

qflog - The Manual

Version 0.95.1

Gregor Schmid
Quality First Software

Copyright © 2000 *Quality First Software*, Gregor Schmid

July 10, 2000

Copyright

The contents of this manual are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this manual except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>.

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is qfs.de code.

The Initial Developer of the Original Code is Gregor Schmid. Portions created by Gregor Schmid are Copyright (C) 1999 Quality First Software, Gregor Schmid. All Rights Reserved.

Contributor(s):

Preamble

When we set out to develop our products for *Quality First Software*¹, we knew one thing for sure: we were going to need a good logging system. The *Java* debuggers available at the time were impractical and very slow, so we wanted log messages of a quality that make debugging unnecessary. The trouble is, once you can get valuable information out of logging, you start to use it a lot, which can lead to useless noise, cluttering the important messages and to serious drawbacks in terms of CPU and memory usage. To get this under control we created the `de.qfs.lib.log` package of our free library *qflib*², which takes care of log message creation and dispatch, and the log server *qflog* which serves as the user interface to the logging system.

As a tribute to the free software community, which has produced so much great software over the years and without which *Quality First Software* would not exist, we decided to place *qflib* and *qflog* under an Open Source license and make them freely available.

We hope that *qflog* is of as much use to you as it is to us and that this manual provides all the information you need to get the best out of it. If you have any problems, questions or suggestions, please let us know at qflog@qfs.de.

One more thing: as you may already have noticed we are not native English speakers. So please, bear with us and if you feel like it, let us know about our spelling and grammatical mistakes, so that in time this manual may become a good read even for the English audience.

Gregor Schmid³, Munich, May 8 2000

¹<http://www.qfs.de>

²<http://www.qfs.de/de/projects/qflib/index.html>

³email: gs@qfs.de

Contents

1	Introduction	6
1.1	Functionality	6
1.2	Application	6
2	Installation	8
2.1	Requirements	8
2.2	Installation	8
2.3	RMI registry	9
3	Invocation	10
3.1	Invocation syntax	10
3.2	Options	10
4	Operation of the main window	12
4.1	Loading and saving log files	13
4.2	Opening a log window	13
4.3	Removing clients	13
4.4	Settings	13
4.4.1	General settings	13
5	Operation of the log window	15
5.1	General	15
5.1.1	Structure of a log window	15
5.1.2	Saving to a log file	16
5.1.3	Loading and saving the configuration	16
5.2	The table	17
5.2.1	Copying messages	18
5.2.2	Deleting messages	18

5.2.3	Extra filters in the table	18
5.2.4	Setting and jumping to marks	18
5.2.5	Incremental search	19
5.2.6	Options	19
5.3	The detail view	20
5.4	The filter tree	20
5.4.1	The structure of the tree	20
5.4.2	Setting the filter levels	22
5.4.3	Displaying defined levels	22
5.4.4	Additional filter mechanism	22
5.5	The client's log levels	23
5.5.1	The structure of the tree	23
5.5.2	Setting the levels	23
5.5.3	Displaying defined levels	24
5.5.4	Options	24
6	A sample application	25
6.1	Invocation	25
6.2	Options	25
6.3	Example uses	26
A	<i>qflog</i> and applets	28
A.1	Internet Explorer	28
A.2	Netscape	28
A.3	Plugin	29
A.4	Example applet	29

List of Figures

4.1	The main window	12
5.1	A log window	16
5.2	The table displaying the log messages	17
5.3	The detail view	20
5.4	The filter tree	21
5.5	The level tree	24

Chapter 1

Introduction

The log server *qflog* is a valuable aid in browsing log messages. By sorting messages and applying various kinds of filters one can easily locate the information one is looking for. *qflog* also helps to keep control over the amount of logging being done. This manual explains what functionality *qflog* offers, how it is installed and how to use it.

