it might be a good idea to point some of these out beforehand so that you are forewarned and don't get blind sided.

The first one that I want to mention is that widgets are now visible by default. This is one of -- one kind of API change that we've made in a number of places where the defaults for properties were sometimes just wrong in the past. Because when you're creating a new widget, it seems very likely that you also want to show it. Having visible default seems like the right thing to do. But nevertheless, if you have a UI file that in GTK4 looks entirely different than it did in GTK3, you may want to go and check that there's not any widgets in there that were always hidden before because nobody set the visible property to true and that are now showing up.

Another surprise is that as already mentioned, gtk_container has gone. And that is probably a bit more annoying than the visibility of widgets. In particular, if you are creating your UI manually calling GTK container on a lot of different containers on windows and boxes, you have to replace those calls by the non-generic per widget occupy that we have replaced it with such as for GTK box, for example, you would call GTK box apparent. Or Windows, GTK_windows. And that's admittedly painful and annoying. But it's a one-time change. The good news is, you're not affected by this if you're using UI for the UI. It works just the same as before. There are some other changes in UI files that are -- that we'll need you to adapt to. For instance, the change from child properties to layout properties.

But the good news there is that GTK4 build-builder-tool can help you. If you pass the 3 to 4 option to the command, it will try to read the gtk.ui file and try to convert it to GTK4. This is an imperfect tool. You should always verify the output that it uses and verify the warning that it spits out. But we've run it over all the files several times in development, but it generally works fine. That's something to keep in mind as you try to port.

That being said, it's time to start talking about what it actually takes to port a widget. If you have a custom widget, there's maybe four -- four main things that such a widget should do. It probably needs to draw something, otherwise why would you have a widget? And more often than not, it will have some child widgets that need to be layed out and sized and positioned. And you will also want to handle input and probably connect to other parts of your application by triggering actions or communicating changes. So, that's the four topics I will work through in the rest of the presentation. And I'll try to do some live demos.

Wish me luck for that.

Starting with drawing. That's one of the areas where we first made radical changes as we started developing GTK4. But changing our target. We're no longer targeting Cairo, but instead targeting OpenGL or Vulkan. And part of that change, changing from a more or less immediate style to a retained mode style. In GTK3, you have a context and you draw your original content on it. And in GTK4, we are maintaining a graph that you can render notes to and then you can take a graph and pass it either to a renderer that would translate it into OpenGL or do something else with it, process it in a didn't way. Maybe save it to a file. And you will open it again later.

So, there's a lot of power that comes with that. In the GTK widget API, this change is reflected in the change from the draw to the snapshot. If you look at the draw on the top, we pass a context in there -- that Cairo context passes the pixels as they draw the content. And the GTK4 occupy at the bottom, we pass in this auxiliary GTK snapshot widget which adds them to the graph as widgets render their content.

There's, of course, a lot of different content that is important in the HTK window. All the style information gets translated into render notes. That's done by GTK itself. GTK will add those render notes to the graph before calling the snapshot function so every widget automatically complies with the CSS rendering drawing model without any extra work on your part. So, the respondent of your custom widget is to put it in the snapshot implementation.

And if you're doing that, you want to call the convenience occupy that GTK has for done that. For example, gtk_snapshot append texture, or gtk_S.N.A.P.shot_append_layout if you want to do the layout. Or Cairo for a Cairo drawing and need context for that.

I have a demo that shows that snapshot function. But before that, I wanted to briefly stop and say if you just need some self-contained Cairo drawing, the gtk_drawing_area_set_draw_func. You will need to call GTK_drawing_area, set_draw_func instead of connecting to the signal handle. Everything else is just the same. You'll get a Cairo context and you can just draw with Cairo as usual.

This, of course, also still GtkGLArea if you want to do drawing with OpenGL. Nothing has changed in that so I will skip over it in this presentation. Yeah. Now leave the safety of a slide deck and try to do a live demonstration. Wish me luck.

hope you can see my screen now. I'll switch down here. So, let's see. Talk about drawing. So, I've prepared a little drawing demo here and try to point out the interesting parts here. So, in our widget class init function from my demo widget, you can see I'm overriding and implementing this snapshot func here. That's the only thing that I need for a widget that does custom drawings. Things have really become a lot simpler than they used to be. And up a little bit, here's my snapshot function.

