

# SUMMARY

Scrollbar updates will be interactive, customizable, created by default on Web & Desktop platforms, and match design languages.

**Author: Kate Lovett (@Piinks)**

**Go Link: flutter.dev/go/update-scrollbars**

**Created:** 11/2020 / **Last updated:** 11/2020

Umbrella issue: <https://github.com/flutter/flutter/issues/70866>

Umbrella PR: <https://github.com/flutter/flutter/pull/71181>

* Full picture to facilitate discussion and review of split up, smaller changes
* Now: <https://github.com/flutter/flutter/pull/71664>

The large PR above is being broken up into smaller portions, so far:

1. Refactor scrollbars with existing functionality down to RawScrollbar
   1. <https://github.com/flutter/flutter/pull/71242>
2. Update Scrollbar to match new Material Design spec
3. Add track gestures (may include in 2)
4. ScrollbarTheme
5. Default instantiation on Desktop & Web
6. Sample apps

# OBJECTIVE

This document will lay out a slew of planned changes for scrollbars in the framework. The primary driving force behind these changes in improving the experience for desktop and web users, as much common functionality on these platforms is missing.

# BACKGROUND

### Interactivity

<https://github.com/flutter/flutter/issues/31753>

Scrollbars are a basic navigation tool for scrollables, especially on desktop and web. Thumbs should be draggable (currently only possible with CupertinoScrollbar), and the track of the scrollbar should receive tap gestures that page the scrollview up or down based on the relative position of the thumb to the tap gesture.

CupertinoScrollbar has a specific native behavior where the thumb gesture is only triggered by a long press gesture.

### Scalability

As Flutter expands to support more platforms, adaptive widgets like Scrollbar do not scale well. Currently, the material/Scrollbar widget will create a CupertinoScrollbar widget based on the current platform. This creates code duplication between the two widgets, particularly around the fade in-out animation and gestures. This document will also discuss design updates to these widgets, like hover and drag behaviors. As their functionality grows, they become more complicated and have more common code, making the adaptive functionality difficult to maintain and debug.

### Platform Availability

<https://github.com/flutter/flutter/issues/40107>

Scrollbars should be created by default on Desktop and Web, rather than relying on the user to instantiate them explicitly. This default behavior should be applied only to relevant scrollables. PageViews and TextFields, for example, should not follow this default Scrollbar convention.

### Theme-ability

<https://github.com/flutter/flutter/issues/61362>

All of these features should be accessible for modifying through a ScrollbarTheme. Currently, Scrollbars expose thickness and radius for the thumb. As Scrollbars grow, there is a reasonable expectation that users will want to further customize tracks, thumb colors, etc. - and have this configuration reflected across their application. The default instantiation mentioned above should also be togglable, so users are not stuck with a behavior they cannot change.

### Glossary

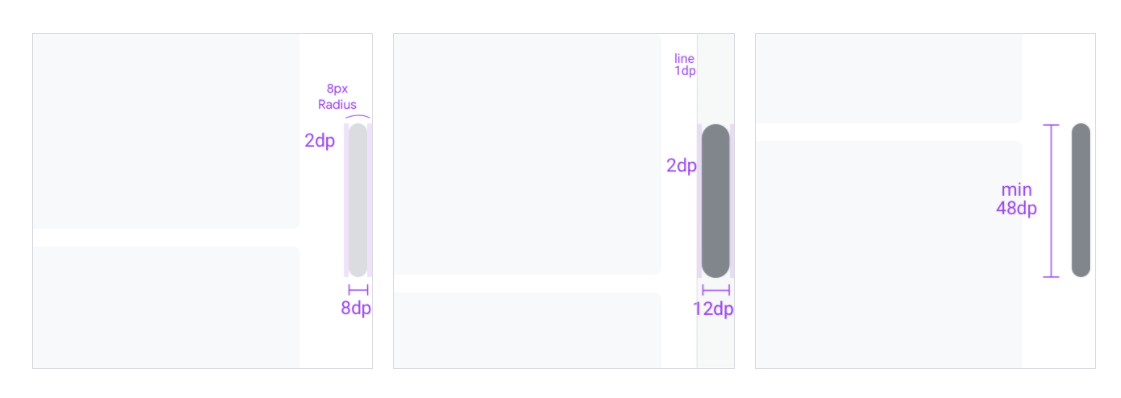
* **Scrollbar** - A material Scrollbar. This widget is currently adaptive and will return a CupertinoScrollbar when on iOS or MacOS. Executes fade in and out animations for the thumb.
* **CupertinoScrollbar** - an iOS style scrollbar. Currently the only scrollbar that supports dragging the thumb, activated by a long press to match native behavior. Executes fade in and out animations for the thumb, as well as additional animations activated by dragging the thumb.