1.1 Functionality

qflog is coupled tightly with the `de.qfs.lib.log` package of *qflib*, but it is possible to view log files created by other programs, even non-*Java* ones, as long as the messages are correctly formatted. At the moment, only the format used by *qflib* is supported, but it would be easy to create import filters for other formats.

At the heart of *qflog* is the log window, which contains components for the two central functions of *qflog*:

- The message display with sorting, multiple filters, incremental search and markers.
- The interface for the parameters that are used to control the creation of log messages.

Operation of the log window is explained in chapter 4.4.1.

1.2 Application

There are three basic methods by which log messages can find their way into the log window:

- A program saves its messages to a log file, which is then read with *qflog*.

- A client/server connection is opened between *qflog* and a program.
- The log window is embedded directly into a program.

Control over log message generation may only be gained in the last two cases, since nothing is known about the source of messages read from a log file.

At the moment RMI is the only protocol supported for client/server communication. If there is a demand, CORBA or direct socket communication might be implemented as well, which would make *qflog* accessible by non-*Java* clients.

Chapter 2

Installation

2.1 Requirements

qflog comes in two flavours, one for *JDK 1.1* with *Swing 1.1* and one for *JDK 1.2* and above. Both versions are essentially the same, the only difference being the location of the collection classes.

The collection framework is a part of the *Java Foundation Classes (JFC)*. With the release of *JDK 1.2* the *JFC* were integrated into the standard *Java* class library.

Since the collection classes are far superior to their predecessors `Vector` and `Hashtable`, use of them is widespread throughout *qflog*. *SUN* has released a *JDK 1.1* compatible version of the collection classes under http://java.sun.com/beans/infobus/#DOWNLOAD_COLLECTIONS. To use *qflog* with *JDK 1.1*, the `collections.jar` archive from the above release must be present on your `CLASSPATH`.

Since *qflog* is built on top of our free *Java* library *qflib*¹, you will have to get that as well. The current release is available from our download² page.

2.2 Installation

Once you have the current versions of *qflog* and *qflib*, unpack both archives wherever you like. Please read and follow the installation instructions for *qflib*. In a typical environment the following should suffice:

JDK 1.1

Ensure that the archives `qflog_11.jar` and `qflib_11.jar` are on your `CLASSPATH`, as well as *Swing* (`swingall.jar`) and the collection classes (`collections-1.1.jar`).

¹<http://www.qfs.de/en/projects/qflib/index.html>

²<http://www.qfs.de/en/download.html>

JDK 1.2

Either add the archives `qflog_12.jar` and `qflib_12.jar` to your `CLASSPATH`, or copy them directly into the `jre/lib/ext` directory of your *JDK/JRE* installation.

There is a start script for *qflog*, named `qflog` for Unix and `qflog.bat` for Windows, in the `bin` directory of the *qflog* distribution, which you should copy to a directory on your `PATH`.

To save its configuration, *qflog* needs write access to a directory somewhere. Unless you start it with a different option (see section 3.1) *qflog* writes its files into the directory `.qflog` located in your home directory on Unix or `Personal Files` on Windows.

2.3 RMI registry

To use the client/server functionality of *qflog* you should have an RMI registry running in the background. This can be achieved with the help of the `rmiregistry` program that comes with every *JDK* or *JRE*.

Both *qflog* and its clients can create an RMI registry if necessary. The disadvantage of this approach is that other programs that register themselves with this registry will become unreachable once *qflog* or the client are terminated.

To make the use of RMI as simple as possible, all the classes involved (the so called *stubs* and *skeletons*) are included with *qflib*, so no web server is needed. As a consequence *qflib* must be on your `CLASSPATH` or in the `jre/lib/ext` directory of your *JDK* at the time that `rmiregistry` is executed.

Chapter 3

Invocation

During startup of *qflog* the following steps are executed, which may be modified or suppressed through options passed on the command line:

- If no RMI registry is running on the machine, *qflog* creates its own (see section 2.2 and section 3.1).
- Afterwards *qflog* registers itself as a log server in the RMI registry.
- Then the registry on the localhost or on remote hosts is searched for clients waiting for a log server to contact them.
- Finally the log files passed on the command line are read.