As you can see, it gets the snapshot object here and then we call GTK_snapshot_append_color a few times to produce render notes. That's all there is to this. So, I run this note. And as you can see, there's our four differently-colored lots. And they resize if I resize the window. So, that's all as expected. I will now bring up the GTK inspector, control shift I. And what I wanted to do here is show you that there's a recorder tab in the inspector which lets you gain some insight into the graph by basically recording the frames as GTK is looking at them. GTK caches and generates only when necessary. And now I have recorded frames here. And as you can see, we have a tree of one does. And up here, the first few rows, a CSS background node where GTK automatically translated our CSS style into render nodes. And then you can see the RenderNode from the snapshot function. There's the four colors that gave it. That is interesting and very useful if you want to go back to what's going on with what's going on with your custom drawing.

Now, let me see if I can... stop sharing my screen again. Okay. hope you can see my presentation again. Let me just skip through my fallback, non-live demo. And now let's talk about layout. Because maybe your custom widget is not actually doing custom drawing. But it arranges a bunch in some way. Which is actually something that we recommend. We want everything to be a widget as far as possible. So, for example, if you need some text and we prefer that you use a label for that, a label widget, rather than draw the text yourself. With the layout. Unless you absolutely have to.

So, the main responsibility of a custom widget that arranges widgets is to do the layout work. In GTK3, you do that by implementing size_allocate. And you can still do that in GTK4. But more convenient alternative nowadays is to create a layout manager and have it do it for you. GTK comes with a number of built in layout managers for all the typical container widgets that we used to have with the GTK box layout, GTK grid layout, as a center layout and various others.

So, you can choose one of those. And layout managers can be created and attached to a widget in the widget's init function. You can also create them in the builder template and get them that way. But the easiest way to get the layout manager for a custom widget is to just set the layout manager type in your class init function by calling this enormously long function that I put here, GTK widget class set layout manager type. And then GTK will create the layout manager of this type for you. When it instantiates the widget.

Now, having a layout manager is not interesting if you don't have child widgets to arrange. So, you probably want to create child widgets as well. And usually that is done in the widgets init function. And so, you create the widget and then you need to make them a child which is done by calling GTK under widget SetParent. You'll see that in a little bit. One thing I wanted to mention before doing that is if you make the widget a child of your custom widget by call GTK_widget_set_parent. You are responsible for when your widget is going away. Usually done in the dispose function by calling GTK widget_end_parent.

Right. And I will try to share my screen again for another live demo. Okay. I hope you can see that. There it is. So, what this time, we look at a layout example. So, let's start by looking at the class init function again. As you can see, I'm calling this function with the long name here to set a layout manager type and I'm using a grid layout. One thing I wanted to briefly point out here looking at the class init function is that I'm no longer implementing snapshot. That is not necessary unless you actually want to do custom drawing. GTK will automatically snapshot your child widgets for you. So, as long as you only to want layout child widgets, you don't need to implement snapshot yourself. So, scroll up to the widget init function and you can see I'm creating a couple of child widgets for labels. And I'm calling GTK_widget_set_parent on each of them. And then I have a dispose function that undoes that by calling GTK_end_parent. Let's see what happens when I run this demo. This is very small. I hope you can see this anyway.

There's a little oops there. As you can see, my four child widgets have all appears, but on top of each other. Something went wrong here. What went wrong is we forgot to move the children each to their own cell in the grid. In GTK3, with a GTK grid widget, you would do that by setting the left attach and top attach child the properties. We don't have child properties in GTK4 them, but we have layout properties that replace that. And we can hopefully fix this in the inspector by going in here and if you look at the widget tree, there's our DemoWidget. And the four labels as children. And now, look at the details of the first label. There's a layout that in fact has the layout properties I was liking for, left-attach and top-attach. So, let me just set them for the first widget and now I can just step through the different children and set these properties.

Unless you can see now the widgets placed into different cells in the grid. So, this is more like what we were hoping for, I think. And you will leave it as an exercise to you to combine this demo with the previous demo and arrange for the labels to show up on top of the right colors. That's an exercise for the user.

And I'll go back to the presentation now. Once again, skip through my non-live fallback. Right. Time to talk about input.

And the previous areas, we've already seen the number of auxiliary objects taking over the role of widget signals. And this trend is much more pronounced in the area of input, actually. Because we use quite a few different signals there. Basically, one for each kind of event. There was a pattern press event, and a focus-in-event and a motion-notify-event signal and so on and so forth. All of these are gone in GTK4. And instead you're expected to use event controllers. I've named the corresponding event controllers. Instead of a button-press-event, you are expected to use a GtkGestureClick. And instead of the focus-in-event, use the GtkEventControllerFocus. And so on and so forth. To make the migration of this new style a little bit easier, we've brought a lot of the basic event controllers back to GTK3 so you can already use them before you start the plotting work for real.

