

# 分析 QML 应用程序

您可以使用 QML 探查器查找应用程序中典型性能问题的原因，例如用户界面缓慢和无响应、断断续续。典型的原因包括在太少的帧中执行过多的 JavaScript。所有 JavaScript 都必须返回，然后 GUI 线程才能继续，如果 GUI 线程未就绪，则帧将被延迟或删除。

导致类似性能问题的另一个典型原因是创建、绘制或更新不可见项，这在 GUI 线程中需要时间。

要发现 JavaScript 的过度使用，请检查动画和场景图形事件中的帧速率，查找间隙，并检查应用程序的行为是否符合预期。JavaScript 类别显示函数的运行时间，应尝试将其保持在每帧 16 毫秒以下。

若要查找由处理不可见项引起的问题，请查找丢弃的帧，并检查是否未使用过多的短绑定或每帧更新的信号处理程序。您还可以[可视化场景图形过度绘制](#)以检查场景布局并查找用户从不可见的项目，因为它们位于屏幕外部或隐藏在其他可见元素下方。


**注意：**在本节中，您将使用高级菜单项。默认情况下，这些是不可见的。要切换高级菜单项的可见性，请参阅[自定义菜单](#)。


## 使用 QML 探查器

要在 QML 探查器中监视应用程序的性能，请执行以下操作：

1. 为了能够分析应用程序，必须为项目设置 QML 调试。有关更多信息，请参见[设置 QML 调试](#)。

**注意：**要[分析设备上的应用程序](#)，必须在这些设备上安装 Qt 库。

2. 选择“**分析**> **QML 探查器**”以分析当前应用程序。
3. 选择“ (**开始**)”按钮以从 QML 探查器启动应用程序。

**注意：**如果数据收集未自动启动，请选择  (**启用分析**) 按钮。

当您开始分析应用程序时，将启动该应用程序，并且 QML 探查器立即开始收集数据。这由“**已用**”字段中运行的时间指示。

将收集数据，直到您选择“**启用分析**”按钮。数据收集需要时间，因此，在显示数据之前可能会有延迟。

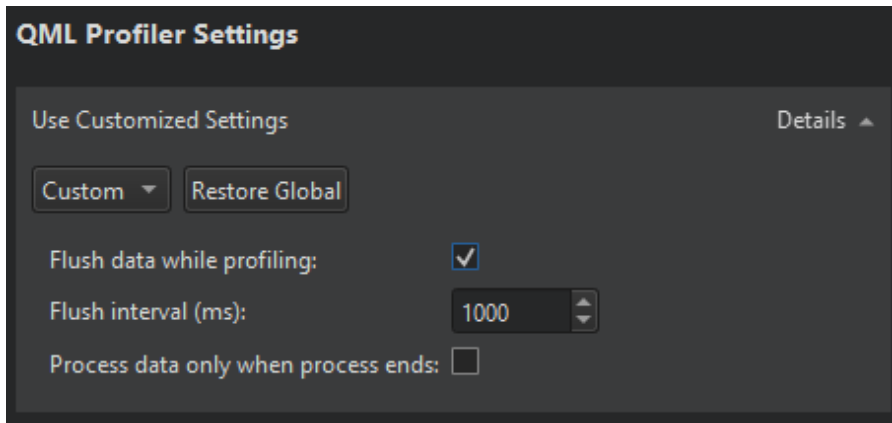
不要使用应用程序命令退出应用程序，因为当您选择“**启用分析**”按钮时，数据将发送到 QML 探查器。应用程序继续运行几秒钟，之后它会自动停止。如果退出应用程序，则不会发送数据。

**QML 跟踪**。您还可以将保存的数据提供给别人进行检查或加载他们保存的数据。

## 指定刷新设置

您可以为所有项目全局指定 QML 探查器的刷新设置，也可以为每个项目单独指定刷新设置。若要指定全局设置，请选择“**编辑>首选项**”> **Analyzer**”。

要为特定项目指定自定义 QML 探查器设置，请选择“**项目>运行**”，然后在“**QML 探查器设置**”中选择“**自定义**”。若要还原项目的全局设置，请选择“**还原全局**”。