3.1 Invocation syntax

The syntax for invoking *qflog* is

```
java [java-options...] de.qfs.apps.logserver.Start  
[qflog-options...] [file...]
```

or, using the *qflog* or the *qflog.bat* script,

```
qflog [qflog-options...] [file...]
```

The files given as arguments will be read and displayed in the main window.

3.2 Options

qflog knows about the following options:

-configfile <filename>

Specifies the file where *qflog* saves its configuration, which includes locations and dimensions of its windows as well as filter informations for different

clients. The default value is `/.qflog/config` for Unix and `Personal Files\qflog\config` for Windows. The directory in which this config file resides is also used as default location for saving and restoring the configuration of a log window (see section 5.1.2).

-nocreateregistry

This option prevents *qflog* from creating its own RMI registry if it can't find one. If this option is given and no RMI registry is available, *qflog* can not be used as a log server.

-noquery

Suppresses the search for clients, both as the default behaviour or initiated with the `-query` Option.

-noserver

Prevents *qflog* from registering itself with the RMI registry. The options `-port` and `-servername` are ignored in this case. The `-query` option is not affected.

-optionfile <filename>

This option names a file or URL that contains further *qflog* options. It must use the standard property file format, i.e. contain lines of the form `name = value`. Options given on the command line override options read from the option file.

-port <port number>

Defines the port number for the RMI registry to search or to create if none already exists. The default value is 1099, the "well known port" of the RMI registry.

-query <[host][:port]>

If neither `-query` nor `-noquery` are specified, *qflog* queries the RMI registry of the host it is running on under the standard registry port 1099 for clients awaiting contact. With `-query`, which can be specified multiple times, the hosts and ports to query are given as arguments. An empty argument is equivalent to `localhost`.

-servername <name>

The name under which *qflog* registers itself in the RMI registry. Default is `"qflog"`.

-serverhost <host>

This option determines the name of the host *qflog* is running on. It is usually not necessary, but can be helpful if you experience problems with name resolution, causing trouble when clients try to connect to *qflog*.

-version

If `-version` is specified, all *qflog* does is print its version number and terminate immediately.

Chapter 4

Operation of the main window

After *qflog* starts up, the main window (figure 4.1) is displayed. It contains a list of the log files read and of the clients that are connected to the log server. A running number is used to disambiguate multiple instances of identically named clients. Beyond that, the number of messages, the time the client connected at or the file was read and the state of the client are shown. The state is one of *file*, *connected client* or *disconnected client*, represented by an icon.

From the main window, log windows for the clients and the log files can be opened, new log files can be read, stale files and clients can be removed and a few general settings can be changed.

Upon termination of *qflog*, active connections to clients are closed and the current configurations are saved, including those of the log windows (see section 3.1 on how to define the config file).

