

Profiling Function Execution

You can use the Callgrind tool included in the [Valgrind tool suite](#) to detect problems that are related to executing functions. In addition, you can load the data files generated by Callgrind into the [KCachegrind](#) profile data visualization tool for browsing the performance results.

After you download and install Valgrind tools and KCachegrind, you can use Callgrind and KCachegrind from Qt Creator.

Note: You can install and run Callgrind and KCachegrind locally on Linux. You can run Callgrind on a remote Linux machine or device from any development machine.

Building Apps for Profiling

Callgrind records the call history of functions that are executed when the application is run. It collects the number of instructions that are executed, their relationship to source lines, the relationships of the caller and callee between functions, and the numbers of such calls. You can also use cache simulation or branch prediction to gather information about the runtime behavior of an application.

Since the run-time characteristics of debug and release [build configurations](#) differ significantly, analytical findings for one build configuration may not be relevant for the other. Profiling a debug build often finds a major part of the time being spent in low-level code, such as container implementations, while the same code does not show up in the profile of a release build of the same application due to inlining and other optimizations typically done there.



Many recent compilers allow you to build an optimized application with debug information present at the same time. For example, typical options for GCC are: `-g -O2`. It is advisable to use such a setup for Callgrind profiling.

Collecting Data


To analyze applications:


1. In the **Projects** mode, select a release build configuration.
2. Select **Debug** to open the **Debug** mode, and then select **Callgrind** on the toolbar.




3. Select the  button to start the application.
4. Use the application to analyze it.
5. Select the  button to view the results of the analysis in the **Functions** view.

logging is paused.


Select  to reset all event counters.

Select  to discard all collected data.

Select  to view the data in KCachegrind. Qt Creator launches KCachegrind and loads the data into it for visualization.

Viewing Collected Data

The results of the analysis are displayed in the **Callgrind** views. You can detach views and move them around. To revert the changes, select **Views > Reset to Default Layout**.

Select **Views** to show and hide views and view titles. The **Visualization** view is hidden by default. Select  to refresh the data displayed in it when it is shown.

As an alternative to collecting data, you can select  to load an external log file into the **Callgrind** views.

Visualization

1
50
51
52

0x00000000000366c0

Callgrind

Instruction read access (lr)

Views

Callees

Callers

Callee	Calls	Cost	Caller	Calls	Cost
0x000000000a682150	1776		430,545	0x000000000000... 2	167,144,063
0x000000000067b540	1833776		41,761,512		

Functions

Function	Location	Called	Self Cost: lr	Incl. Cost: lr
0x00000000000366c0	/usr/lib/x86_64-linux-gnu/libGLX_mesa....	2	124,952,006	167,144,063
0x0000000000025f60		1714557	82,271,202	109,149,654
0x0000000000001e100	0x000000000000366c0	997245	48,451,275	143,388,157
0x0000000000001e1c0		44343	45,475,942	200,333,734
QUnicodeTables::qGetProp...		1128547	33,856,410	33,856,410
unsigned int convertCase_h...		1088577	28,303,002	60,960,312
foldCase(unsigned int, unsi...		1088494	26,123,856	97,964,460
_strchr_avx2		1171320	24,598,832	24,598,832
do_lookup_x	/usr/lib/x86_64-linux-gnu/libGLX_mesa.so.0.0.0	26203	24,312,832	26,271,414
0x00000000000021780		1341933	22,691,091	22,691,091
0x00000000000067b2e0		1833776	21,595,304	21,595,304
0x00000000000067b540		1833776	20,166,208	41,761,512
_int_malloc		114772	17,668,208	22,199,688
ucstricmp(QChar const*, Q...		48255	17,056,566	123,728,978
0x0000000000001b2b0	/usr/lib/x86_64-linux-gnu/libfontconfig....	240814	15,287,896	15,287,896

Function:

File:

Line:

Object:

Called:

Events


	Self costs (%)	Incl. costs (%)
Instruction read access (lr)	124952006 (11.54%)	167144063 (15.43%)

Enter a string in the **Filter** field to filter the results.

Move the cursor on a function in the **Functions** view for more information about it.

Double-click a function to view information about the calling functions in the **Callers** view and about the called functions in the **Callees** view.

Select  or  To move between functions in the **Callee** view.

