Using the Timeline view

Contents

Note: The timeline view works with mobile apps only. Use Chrome DevTools to generate timeline events for a web app.

What is it?

The timeline view displays information about Flutter frames. It consists of three parts, each increasing in granularity.

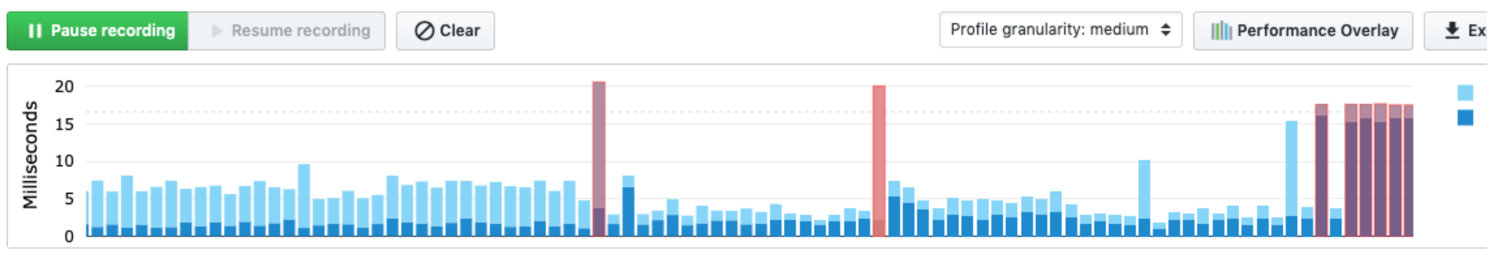
- Frame rendering chart
- Frame events chart
- CPU profiler

Note: Use a profile build of your application to analyze performance. Frame rendering times are not indicative of release performance unless your application is run in profile mode.

The timeline view also supports importing and exporting of timeline data files. For more information, see the Import and export section.

Frame rendering chart

This chart is populated with individual frames as they are rendered in your application. Each bar in the chart represents a frame. T
bars are color-coded to highlight the different portions of work that occur when rendering a Flutter frame: work from the UI threa
and work from the raster thread (previously known as GPU thread).



Clicking a bar displays additional details about that frame.

UI

The UI thread executes Dart code in the Dart VM. This includes code from your application as well as the Flutter framework. Whe
your app creates and displays a scene, the UI thread creates a layer tree, a lightweight object containing device-agnostic painting
commands, and sends the layer tree to the raster thread to be rendered on the device. Do **not** block this thread.

Raster

The raster thread (previously known as the GPU thread) executes graphics code from the Flutter Engine. This thread takes the lay
tree and displays it by talking to the GPU (graphic processing unit). You cannot directly access the raster thread or its data, but if
thread is slow, it's a result of something you've done in the Dart code. Skia, the graphics library, runs on this thread.

Sometimes a scene results in a layer tree that is easy to construct, but expensive to render on the raster thread. In this case, you r to figure out what your code is doing that is causing rendering code to be slow. Specific kinds of workloads are more difficult for t GPU. They might involve unnecessary calls to `saveLayer()`, intersecting opacities with multiple objects, and clips or shadows in specific situations.

For more information on profiling, see Identifying problems in the GPU graph.