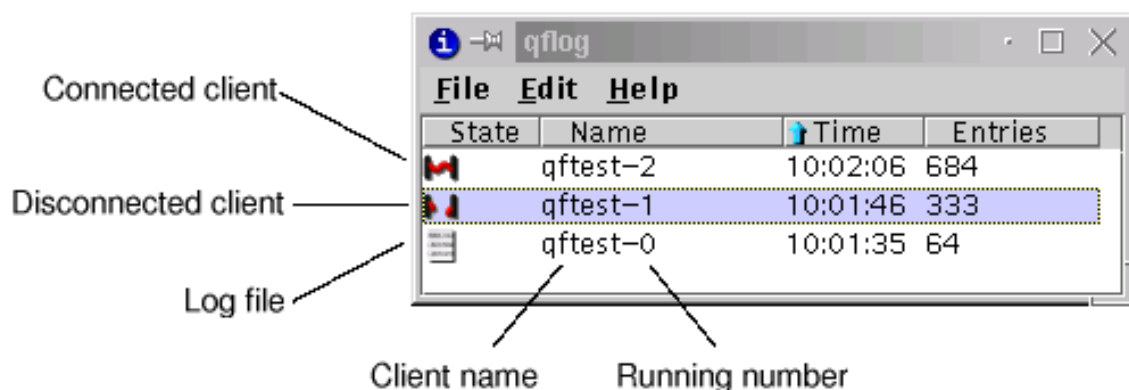


Figure 4.1: The main window

4.1 Loading and saving log files

With the **File→Open...** menu, a log file can be read via a standard file selection dialog. If the file was created with *qflog*, the client's name will be retrieved from the file, otherwise the client name "unknown" is used.

To save the log messages of the currently selected client, use the **File→Save as...** menu. The file selection dialog is used to determine the file in which all messages from the client that are available at the moment are saved. Saving only a part of the messages can be done from the log window (see section 5.1.1).

4.2 Opening a log window

The log window for the selected client can be opened either by pressing the **Return** key, or through the **Edit→Show...** menu.

4.3 Removing clients

While true clients are removed automatically from the main window, depending on *qflog*'s configuration, log files must be removed by hand. To get rid of the selected client's or log file's messages, either press **Delete** or use the **Edit→Remove client** menu.

4.4 Settings

The dialog through which *qflog*'s settings are changed is available via the **Edit→Options...** menu.

4.4.1 General settings

The following settings are used to limit the number of clients handled simultaneously, in order to keep memory usage in check. All values apply to clients only, not to log files. If one of the values is exceeded, clients are removed, where disconnected clients are removed before connected clients and older clients before younger ones.

Maximum number of clients

Maximum for the total number of clients allowed, either connected or disconnected.

Maximum number of clients per name

Limits the number of clients connected under the same name, independent of the state of the connection.

Maximum number of disconnected clients

A limit for disconnected clients, independent of their name.

Maximum number of disconnected clients per name

This value limits the number of clients that connected under the same name and are now disconnected.

Chapter 5

Operation of the log window

5.1 General

5.1.1 Structure of a log window

A log window (figure 5.1) holds up to five components, four of which can be turned on or off at will.

Messages are displayed in the table. Additionally the message that was selected last is shown in the detail view. Which of the messages are visible is determined with the filter component.

The log levels component is available only for clients, not for log files. When a client disconnects, the level component stays active but is no longer useful except to display the setting of the client's log levels before the connection was terminated.

The menus for the operation of the three main components *Messages*, *Filter* and *Level*, are named accordingly Messages, Filter and Level. Each of the components has a context menu as well, that is identical to the corresponding main menu.

There is not much to say about the status line except for the three numbers in the right corner. From left to right they stand for

- The number of messages currently visible in the table.
- The total number of messages currently available to the log view, i.e. the visible messages plus those suppressed by a filter.
- The total number of messages for this window including those that were deleted.

Except for the messages table, all components can be turned on or off via the View menu.