选中“**分析时刷新数据**”复选框以定期刷新数据，而不是在分析停止时刷新所有数据。这样可以节省目标设备上的内存，并缩短停止分析与显示数据之间的等待时间。

在“**刷新间隔**”字段中，设置刷新间隔（以毫秒为单位）。间隔越短，刷新数据的频率就越高。间隔越长，目标应用程序中必须缓冲的数据就越多，这可能会浪费内存。但是，刷新本身需要时间，这可能会扭曲性能分析结果。

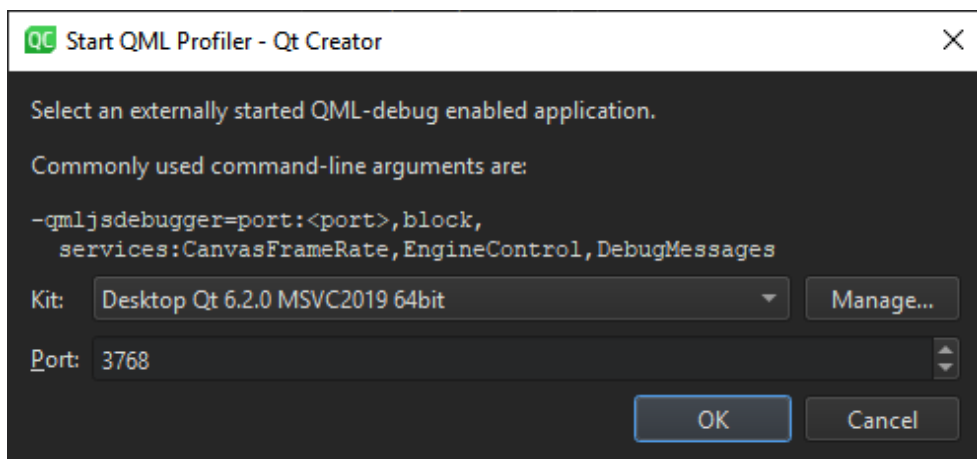
如果您有多个 QML 引擎，并且希望将所有这些引擎生成的数据聚合到一个跟踪中，请选中“**仅在进程结束时处理数据**”复选框。否则，当其中一个引擎停止时，分析将停止。

## 附加到正在运行的Qt快速应用程序

You can profile Qt Quick applications that are not launched by Qt Design Studio. However, you must enable QML debugging and profiling for the application in the project build settings. For more information, see [Setting Up QML Debugging](#).

To attach to waiting applications:

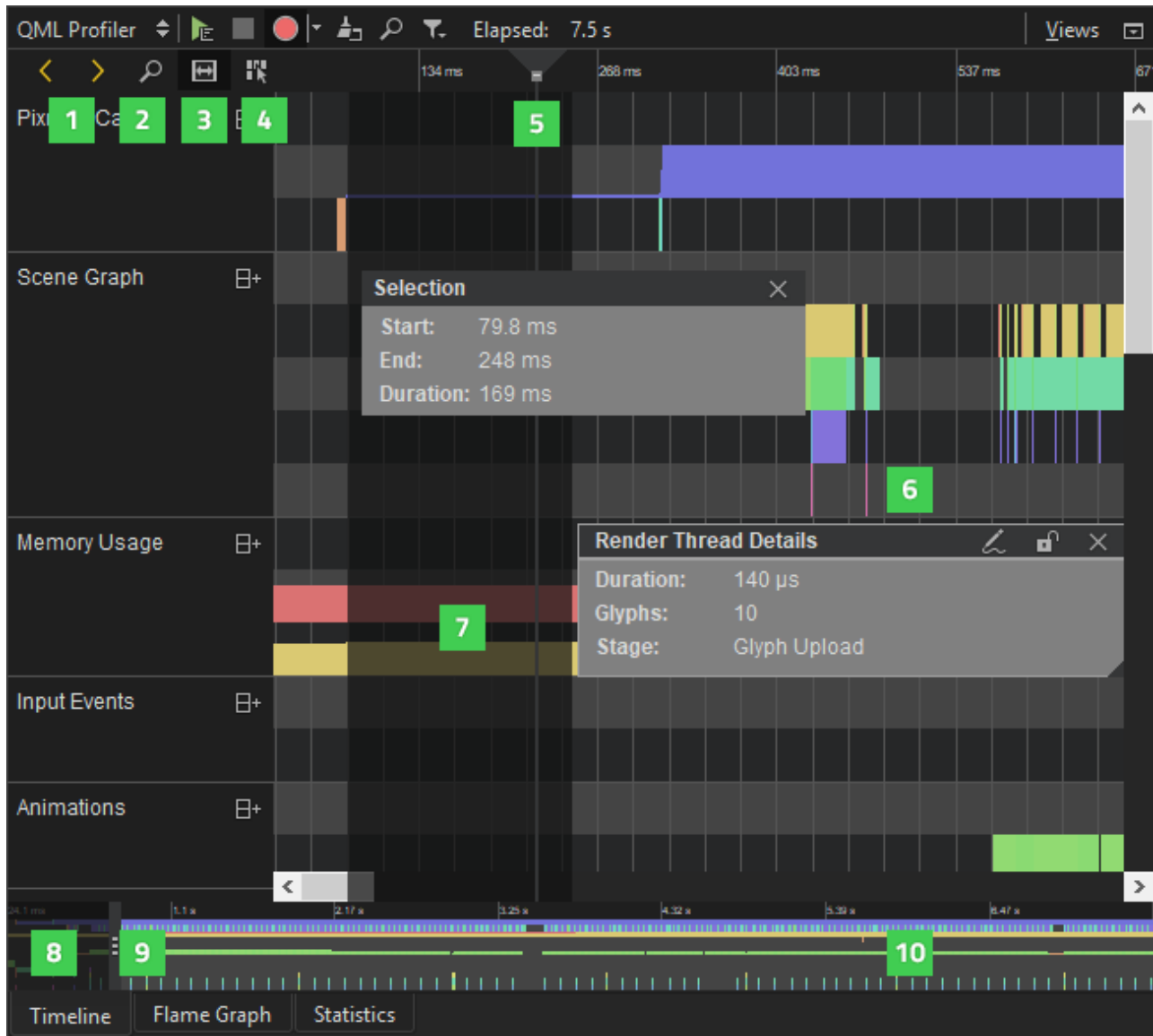
1. Select **Analyze > QML Profiler (Attach to Waiting Application)**.