But we couldn't quite bring all of them back. A number of them were either introduced very recently or they rely on infrastructure that just is not present in GTK3 so they could not, for example, backport the event controllers that they are now using for drag and drop in GTK4.

Yes. The unifying principle behind all these different event controllers is GTK propagates all the input events that it receives from the system in a uniform way. They are all propagated from the top of the -- the key down to what's a target widget and then back up. And then that is commonly referred to as capture bubble. So, the blue arrows here. That's when the event goes down, the capture phrase. And then the red arrows, when it propagates back up, that's the bubble phase.

And at each point along the way and each widget, those widgets event controllers get a chance to inspect the event and handle it in whichever way they choose. And I should maybe say that the way the target widget is chosen depends on the kind of event. For key events, the target widget is the current focus widget. And for point events, the target widget is hovered, basically what's currently under the pointer. All right.

And I do have another demo for this. I apologize in advance. That this demo is a little less convincing maybe than the other two. I was struggling a little bit to find an interesting demo. So, I just made a boring one.

If you look at the wasn't -- the demo widget init function here, you see that I still have two child widgets. I dropped two of them because one was too much. And I created two event_controllers here now. Those are click gestures. And then I collected a signal handler to the pressed signal. And addressed them to the two widgets. And scrolling up a little bit, here the signal handlers for those two signals and they don't do anything interesting. They just print a string so I can see that they were actually called. And when I run this...

And I click on -- you see the text appears. And the rule. And the other text appears. It's all as expected, but not very exciting. Okay. I hope you can see my presentation again. And I'll skip through my fallback here.

And so, part of the reason why this demo is not -- was not very convincing or not very interesting is that there was no real need to use the event_controller here in the first place. I could have just used a pattern widget instead and connected to the click signal. Another reason why it's not very convincing is that there was no real action to be triggered. I just printed a string. And maybe that's something we should fix. So, let's talk about actions what we can do for that. Yes.

That means, basically, GAction, which is something that has been around for a long time. Ever since we started doing app menus in GNOME 3 and needed to export actions on the session BUS. We introduced GAction for that. And back then, actions were mainly global. You were adding them, those actions directly to your GTK application or your GTK application windows since we needed global actions for the application menu. We're not doing application menus anymore, but the GAction machinery is still around. In fact, we're using it more than ever.

And at some point in doing GTK3, we had a way to attach actions to widgets that are not necessarily global by using GTK_widget_insert_action_group. And you can add that anywhere in the widget. And the widgets added this way are considered for activation when the activation happens below that widget. There's some interesting scoping that gets introduced this way.

Now, in GTK4, we added another way to add actions -- attach actions to widgets by calling GTK_widget class install and action function. Similar to the way you install properties by calling G object class in your class init function. And classes that are associated with the class in this way are available for all the instances of that widget class.

And then in the past, you often used signals for when we needed to connect an event to an action, for instance, you had a way to call on GTK widget class and binding to connect shortcuts with action signals on a widget. And basically, every place where we're using signals for that in the past, you can now use actions as well. As an example, we added at the very end of that binding API is called GTK widget class add binding action. And you can associate that with the signal. So, let's go back to that input demo one more time and see if we can actually make it a little more interesting by adding an action.

So, here's my -- here's my -- one more again. And start by looking at the class init function. And you can see at the bottom of the class init function. Calling GTK widget class install action to create two actions. We need to pass a name for the action. Actions can have arguments. Or in theory, provide a G variant type here. But not using that. I'm just passing the second argument. And the last argument here is the callback that gets called when the action is actually activated.

Scrolling up a little bit here are my two callbacks for my action. They just set a CSS class on my widget just to see that something happens. And then going further up to our -- to the signal callbacks for our event_controllers. You can see that we are now calling GTK_widget_activate_action to use our newly created actions here. And let's see what happens when we run this.

So, so far it looks just the same. Now if I click on the red label, you will see that my turn red action gets activated and the red slide gets added to my widget and transferred. And actually gone blue, similar. So, that's a lot more exciting.

Let me skip to my backups, sorry. This is basically the material I have prepared for giving you some hints of how porting from GTK3 to GTK4 will look in practice. And there's obviously a reading list here with further information if you want to actually try this out in practice. There's obviously a migration guide as part of our API documentation. And we'll try to flesh that out and make it more useful in the runup to GTK4.0. The feedback on that is very much appreciated. If anything is missing or unclear, we would like it know about it. And the GTK blog where we post some longer form material to. And there's Discourse, you can ask about not just porting to GTK4, but about GTK in general. Anything you want to talk about.