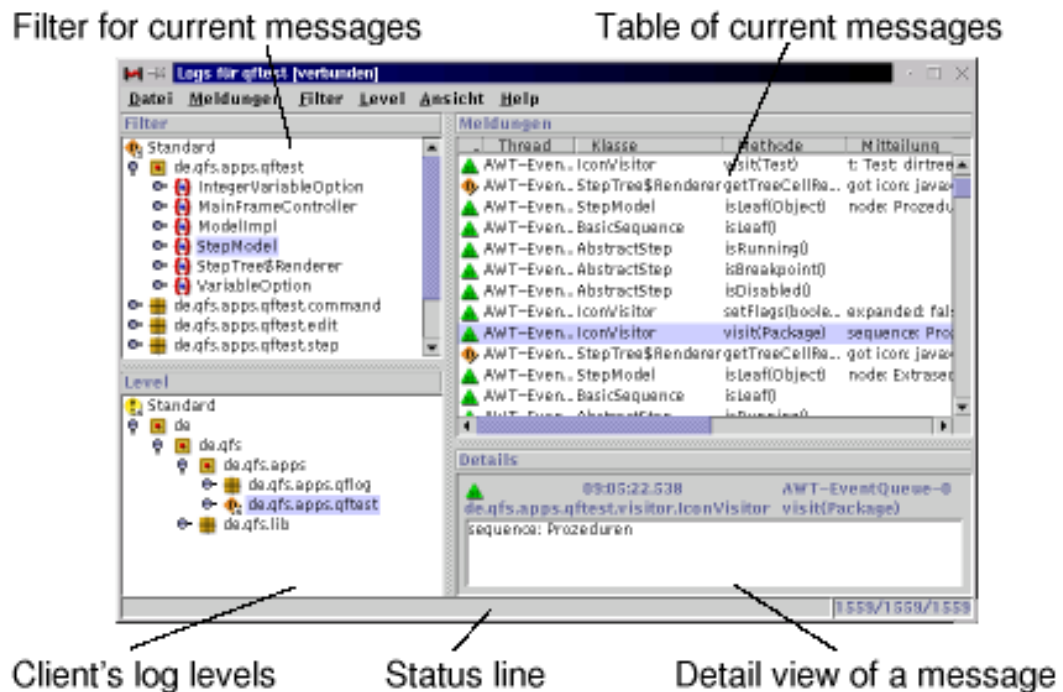


Figure 5.1: A log window

5.1.2 Saving to a log file

In the **File** menu are two menu items that will save messages from the log window into a log file. In both cases the file is selected with the standard dialog. As the names suggest, using **Save all messages as...** will save all messages, including invisible ones, in the order they were created. To save only the messages currently visible in the order imposed by the table, use **Save visible messages as...**.

5.1.3 Loading and saving the configuration

All settings for the log window and its components are saved by *qflog* in its config file when it shuts down (see chapter 3.2). Though a separate configuration is used for every distinct client name, this is often not enough, since different sets of filter settings for the same client may be useful at different times. To that end, a log window's configuration can additionally be saved via **File→Save configuration as...** and restored with **File→Restore configuration...**.

Sort order Filter Search pattern

Messages search: abstract					
...	Time	Thread	Class	Method	Message
	09:45:58.389	AWT-Even...	AbstractStep	isRunning()	
	09:45:58.393	AWT-Even...	AbstractStep	isBreakpoint()	
	09:45:58.398	AWT-Even...	AbstractStep	isDisabled()	
	09:45:58.403	AWT-Even...	IconVisitor	setFlags(boole...	expanded: false, s...
	09:45:58.408	AWT-Even...	IconVisitor	visit(Package)	sequence: Prozedu...
	09:45:58.413	AWT-Even...	StepTree\$Render...	getTreeCellRe...	got icon: javax.swi...

Level of the message

Figure 5.2: The table displaying the log messages

5.2 The table

The main part of a log window is the table displaying the messages (figure 5.2). It is always visible and cannot be turned off.

The columns of the table correspond to the elements of a log message.

Level

The level of a message is shown as an icon. How icons and levels correlate is best seen by looking at the [Filter](#) or the [Level](#) menu.

Time

The time at which the message was generated, accurate to the millisecond.

Thread

The name of the thread in which the message was generated.

Class

The class of the object or the `static` method that generated the message.

Method

The method from which the message originated.

Message

The content of the message.

The order and width of the columns can be adjusted. These settings are saved separately for each client.

The sort order of the table rows is depicted by a small blue arrow in the header of a column. One single mouse click on a column header sets the sort order according to that column, another click in the same column reverts the direction. The same effect can be achieved for the selected column by pressing [Ctrl-S](#) or via [Set sort column](#).

5.2.1 Copying messages

Using the menu items `Copy selected messages`, `Copy visible messages` and `Copy all messages`, you can copy messages into the system clipboard. Care should be taken to avoid overextending certain operating systems by copying too many messages.

The format of the copies is the same as that used for a log file.