4. Select **OK**.

## Analyzing Collected Data

The **Timeline** view displays graphical representations of QML and JavaScript execution and a condensed view of all recorded events.



Each row in the timeline (6) describes a type of QML events that were recorded. Move the cursor on an event on a row to see how long it takes and where in the source it is being called. To display the information only when an event is selected, disable the **View Event Information on Mouseover** button (4).

The outline (10) summarizes the period for which data was collected. Drag the zoom range (8) or click the outline to move on the outline. You can also move between events by selecting the **Jump to Previous Event** and **Jump to Next Event** buttons (1).

Select the **Show Zoom Slider** button (2) to open a slider that you can use to set the zoom level. You can also drag the zoom handles (9). To reset the default zoom level, right-click the timeline to open the context menu, and select **Reset Zoom**.

Click the time ruler to add vertical orientation lines (5) to the timeline.

events. Select the **Select Range** button (3) to activate the selection tool. Then click in the timeline to specify the beginning of the event range. Drag the selection handle to define the end of the range. The length of the range indicates the frame rate of the event.

You can use event ranges also to measure delays between two subsequent events. Place a range between the end of the first event and the beginning of the second event. The **Duration** field displays the delay between the events in milliseconds.

To zoom into an event range, double-click it.

To narrow down the current range in the **Timeline**, **Statistics**, and **Flame Graph** views, right-click the range and select **Analyze Current Range**. To return to the full range, select **Analyze Full Range** in the context menu.

To remove an event range, close the **Selection** dialog.

## Understanding the Data

Generally, events in the timeline view indicate how long QML or JavaScript execution took. Move the mouse over them to see details. For most events, they include location in source code, duration and some relevant parts of the source code itself.

You can click on an event to move the cursor in the code editor to the part of the code the event is associated with.

The following types of events are displayed in the timeline view on one or several rows. The availability of event types depends on the version of Qt the application was compiled with and the version of Qt Quick it is using.