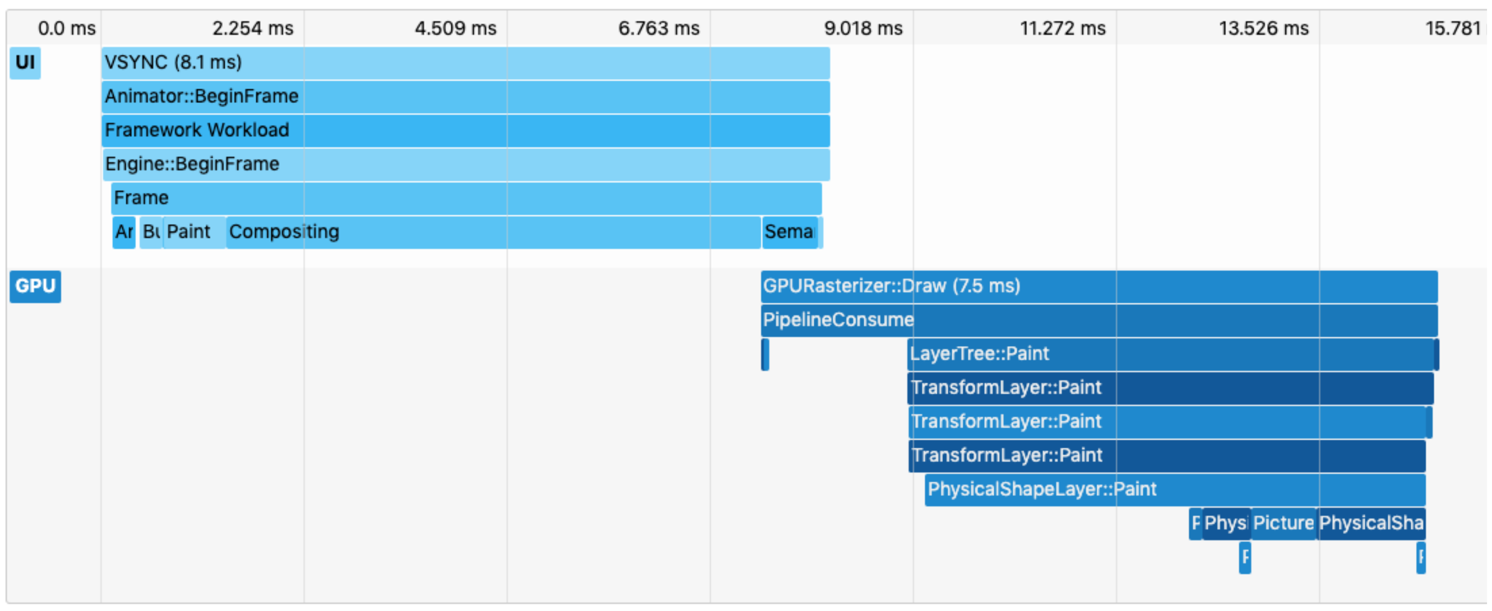
Jank

The frame rendering chart shows jank with a red overlay. A frame is considered to be janky if it takes more than ~16 ms to compl To achieve a frame rendering rate of 60 FPS (frames per second), each frame must render in ~16 ms or less. When this target is missed, you may experience UI jank or dropped frames.

For more information on how to analyze your app's performance, see Flutter performance profiling.

Frame events chart

The frame events chart shows the event trace for a single frame. The top-most event spawns the event below it, and so on. The U and GPU events are separate event flows, but they share a common timeline (displayed at the top of the frame chart). This timelir strictly for the given frame. It does not reflect the clock shared by all frames.



The flame chart supports zooming and panning. Scroll up and down to zoom in and out, respectively. To pan around, you can eith click and drag the chart or scroll horizontally. You can click an event to view CPU profiling information in the CPU profiler, describe the next section.

CPU profiler

This section shows CPU profiling information for a specific event from the frame events chart (Build, Layout, Paint, etc).