Unfortunately the system clipboard is not implemented correctly in some *JDK* versions, *JDK 1.1* for Linux among them, so that this functionality is not available everywhere.

5.2.2 Deleting messages

Sometimes it can be useful to delete some or even all of the current messages to get a better overview or to emphasize new input from a client. The menu items `Delete all messages`, `Delete invisible messages` and `Delete visible messages` will do just that.

5.2.3 Extra filters in the table

Which messages are visible in the table is determined mainly by the filter component. An additional filter mechanism inside the table can be used to further restrict this selection.

This extra filter can be activated for the columns *Level*, *Thread*, *Class* and *Method*. It reduces the visible messages to those which have in that column a value identical to the currently selected message. As an example, if you turn on the extra filter for the *Level* column while the selected message has a level of `ERR`, only messages with this level will be displayed.

The extra filters for different columns can be combined. This way it only takes a few keystrokes to e.g. restrict the view to messages belonging to the Thread named `AWT-EventQueue-0` and originating from the class `de.qfs.lib.gui.SwingUtil`.

Toggling an extra filter is either done through the menu item `Toggle column filter` or the `Ctrl-F` key. To turn off all extra filters at once, use `Ctrl-K` or `Clear all column filters`.

5.2.4 Setting and jumping to marks

Up to 10 different marks can be set on the rows of the table in order to simplify navigation. Unfortunately there is no visual feedback yet about which marks are

set on which messages.

To set a mark, use the **Set mark** sub-menu, or the key combinations **Alt-0** through **Alt-9**. Similarly, jumping to a mark is done via the **Goto mark** sub-menu or the key combinations **Ctrl-0** through **Ctrl-9**.

It is possible, depending on the filter settings, that the message on which a mark was set is not visible when you try to jump to it. In that case the entry closest to the marked one with respect to the current sort order is used. If a marked message is removed, either explicitly or due to message numbers exceeding the limit, the mark is unset and not moved on to the next message available, since that might be confusing.

5.2.5 Incremental search

The incremental search in the message table is a vital function that is constantly active. To start a search, simply type some text on the keyboard.

Searching is always done in the current column. Of course in the *Level* column you can only search for the level. In this case, use the keys **0** through **9**, where **1** searches for level `ERR`, **9** for `DBG` and **0** for `DBGDETAIL`.

In all other columns, arbitrary text can be searched, where case is not significant. The current search pattern is displayed in the title of the table component. The **Backspace** key will remove the last character from the pattern, **Escape** clears it.

The following keys are working in all columns, including the *Level* column: **Ctrl-R** reverses the direction of the search, **F2** copies the selected message's value for the current column into the search pattern and **F3** repeats the search.

Searching always moves you to the next message, depending on search direction, that contains the search pattern anywhere in the current column, not necessarily at the beginning. When the end of the table is reached, the search is continued from the other end. This is signaled through the string *wrapped* displayed next to the search pattern, which is also where a failed search is reported.

5.2.6 Options

There are only two options that can be set for the table: whether to display lines or not and how many messages it can hold. When the limit is exceeded, the oldest messages are dropped. A value of 0 means no limit.

The **Options...** menu item will open a dialog in which these options can be edited.