Event Category	Description	Minimum Qt Version	Qt Quick Version
<b>Pixmap Cache</b>	Displays the general amount of pixmap data cached, in pixels. In addition, displays a separate event for each picture being loaded, with specifics about its file name and size.	Qt 5.1	Qt Quick 2
<b>Scene Graph</b>	Displays the time when scene graph frames are rendered and some additional timing information for the various stages executed to do so.	Qt 5.1	Qt Quick 2
<b>Memory Usage</b>	Displays block allocations of the JavaScript memory manager. Generally, the memory manager will reserve larger blocks of memory in one piece and later hand them out to the application in smaller bits. If the application requests single blocks of memory surpassing a certain size, the memory manager will allocate those separately. Those two modes of operation are shown as events of different colors. The second row displays the actual usage of the allocated memory. This is the amount of JavaScript heap the application has actually requested.	Qt 5.4	Qt Quick 2
<b>Input Events</b>	Displays mouse and keyboard events.	Qt 4.7.4	Qt Quick 1 or Qt Quick 2
<b>Painting</b>	Displays the time spent painting the scene for each frame.	Qt 4.7.4	Qt Quick 1
<b>Animations</b>	Displays the amount of animations that are active and the frame rate that they are running at. Information about render thread animations is displayed for applications that are built with Qt 5.3 or later. Render thread animations are shown in a separate row then.	Qt 5.0 (Qt 5.3)	Qt Quick 2
<b>Event Compiling</b>	Displays the time spent compiling the QML files.	Minimum Qt 4.7.4	Qt Quick 1

<b>Creating</b>	Displays the time spent creating the elements in the scene. In Qt Quick 2, creation of elements takes place in two stages. The first stage is for the creation of the data structures, including child elements. The second stage represents the completion callbacks. Not all elements trigger completion callbacks, though. The stages are shown as separate events in the timeline. For Qt Quick 2 applications compiled with versions of Qt before 5.2.1 only the creation of top-level elements is shown, as single events.	Qt 4.7.4 (Qt 5.2.1)	Qt Quick 1 or Qt Quick 2
<b>Binding</b>	Displays the time when a binding is evaluated and how long the evaluation takes.	Qt 4.7.4	Qt Quick 1 or Qt Quick 2
<b>Handling Signal</b>	Displays the time when a signal is handled and how long the handling takes.	Qt 4.7.4	Qt Quick 1 or Qt Quick 2
<b>JavaScript</b>	Displays the time spent executing the actual JavaScript behind bindings and signal handlers. It lists all the JavaScript functions you may be using to evaluate bindings or handle signals.	Qt 5.3	Qt Quick 2
<b>Quick3D</b>	Displays the time spent rendering Qt Quick 3D frames, timing information for frame preparation and synchronization, particle system update times and particle update count, as well as texture and mesh memory allocations and memory consumption.	Qt 6.3	Qt Quick 3D

## Analyzing Scene Graph Events

In order to understand the scene graph category, it's important to understand how the Qt Quick scene graph works. See [Qt Quick Scene Graph](#) and [Qt Quick Scene Graph Default Renderer](#) for a detailed description. The following events are reported in the **Scene Graph** category. Not all events are generated by all render loops. In the Windows and Basic render loops everything runs in the same thread and the distinction between GUI thread and render thread is meaningless.

If you set the environment variable QSG\_RENDER\_TIMING, you get a textual output of similar, but slightly different timings from the application being profiled. For easier orientation, the differences are listed below.

Event Type	Thread	Render Loop Types	Label in output of QSG_RENDER_TIMING	Description	Caveats
<b>Polish</b>	GUI	Threaded, Basic, Windows	polish	Final touch-up of items before they are rendered using <code>QQuickItem::updatePolish()</code> .	Versions of Qt prior to Qt 5.4 record no polish times for the basic render loop and incorrect ones for the windows render loop.
<b>GUI Thread Wait</b>	GUI	Threaded	lock	Executing slots connected to the <code>QQuickWindow::afterAnimating()</code> signal and then locking the render thread's mutex before waiting on the same mutex at <b>GUI Thread Sync</b> . If this starts long before <b>Render Thread Sync</b> , there is <i>free</i> time in the GUI thread you could be using for running additional QML or JavaScript.	None
Event Type	Thread	Render Loop Types	Label in output of QSG_RENDER_TIMING	Description	Caveats