And then there's the materials for this talk, including the demos that I ran here. Made those available on GitLab earlier today. So, feel free to look over that. And I think I still have a little bit of time left so, I want to show a few more slides. Some ideas for what you want to explore after you have done your initial porting. Because GTK4 has a little bit more to offer than just the same old, same old. The first thing I wanted to mention here is media support. That's one area where GTK4 is doing I think significantly better than GTK3.

In the past, whenever we needed image data, GdkPixbuf was our go-to API. And unfortunately, GdkPixbuf's API for animated images is not very convincing and not widely used. In GTK4, we have a new image data, GtkPaintable. And it is more flexible and more useful. And Pixbuf is a GTK paintable, but you can use a web file and present it as a paintable. That's what the GtkMediaFile that I mentioned down here. GtkMediaFile has a backend to launch animated images in various formats. And our display widgets, for example, GtkPicture that I mentioned here, all accept Paintables as a source of images and will happily show animated Paintables just as well as static ones.

And I think I can maybe demo that live. And I may have to turn off my webcam sharing for that. Okay.

So, a few months ago, I wrote this little weekend project to explore GTK's media capabilities which was called video play. And it's a -- a Gtk application that is using GtkMediaFile and talking to the Screencast portal. So, it's running in a Flatpak and it can, for example, get a video stream via PipeWire from the webcam and show it in a GTK widget without any further dependencies needing to be added here. I think that's very nice and something you may want to explore once you've done your initial GTK4 port.

So, let me switch back to my presentation. And that's only one more thing I want to mention here. And that is -- oops. That's me. Scalable lists is another big topic that is worth exploring after the initial port. We do have model-based list and widgets now that recycle widgets and promise to be a lot more scalable than GTK GridView or ListBox could be. In fact this morning we merged a lot of intense work that Benjamin did over the last month to make our filtering and sorting of list models incremental so we can now re-sort a list model that has millions of items in it. And we can keep scrolling the list view and don't miss a frame while the sorting is going on in the background.

So, that is a very impressive result, I think. And something that you should probably explore. But it's worth a whole talk of its own. So, I just mentioned it here. And with this, I'll say thanks for your attention. And if there's any questions, I would be happy to answer them.

>> Oh, thank you, Matthias. The questions, you can find them in the shared notes. A Track 1 Q&A. There's a lot of questions waiting for you. You can read them yourself.

Matthias Clasen: I'm seeing that now. Let me maybe start reading the first one. As a developer new to writing GTK applications, would you start learning on the latest GTK4 or GTK3 stable? That's a good question and depends on your perspective of how soon you need results and how widely you want your application to be deployed. GTK4 doesn't currently exist. I hope it will exist before the end of the year. But even so, if you want your application to run, they will not have GTK4 for a while. From that perspective, it makes a lot of sense to stick to GTK3 for now just to get your application deployed right away. But if you're interested in exploring some of the new things I mentioned towards the end of the talk, looking at GTK3.99 and just playing with it is probably a lot more exciting and maybe interesting to do.

Okay. So, the next question is, how does one get started moving an application using PyGI and GTK3 to GTK4? I think our binding support should be in pretty good shape. We do ship files and type -- and those should work better than in GTK3 because one of the things we did as we did all these API changes was to make the API hopefully a little more uniform and easier to bind. We've emphasized properties a lot more so that everything that can reasonably be made available as an object property should be an object property now. So, that should make it easier for bindings to fully support our functionality. Yes.

Next question is what is the cross platform story for GTK4 and does the Vulkan backend help at all? Those are two questions. The first one, the cross platform story is not much better than in GTK3. We do have a Windows backend that is semi-actively maintained by a few people who actually work on Windows. So, I think it is in decent shape. But we do not really have anybody working on the OS backend until as I mentioned during the talk, until very recently Christian Herget broke down and wrote a new OSX backend. Which hopefully improves our OSX story quite a bit. That is not using Vulkan, though. That is currently using GL. And we have to see a little bit where it's going.

The Vulkan backend that we have for GTK in general is -- some are complete. But the focus over the last year has been on the backend. Really don't have enough expertise to actively push forward multiple backends at the same time. So, the GA backend is pushing to completion for GTK4. And the Vulkan one is more or less left for an exercise for the future to polish up and bring up to par again. Okay. Let's see what the next question is.

Can you elaborate on the use case of bubbling up events? That's a good question. One answer is that more or less we are following partners that you can see in other areas. Like Emmanuele said earlier today for the accessibility work, we are following what event is doing for ARIA. And for event handling, it's a similar story. We're trying to be more similar to the web than we were in the past just because everybody knows the web and we win by being similar. Because everybody knows that. And on the web events, a lot that way, they capture that bubble.