Profile granularity

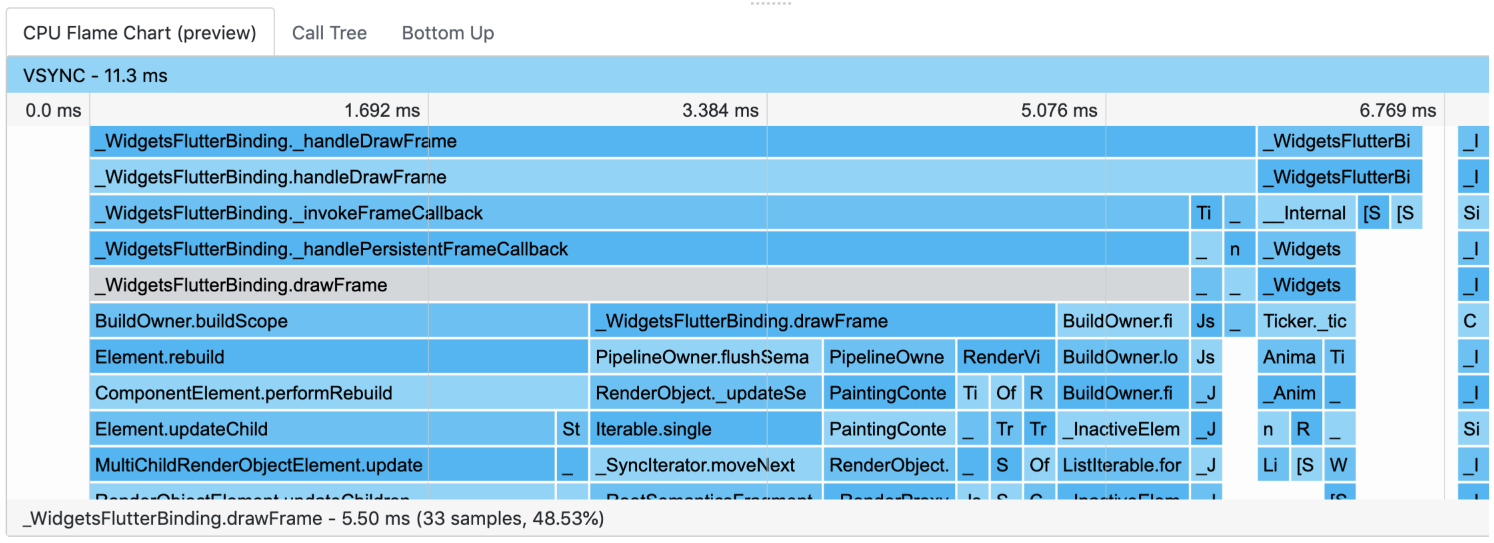
The default rate at which the VM collects CPU samples is 1 sample / 250 μs. This is selected by default on the Timeline view as "Profile granularity: medium". This rate can be modified via the selector at the top of the page. The sampling rates for low, mediur and high granularity are 1 / 50 μs, 1 / 250 μs, and 1 / 1000 μs, respectively. It is important to know the trade-offs of modifying this setting.

A **higher granularity** profile has a higher sampling rate, and therefore yields a fine-grained CPU profile with more samples. This ma also impact performance of your app since the VM is being interrupted more often to collect samples. This also causes the VM's CPU sample buffer to overflow more quickly. The VM has limited space where it can store CPU sample information. At a higher sampling rate, the space fills up and begins to overflow sooner than it would have if a lower sampling rate was used. This means you may not have access to CPU samples for frames in the beginning of the timeline.

A **lower granularity** profile has a lower sampling rate, and therefore yields a coarse-grained CPU profile with fewer samples. Howe this impacts your app's performance less. The VM's sample buffer also fills more slowly, so you can see CPU samples for a longe period of app run time. This means that you have a better chance of viewing CPU samples from earlier frames in the timeline.

Flame chart

This tab of the profiler shows CPU samples for the selected frame event (such as Layout in the following example). This chart sh be viewed as a top-down stack trace, where the top-most stack frame calls the one below it. The width of each stack frame represents the amount of time it consumed the CPU. Stack frames that consume a lot of CPU time may be a good place to look f possible performance improvements.


CPU Flame Chart (preview)
Call Tree
Bottom Up
VSYNC - 11.3 ms
0.0 ms
1.692 ms
3.384 ms
5.076 ms
6.769 ms
_WidgetsFlutterBinding._handleDrawFrame
_WidgetsFlutterBinding.handleDrawFrame
_WidgetsFlutterBinding._invokeFrameCallback
_WidgetsFlutterBinding._handlePersistentFrameCallback
_WidgetsFlutterBinding.drawFrame
BuildOwner.buildScope
Element.rebuild
ComponentElement.performRebuild
Element.updateChild
MultiChildRenderObjectElement.update
_WidgetsFlutterBinding.drawFrame
PipelineOwner.flushSema
RenderObject._updateSe
Iterable.single
_SyncIterator.moveNext
PipelineOwne
PaintingConte
RenderObject.
RenderVi
BuildOwner.fi
BuildOwner.lo
_InactiveElem
ListIterable.for
_WidgetsFlutterBi
__Internal
_Widgets
Ticker._tic
Anima
_Anim
_WidgetsFlutterBinding.drawFrame - 5.50 ms (33 samples, 48.53%)