Sync				blocked, waiting for the render thread.	
Animations	GUI	Threaded, Windows	animations	Advancing animations in the GUI thread. The basic render loop does not drive animations in sync with the rendering. This is why no animation events will be shown when using the basic render loop. Watch the <b>Animations</b> category to see animation timing in this case.	None
Render Thread Sync	Render	Threaded, Basic, Windows	Frame rendered ... sync	Synchronizing the QML state into the scene graph using <code>QQuickItem::updatePaintNode()</code> .	None
Render	Render	Threaded, Basic, Windows	Frame rendered ... render	Total time spent rendering the frame, including preparing and uploading all the necessary data to the GPU. This is the <i>gross</i> render time. Do not confuse it with the <i>net</i> <b>Render</b> time below.	With versions of Qt prior to Qt 5.5, the gross render time and the below breakup of render times may be misaligned by some microseconds due to different, unsynchronized timers being used to measure them. For example <b>Render Preprocess</b> might seem to start before <b>Render Thread Sync</b> is finished.
Swap	Render	Threaded, Basic, Windows	Frame rendered ... swap	Swapping frames after rendering.	The output of swap times triggered by setting <code>QSG_RENDER_TIMING</code> is incorrect for the basic render loop and versions of Qt prior to Qt 5.4. The QML profiler shows the correct swap times.
Render Preprocess	Render	Threaded, Basic, Windows	time in renderer ... preprocess	Calling <code>QSGNode::preprocess()</code> on all nodes that need to be preprocessed. This is part of the gross <b>Render</b> step.	May not be properly aligned with <b>Render</b> with versions of Qt prior to Qt 5.5.
Render Update	Render	Threaded, Basic, Windows	time in renderer ... updates	Iterating and processing all the nodes in the scene graph to update their geometry, transformations, opacity, and other state. In the <b>Render Thread Sync</b> stage, each node is updated separately with state from the GUI thread. In <b>Render Update</b> , all the nodes are combined to create the final	May not be properly aligned with <b>Render</b> with versions of Qt prior to Qt 5.5.
Event Type	Thread	Render Loop Types	Label in output of <code>QSG_RENDER_TIMING</code>	Description	Caveats

<b>Render Bind</b>	Render	Threaded, Basic, Windows	time in renderer ... binding	Binding the correct framebuffer for OpenGL rendering. This is part of the gross <b>Render</b> step.	May not be properly aligned with <b>Render</b> with versions of Qt prior to Qt 5.5.
<b>Render Render</b>	Render	Threaded, Basic, Windows	time in renderer ... rendering	The actual process of sending all the data to the GPU via OpenGL. This is part of the gross <b>Render</b> step.	May not be properly aligned with <b>Render</b> with versions of Qt prior to Qt 5.5.
<b>Material Compile</b>	Render	Threaded, Basic, Windows	shader compiled	Compiling GLSL shader programs.	None
<b>Glyph Render</b>	Render	Threaded, Basic, Windows	glyphs ... rendering	Rendering of font glyphs into textures.	Versions of Qt prior to Qt 5.4 report incorrect times for these events.
<b>Glyph Upload</b>	Render	Threaded, Basic, Windows	glyphs ... upload	Uploading of glyph textures to the GPU.	Versions of Qt prior to Qt 5.4 report incorrect times for these events.
<b>Texture Bind</b>	Render	Threaded, Basic, Windows	plain texture ... bind	Binding a texture in the OpenGL context using glBindTextures.	None
<b>Texture Convert</b>	Render	Threaded, Basic, Windows	plain texture ... convert	Converting the format and downscaling an image to make it suitable for usage as a texture.	None
<b>Texture Swizzle</b>	Render	Threaded, Basic, Windows	plain texture ... swizzle	Swizzling the texture data on the CPU if necessary.	None
<b>Texture Upload</b>	Render	Threaded, Basic, Windows	plain texture ... upload / atlastexture uploaded	Uploading the texture data to the GPU.	None
<b>Texture Mipmap</b>	Render	Threaded, Basic, Windows	plain texture ... mipmap	Mipmapping a texture on the GPU.	None
<b>Texture Delete</b>	Render	Threaded, Basic, Windows	plain texture deleted	Deleting a texture from the GPU that became unnecessary.	None

## Analyzing Qt Quick 3D Events

The following is the list of events for Qt Quick 3D. Each rendered frame consists of synchronize, prepare, and render phases, which are done in that order. Synchronize happens in scene graph synchronizing phase, while prepare and render happen in scene graph rendering phase.