As concrete use case for the difference between interfaces is important is in things like scrolled windows where you may want the scrolled window to eagerly grab scroll events, to be able to scroll even when there are switches. And may be interested in scroll events. And these use cases are sometimes difficult to resolve satisfactorily. But having those two phases makes it at least possible. [no audio]

>> Matthias? We lost you. Cat killed the mic. Hopefully he will be back soon. It's back now.

Matthias Clasen: Oh, really? Can you hear me?

>> Yeah.

Matthias Clasen: My cat was -- I don't know what happened. Let me go back and maybe answer one or two more questions here. What do you think is the most exciting new change in GTK4 that you think deserves the most fanfare? So, that's a good question.

>> Sorry, you're a bit loud.

Matthias Clasen: Okay. I'm going to have to -- to see. Okay. So, what am I most excited about? That's an interesting question. It changed a little bit over time. If you had asked me that same question a year ago, I would have said I'm mostly excited about the constraints event controller that Emmanuele was working on back then that we have landed. But so far we haven't really embraced that much. It's there and it works. But we haven't made exciting use of it. I'm kind of excited about it for the prospect of doing a UI builder that would use it. But so far nobody has shown up to do that work.

So, my excitement currently recently is driven by the work that Benjamin has been doing on list models and scalability. Because that -- as mentioned during the talk, it's really quite impressive to see how we can now filter and sort really big lists and the UI just stays responsive. That's just not something you could do before. And I think that's -- that's an exciting thing that we should make use of in our applications.

And that sometimes requires some remodeling of the application infrastructure. All our lust widget work is based on the GList model which is a very simple container interface for just presenting a list of objects. But you need to have objects. So, for example, earlier this week, I was looking at libg with an eye towards maybe -- untying if from the GTK3 dependency and making the time zone data and the location data available in a form that can easily be dumped into one of our new list widgets. But it turns out that the G weather time zone and Gweather location structures are not actually objects currently. They are blocks and structs. We will have to change that to take advantage of this new infrastructure.

All right. Maybe one more question here. Will GTK-based apps run faster under my PinePhone when ported to GTK4? That's a good question. I don't know that I can actually give you an honest answer about that. I hope so. But I think that depends to a large degree on whether the GL infrastructure on that PinePhone works well and whether we get to use it. But I certainly hope we have the potential to make GTK work better on more platforms just by having this new rendering infrastructure. All right. There's a question here about QML. Can we have having similar to QML, but for GTK UIs? Yes, maybe. If somebody's working on that.

As I said earlier, it would be great to start working on a new UI builder where we could use constraints and maybe could -- maybe also have something that is more -- more akin to QML than what our builder, XML, currently gives us. But that's up to whoever ends up working on a new UI builder.

All right. Last question here is, what's the plan for adding an animation API? If recall correctly, that was the goal for GTK4 at some point. And, yes. That is -- that's correct. That was a goal. And it was on our list of

-- our wish list of features that we had targeted for GTK4. Back when I announced GTK suite .98 at the beginning was year, it was still on the list of things that we wanted to land. But at this point, we decided to drop it from the list because it turns out that it just needs more infrastructure work inside GTK to make -- make it scalable and make it work well that we just don't have the time to do that if we want to still release GTK4 before the end of this year. So, we decided to drop that from our list for now. And pick it back up after GTK4.0. And we'll see if what we come up then is compatible with something that we can merge into GTK4 during its life cycle or whether we need to make that a GTK 5 goal that's to be seen. I think we might have opinions on that. And if there's no additional questions, once again, thank you for your attention.

And that's it. Oh, the last question. What's the GTK4 release date? Good one. So, I think I can make it more concrete than what we've said so far. Before the end of the year. So, my goal for the 3.99 release is to have it out ideally before the end of next week. And then we'll see how many point releases we do between 3.99 and 4.0. But at that point, only major APIs should be in and should be basically -- they should be in the polished phase where we focus on writing documentations and writing examples and preparing the ground for 4.0.

And yes, there's another question. Does that mean that GTK WebKit is making progress? Yes. There is a an active porting effort to make GTK WebKit work with GTK4. And as far as I know, that is working pretty well and more or less done. Last I heard.

>> Is that all? It seems like it's all.

Matthias Clasen: I think that's all. Thanks again.

>> Thank you, Matthias, for a really, really in-depth presentation about GTK4. We will continue in 10 minutes with Martin Abente Lahaye, about flooding the desktop with learning tools. See you in 10 minutes.

[break]

>> I'm just testing the audio. Is it working okay?

>> Yep. Working.

>> Thank you.

>> Well, let's see if the bandwidth helps. If I start breaking up, just let me know and I'll disable the video.