Call tree

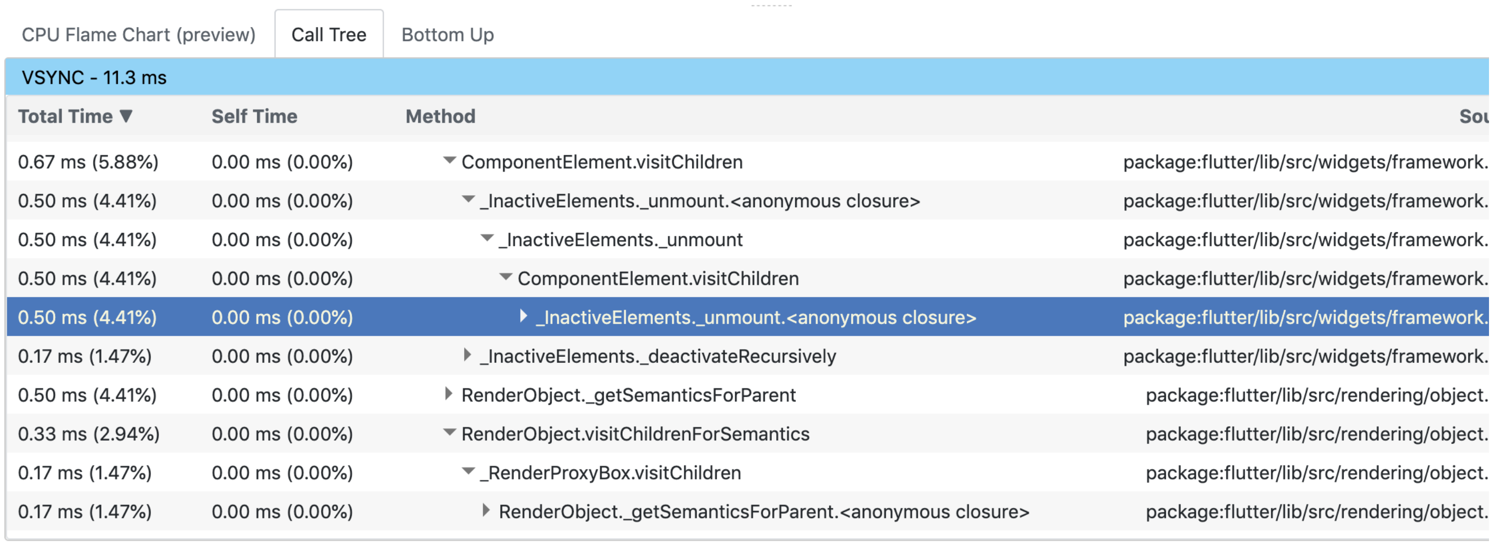
The call tree view shows the method trace for the CPU profile. This table is a top-down representation of the profile, meaning that method can be expanded to show its *callees*.

Total time
Time the method spent executing its own code as well as the code for its callees.

Self time
Time the method spent executing only its own code.

Method
Name of the called method.

Source
File path for the method call site.



Total Time ▼	Self Time	Method	Sou
0.67 ms (5.88%)	0.00 ms (0.00%)	▼ ComponentElement.visitChildren	package:flutter/lib/src/widgets/framework.
0.50 ms (4.41%)	0.00 ms (0.00%)	▼ _InactiveElements._unmount.<anonymous closure>	package:flutter/lib/src/widgets/framework.
0.50 ms (4.41%)	0.00 ms (0.00%)	▼ _InactiveElements._unmount	package:flutter/lib/src/widgets/framework.
0.50 ms (4.41%)	0.00 ms (0.00%)	▼ ComponentElement.visitChildren	package:flutter/lib/src/widgets/framework.
0.50 ms (4.41%)	0.00 ms (0.00%)	▶ _InactiveElements._unmount.<anonymous closure>	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.00 ms (0.00%)	▶ _InactiveElements._deactivateRecursively	package:flutter/lib/src/widgets/framework.
0.50 ms (4.41%)	0.00 ms (0.00%)	▶ RenderObject._getSemanticsForParent	package:flutter/lib/src/rendering/object.
0.33 ms (2.94%)	0.00 ms (0.00%)	▼ RenderObject.visitChildrenForSemantics	package:flutter/lib/src/rendering/object.
0.17 ms (1.47%)	0.00 ms (0.00%)	▼ _RenderProxyBox.visitChildren	package:flutter/lib/src/rendering/object.
0.17 ms (1.47%)	0.00 ms (0.00%)	▶ RenderObject._getSemanticsForParent.<anonymous closure>	package:flutter/lib/src/rendering/object.

Bottom up

The bottom up view shows the method trace for the CPU profile but, as the name suggests, it's a bottom-up representation of the profile. This means that each top-level method in the table is actually the last method in the call stack for a given CPU sample (in other words, it's the leaf node for the sample).