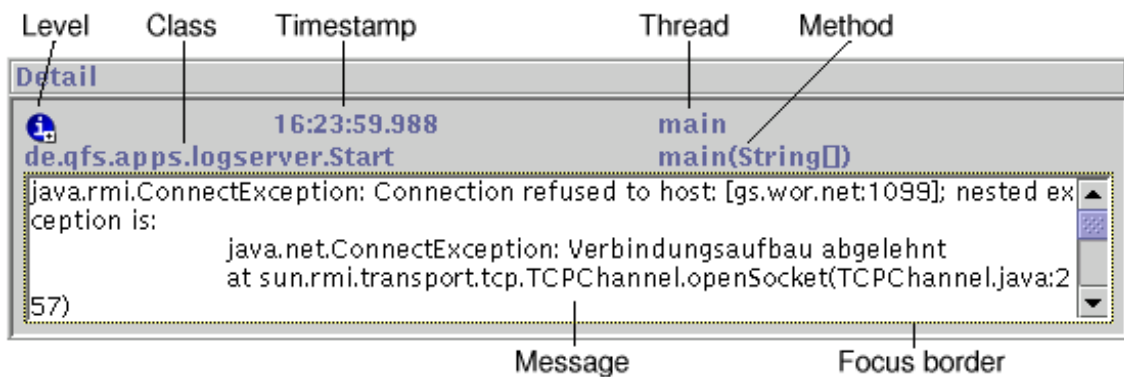


Figure 5.3: The detail view

5.3 The detail view

The most recently selected message is displayed in the detail view (figure 5.3), where the level is represented by its icon and thread, time, class and method are shown as labels. The message itself, which can be of arbitrary length, is shown in a multi line text area to which you can navigate with the **Tab** key or the mouse in order to scroll it by means of the arrow keys.

The detail view can be removed or displayed at will with the help of the **View→Show detail view** menu.

5.4 The filter tree

The filter tree (figure 5.4) is a vital aid in controlling a flood of log messages. With it you can define for each package, class or method up to which level messages originating from it will be displayed.

Multi selection is enabled for the filter tree. All actions operate on all selected nodes, no matter whether they are initiated through the **Filter** menu, the context menu or the keyboard.

Every change of the filter settings causes a redisplay of the messages in the table. Every effort is made to keep the row selection in the table intact at least for those messages that are still visible under the new setting. Also an implicit mark is set on the most recently selected message, which the display jumps to after the change.

5.4.1 The structure of the tree

The classes and methods of all messages in the log table are arranged into a tree structure, where the methods are nested inside their classes, while classes are

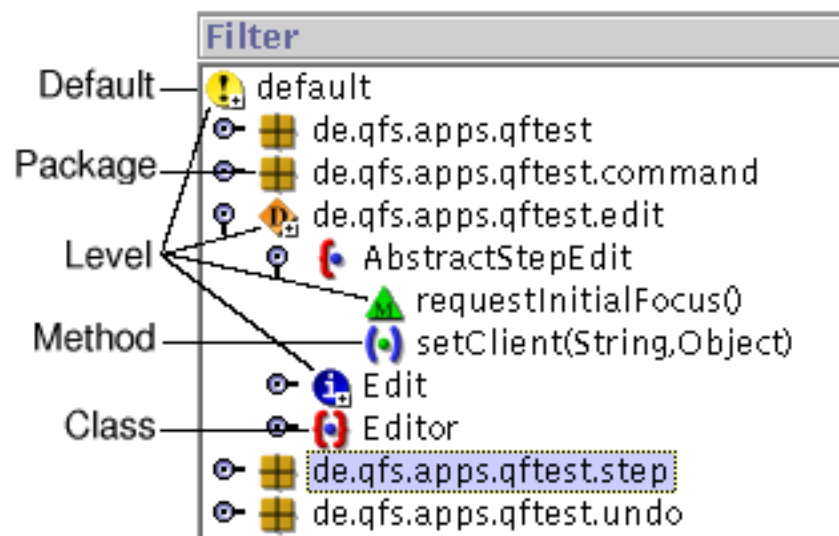


Figure 5.4: The filter tree

collected inside their package. All packages are placed directly below the root node and not according to the directory structure they represent, i.e. the package nodes `de.qfs.lib.config` and `de.qfs.lib.gui` are not child nodes of the node for the package `de.qfs.lib` but of the root node.

A level can be assigned to every node in the tree. Messages belonging to that node will be displayed in the table only if their level is less than or equal to the one set on the node. "Belonging to a node" means, that the message originated from a method that is represented by the node, or whose node is a descendant of the node. When deciding whether a message is displayed, the method node is checked first, then the class node, if no level is defined on the method node, then the package node and finally the default level of the root node.