Setting the environment variable will also give additional textual output of render call counts of different rendering passes. These call counts are summed up in the Render Frame event. `QSG_RENDERER_DEBUG=render`

Event Type	Thread	Description
<b>Render Frame</b>	Render Thread	Render time of a frame. Also shows the number of draw calls.



		loaded at that time.
<b>Synchronize Frame</b>	Render	Synchronize time of a frame. Synchronize takes care of updating backend values from the frontend. Also manages shared resources between Qt Quick Scene Graph and Qt Quick 3D.
<b>Mesh Load</b>	Render	Load time of a mesh. Shows total memory usage of all meshes. Also shows unloads.
<b>Custom Mesh Load</b>	Render	Load time of a custom mesh. Shows total memory usage of all meshes. Also shows unloads.
<b>Texture Load</b>	Render	Load time of a texture. Shows total memory usage of all textures. Also shows unloads.
<b>Generate Shader</b>	Render	Time for generating a shader for a material.
<b>Load Shader</b>	Render	Time for loading a built-in shader.
<b>Particle Update</b>	GUI	Update time of a particle system. Shows the number of particles updated.
<b>Mesh Memory Consumption</b>	Render	Shows a bar view of total mesh memory consumption.
<b>Texture Memory Consumption</b>	Render	Shows a bar view of total texture memory consumption.

## Viewing Statistics

The **Statistics** view displays the number of times each binding, create, compile, JavaScript, or signal event is triggered and the average time it takes. This allows you to examine which events you need to optimize. A high number of occurrences might indicate that an event is triggered unnecessarily. To view the median, longest, and shortest time for the occurrences, select **Extended Event Statistics** in the context menu.

Click on an event to move to it in the source code in the code editor.

Location	Type	Time in Percent	Total Time	Self Time in Percent	Self Time	Calls	Mean Time	Details	
<program>		100 %	304 ms	0.00 %	0 ns	1	304 ms	Main program	
clocks.qml:59	Creating	78.51 %	238 ms	0.15 %	466 μs	2	119 ms	QtQuick/ListV	
Clock.qml:141	Creating	77.87 %	236 ms	77.87 %	236 ms	16	14.8 ms	QtQuick/Text	
clocks.qml:0	Compiling	14.19 %	43.1 ms	12.63 %	38.3 ms	1	43.1 ms	clocks.qml	
Clock.qml:87	Handling ...	4.28 %	13 ms	0.42 %	1.28 ms	553	23.5 μs	onTriggered: c	
Clock.qml:87	JavaScript	3.86 %	11.7 ms	0.70 %	2.13 ms	553	21.2 μs	expression for	
Clock.qml:77	JavaScript	3.16 %	9.58 ms	2.21 %	6.72 ms	553	17.3 μs	timeChanged	
clocks.qml:90	Creating	2.41 %	7.3 ms	2.41 %	7.3 ms	2	3.65 ms	QtQuick/Imag	
Clock.qml:0	Compiling	1.56 %	4.73 ms	1.56 %	4.73 ms	1	4.73 ms	Clock.qml	
Clock.qml:130	Binding	0.64 %	1.96 ms	0.60 %	1.81 ms	72	27.2 μs	angle: clock.se	
clocks.qml:67	Creating	0.32 %	978 μs	0.02 %	65.1 μs	8	122 μs	Clock.qml	
Clock.qml:53	Creating	0.30 %	914 μs	0.08 %	240 μs	16	57.1 μs	QtQuick/Item	
Clock.qml:95	Creating	0.23 %	701 μs	0.23 %	701 μs	16	43.8 μs	QtQuick/Imag	
Caller	Type	Total Time	Calls	Caller Description	Callee	Type	Total Time	Calls	Callee Description
<program>		238 ms	1	Main Program	Clock.qml:141	Creating	235 ms	4	QtQuick/Text
clocks.qml:54	Creating	224 μs	1	QtQuick/Rectangle	Clock.qml:95	Creating	671 μs	4	QtQuick/Image
					Clock.qml:94	Creating	550 μs	4	QtQuick/Image
					clocks.qml:67	Creating	464 μs	4	Clock.qml
					Clock.qml:137	Creating	147 μs	4	QtQuick/Image
Timeline	Flame Graph	Statistics							

The **Callers** and **Callees** panes show dependencies between events. They allow you to examine the internal functions of the application. The **Callers** pane summarizes the QML events that trigger a binding. This tells you what caused a change



Click on an event to move to it in the source code in the code editor.