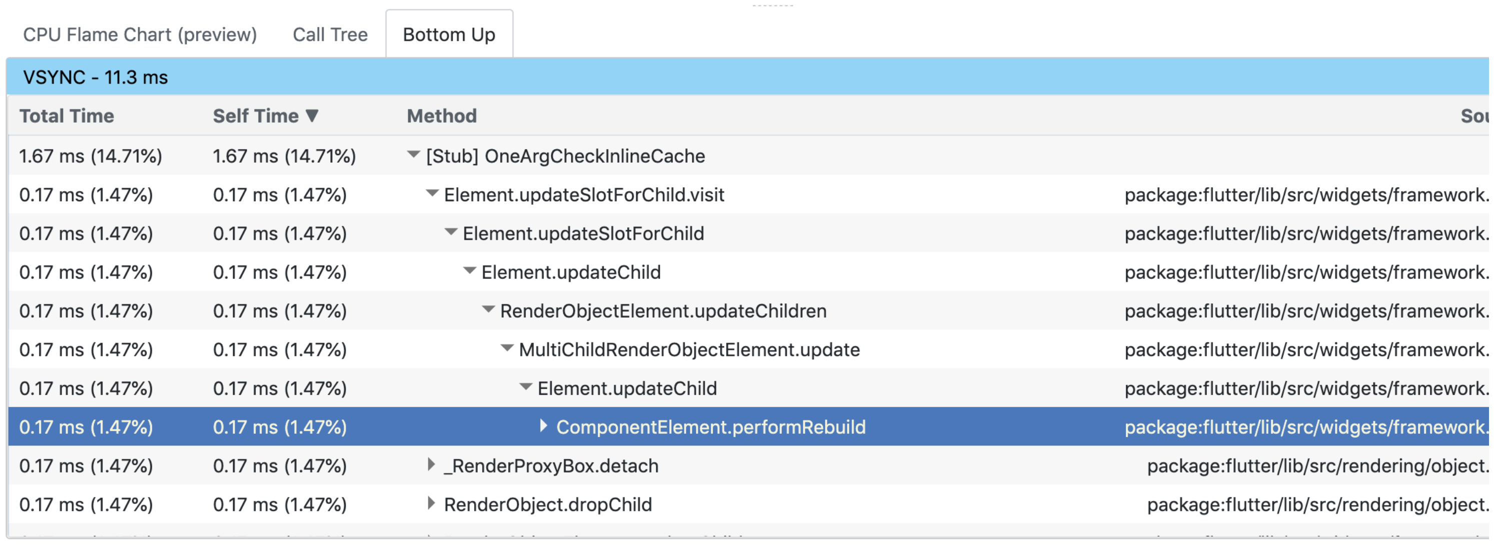
In this table, a method can be expanded to show its *callers*.

Total time
Time the method spent executing its own code as well as the code for its callee.

Self time
For top-level methods in the bottom-up tree (leaf stack frames in the profile), this is the time the method spent executing only its code. For sub nodes (the callers in the CPU profile), this is the self time of the callee when being called by the caller. In the followi example, the self time of the caller `Element.updateSlotForChild.visit()` is equal to the self time of the callee `[Stub] OneArgCheckInLineCache` when being called by the caller.

Method
Name of the called method.

Source
File path for the method call site.



Total Time	Self Time ▼	Method	Sou
1.67 ms (14.71%)	1.67 ms (14.71%)	[Stub] OneArgCheckInlineCache	
0.17 ms (1.47%)	0.17 ms (1.47%)	Element.updateSlotForChild.visit	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	Element.updateSlotForChild	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	Element.updateChild	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	RenderObjectElement.updateChildren	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	MultiChildRenderObjectElement.update	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	Element.updateChild	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	ComponentElement.performRebuild	package:flutter/lib/src/widgets/framework.
0.17 ms (1.47%)	0.17 ms (1.47%)	_RenderProxyBox.detach	package:flutter/lib/src/rendering/object.
0.17 ms (1.47%)	0.17 ms (1.47%)	RenderObject.dropChild	package:flutter/lib/src/rendering/object.

Import and export

DevTools supports importing and exporting timeline snapshots. Clicking the export button (upper-right corner above the frame rendering chart) downloads a snapshot of the current timeline state. To import a timeline snapshot, you can drag and drop the snapshot into DevTools from any page. **Note the DevTools only supports importing files that were originally exported from DevTo**