# OVERVIEW

To meet the specifications laid out in the above Background, I would like to first refactor the existing scrollbars to have a common base class, RawScrollbar. This widget can share the common gestures (drag thumb and tap track) and animations (fade in and out) between the two while also providing a basic scrollbar for the widgets layer. This will leave the Scrollbar and CupertinoScrollbar to extend the RawScrollbar to add more animations and custom features. This will change the Scrollbar widget to no longer create a CupertinoScrollbar adaptively. The new Material Design Scrollbar is very similar to the CupertinoScrollbar (below).

After introducing the RawScrollbar widget, the Scrollbar widget should be updated to match new Material Design specifications.

These new specs are similar to the CupertinoScrollbar, with a slightly padded, rounded rect as the default behavior.

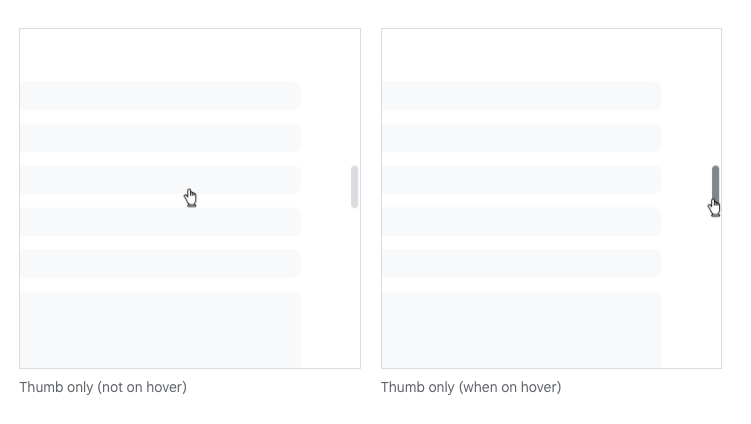


On the left is the thumb-only rendering, the right reflects the Scrollbar when a track is included.

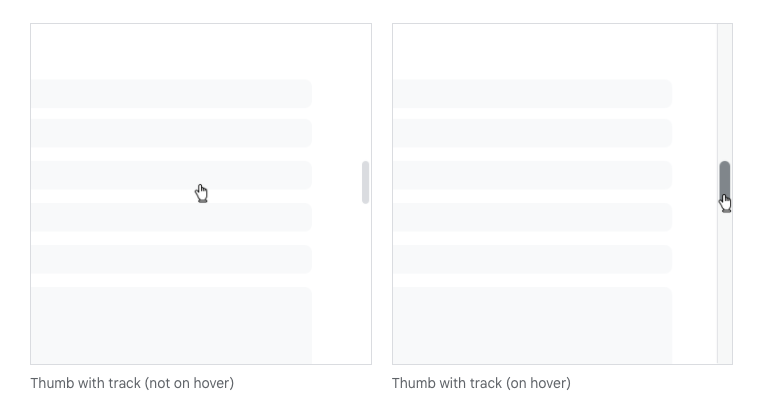
The addition of a track can be accounted for in the already existing ScrollbarPainter. This will be beneficial in gesture handling as well, since the ScrollbarPainter can determine hit testing for the different components of the scrollbar (thumb or track), as they will need individual handling for various gestures (tapping and dragging).

The track comes with its own special states, primarily that it shows on mouse hover, and the overall scrollbar is larger when including a track.

The following image shows the thumb only configuration of the Material scroll bar, with the thumb color becoming darker on hover.