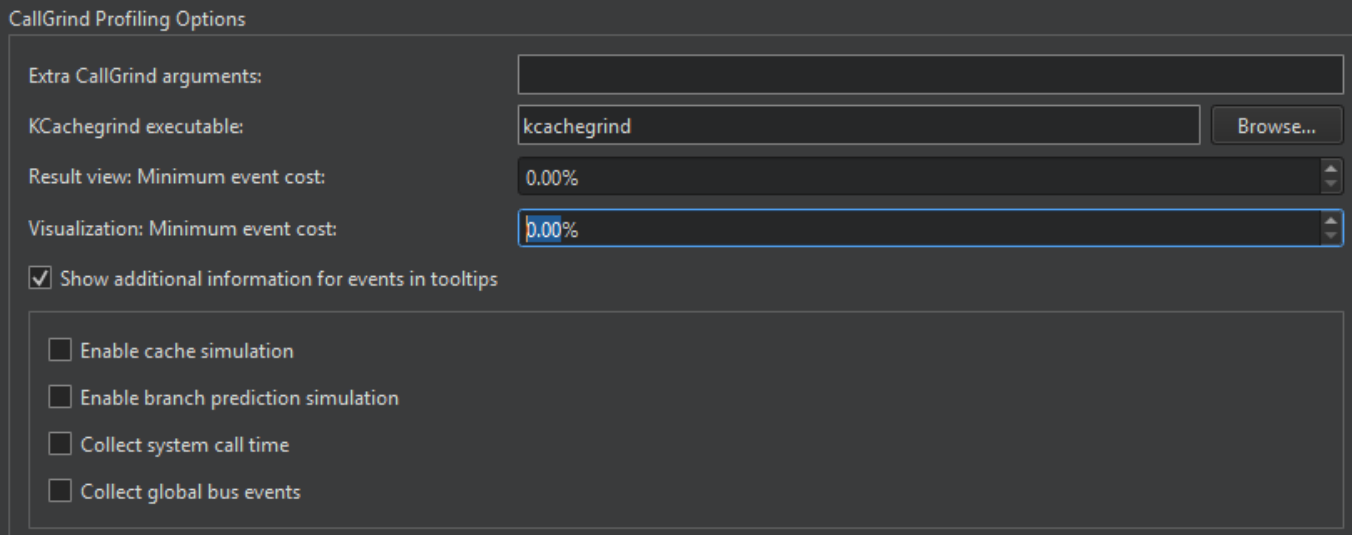
To set the cost format, select **\$**. You can view absolute or relative costs, as well as relative costs to parent. Select  to view only profiling info that originated from the project.

To properly handle recursive or circular function calls, enable cycle detection by selecting .

Selecting Profiling Options

You can specify analyzer settings either globally for all projects or separately for each project in the **run settings** of the project.

To specify global settings for Valgrind, select **Edit > Preferences > Analyzer**. The **Callgrind Profiling Options** group contains Callgrind options.



In the **KCachegrind executable** field, enter the path to the KCachegrind executable to launch.

In **Extra Callgrind arguments**, specify additional arguments for launching the executable.

In the **Result view: Minimum event cost** and **Visualization: Minimum event cost** fields, limit the amount of results the profiler presents and visualizes to increase profiler performance.

To show additional information about events in tooltips, select **Show additional information for events in tooltips**.

To collect information about the system call times, select **Collect system call time**. To collect the number of global bus events of the event type Ge that are executed, select **Collect global bus events**.

Enabling Full Cache Simulation

By default, only instruction read accesses (Ir) are counted. To fully simulate the cache, select the **Enable cache simulation** check box. This enables the following additional event counters:

- › Cache misses on instruction reads (I1mr/I2mr)
- › Data read accesses (Dr) and related cache misses (D1mr/D2mr)
- › Data write accesses (Dw) and related cache misses (D1mw/D2mw)

Enabling Branch Prediction Simulation

To enable the following additional event counters, select the **Enable branch prediction simulation** check box:

- › Number of conditional branches executed and related predictor misses (Bc/Bcm)
- › Executed indirect jumps and related misses of the jump address predictor (Bi/Bim)

© 2022 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the [GNU Free Documentation License version 1.3](#) as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.



Contact Us

Company

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

Licensing

- Terms & Conditions
- Open Source
- FAQ

Support

- Support Services
- Professional Services
- Partners
- Training

For Customers

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

Community

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