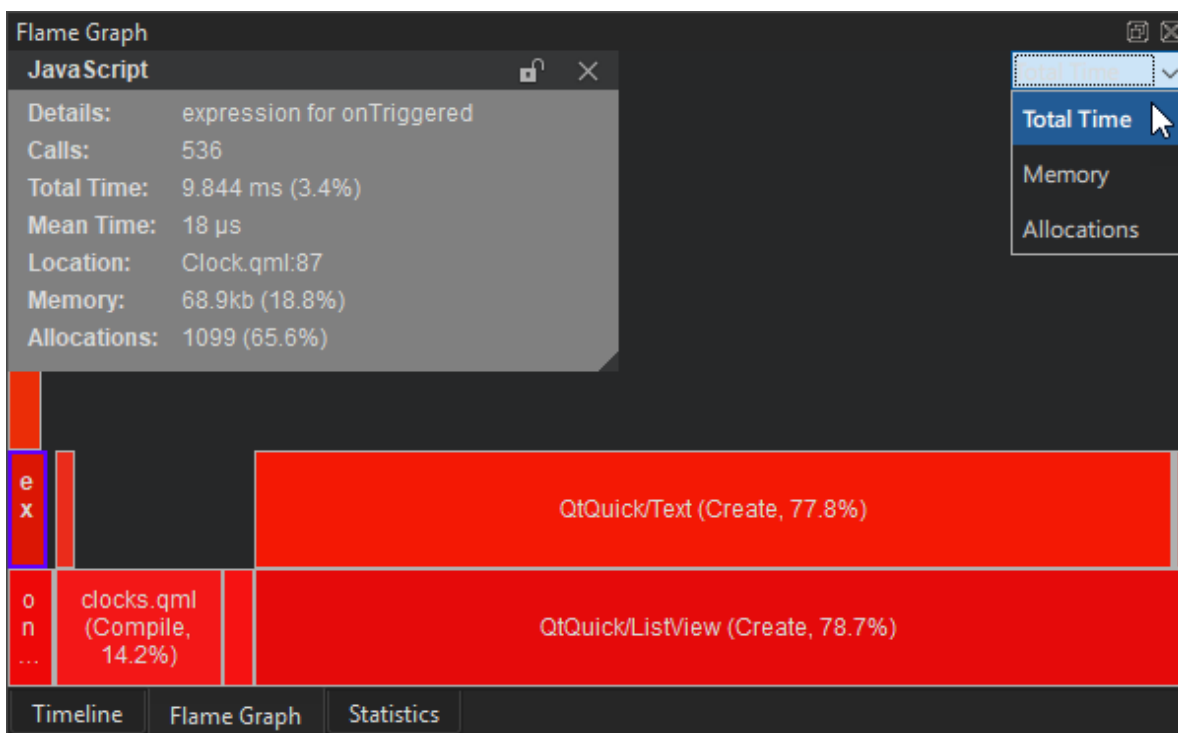
When you select an event in the **Timeline** view, information about it is displayed in the **Statistics** and **Flame Graph** views.

To copy the contents of one view or row to the clipboard, select **Copy Table** or **Copy Row** in the context menu.

JavaScript events are shown in the **Statistics** view only for applications that use Qt Quick 2 and are compiled with Qt 5.3 or later.

## Visualizing Statistics as Flame Graphs

The **Flame Graph** view shows a more concise statistical overview of QML and JavaScript execution. In the **Total Time** view, the horizontal bars show the amount of time all invocations of a certain function took together, relative to the total runtime of all JavaScript and QML events. The nesting shows which functions were called by which other ones.



To view the total amount of memory allocated by the functions, select **Memory** in the drop-down menu.

To view the the number of memory allocations performed by the functions, select **Allocations**.

Double-click an item in a view to zoom into it. Double-click an empty area in the view to zoom out again.

Unlike the **Timeline** view, the **Flame Graph** view does not show the time spans when no QML or JavaScript is running at all. Thus, it is not suitable for analyzing per frame execution times. However, it is very easy to see the total impact of the various QML and JavaScript events there.

[◀ Debugging a Qt Quick Example Application](#)

[Advanced Designer Topics >](#)



Contact Us

Company

- About Us
- Investors
- Newsroom
- Careers
- Office Locations

Support

- Support Services
- Professional Services
- Partners
- Training

Community

- Contribute to Qt
- Forum
- Wiki
- Downloads
- Marketplace

Licensing

- Terms & Conditions
- Open Source
- FAQ

For Customers

- Support Center
- Downloads
- Qt Login
- Contact Us
- Customer Success