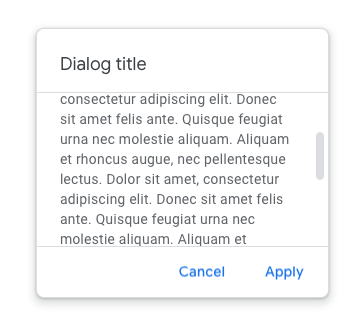


The following image shows the thumb & track configuration of the Material scrollbar, with hover triggering the appearance of the track and darkening the color of the thumb.

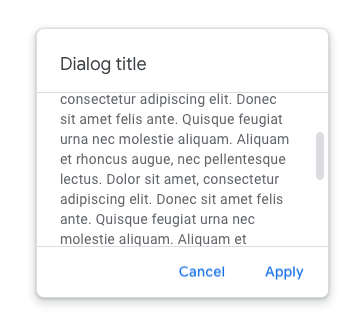


Regardless of the inclusion of the track visually, the scrollbar should always respond to the track area for hover and gesture events.

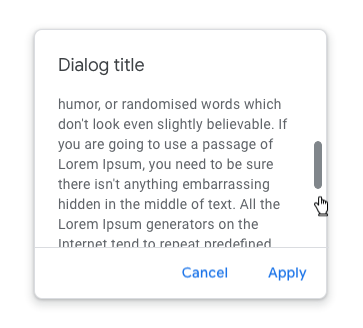
The following animation illustrates a tap gesture on the track area of a thumb-only scrollbar, which initiates a page scroll. The darkening of the thumb on hover is not restricted to hover solely over the thumb, but over the track area as well.



The following animation illustrates a tap gesture on the track area of a thumb & track scrollbar, which initiates a page scroll. Hovering immediately shows the track and makes the scrollbar thicker for interaction.



Finally, when dragging, the new Material scrollbar becomes even a little darker to represent this state. The following animation shows the transition from the hover thumb color to the drag thumb color.



Track gestures are being split out to another separate change to ease the amount of code being reviewed at a given time, this will add the ability to tap on the track area of the scrollbar to initiate a page ScrollIncrement. Since these gestures will be added to the RawScrollbar base class, they will automatically be supplied to Scrollbar and CupertinoScrollbar (<https://github.com/flutter/flutter/issues/31753>).

Next would be a ScrollbarTheme, which will allow the ability to uniformly customize scrollbars across an app. This should cover all of the existing custom parameters like thickness and radius, while also adding others like thumb color. The Material and Cupertino Scrollbars have several unique properties, so it may be beneficial to create separate themes for these scrollbars. I imagine it would be confusing if the thicknessWhileDragging did not have an effect on the Material Scrollbar, since it does not support this behavior.

These themes should also provide the ability to control the default instantiation described below. Some users may not want this behavior, so we should give them a way to opt out.

Finally, we want to provide platform-specific instantiation. On Web and Desktop platforms, it is expected that scrollable content will always show a scrollbar. Material Design also specifies that horizontal ScrollViews should always have a scrollbar, as they are less discoverable as scrollable.

Cupertino, Material and Widgets Apps each have their own ScrollBehavior. It is here where widgets like the GlowingOverscrollIndicator are created by default.

The buildViewportChrome is where the indicator is created, and where I think we should be adding our default scrollbars for Web and Desktop.

This would require a change to the signature of this function to pass the ScrollController. The controller is necessary in order to support the gestures on the Scrollbar, but we could make it optional to reduce the amount of breakage on the function signature. This will afford us the ability to create the right scrollbar for the given ScrollBehavior, with the RawScrollbar serving the widgets layer ScrollBehavior and the design language specific scrollbars available for the rest.

However, we also need to consider Scrollables that should not have default Scrollbars, like TextFields and PageViews. I imagine this could take the form of a flag on Scrollable, like excludeFromScrollbars.

### Non-goals

* We explored creating a RawScrollbar that could take any set of widgets from the user and compose them into a functional scrollbar. (Some of this exploratory work: <https://github.com/flutter/flutter/pull/70850> ) We decided not to do this, and will not be discussing it in this document.