As an example the settings shown in figure 5.4 would have the following effect:

- The default value is `WRNDETAIL`. All messages for which nothing else has been defined will be displayed up to that level.
- Messages originating from classes belonging to the `de.qfs.apps.qftest.edit` package are displayed up to the level `DBGDETAIL`, the highest level available.
- This does not hold for messages coming from the class `de.qfs.apps.qftest.edit.Edit`, for which the level `MSGDETAIL` has been set.
- Also excluded are messages from the method `requestInitialFocus()` in the class `de.qfs.apps.qftest.edit.AbstractStepEdit`, since the level `MTD` is defined for these.

This level hierarchy makes it easy to narrow the table view down to the problem area. If, on closer inspection, you find that you need more information, simply ease the filter setting a little. You have to keep in mind though, that only messages that have been generated in the first place can actually be displayed. Controlling the generation of the messages is very similar to using the filter tree and is described in section 5.4.4.

From experience there are always some trouble spots in a program that are especially delicate and cause recurring problems. By saving the configuration of the log window (see section 5.1.1) you will also save the filter settings, enabling you to work on the same problem area again should the need arise.

5.4.2 Setting the filter levels

The easiest way to change the filter settings is through the keys [0] through [9]. Again the key [1] stands for the level ERR, [9] for DBG and [0] for DBGDETAIL.

Additionally there's a menu item for every level in the **Filter** menu and the context menu of the tree.

There are two variants for clearing filter levels. The simple variant only clears the levels of the selected nodes and is available via **Remove level** or the **Delete** key. The recursive version, accessible via **Remove recursive** or **Ctrl-Delete**, additionally clears the levels on all direct or indirect descendants of the selected nodes.

5.4.3 Displaying defined levels

To find out which levels are set in the filter tree, use the **Display set levels** menu item. It causes the nodes of the tree to collapse or expand as necessary, so that all explicitly set levels are visible. This is also the initial display when the log window is opened or the configuration is restored.

5.4.4 Additional filter mechanism

There is an additional filter mechanism available for the filter tree that is similar in a way to the extra filter in the table (see section 5.2.2), but with a different character.

The whole mechanism can be toggled on or off via the **Ctrl-F** key or the **Toggle extra filters** menu item.

Each node in the tree can be declared an extra filter with the **Ctrl-A** key or the **Add extra filter** menu item. An extra filter node is displayed in a red font, whenever the extra filter mechanism is turned on.

This designation can be removed for the selected nodes with **Ctrl-R** and **Remove extra filter**, or for all nodes with **Ctrl-K** and **Clear extra filters**.

Activating the extra filters causes a further reduction of the messages in the table. In addition to the normal filter function of the tree, only those messages are displayed that belong to a node marked as an extra filter. Again, "belong" is interpreted recursively, i.e. an extra filter for a class will enable the messages for all methods of that class, assuming that they are not suppressed by the filter level settings.

This extra filter mechanism is intended as a simple means to achieve a short term increase of the filter effect, without having to modify the levels. For example, by activating two method nodes as extra filters you can study the interaction just between those methods, putting it back into context after deactivating the extras.

Due to the short term character of the extra filters they are not saved in the configuration like the filter levels.

5.5 The client's log levels

This component is available only when the log window is either embedded directly in a program or belongs to an RMI client. It gives access to all the settings of the client that determine which log messages are generated. Its operation is similar in many respects to that of the filter tree (see section 5.3).

5.5.1 The structure of the tree

As figure 5.5 shows, the structure of the level tree is similar to that of the filter tree (see section 5.4), except for two differences: the hierarchy ends at the class level, i.e. there are no method nodes, and packages do nest according to the directory structure they represent. Thus the package `de.qfs.apps.qftest` is nested four levels deep under the root node and the nodes `de`, `de.qfs` and `de.qfs.apps`.

Both differences arise from the need for the level tree to represent the structure of the levels of the `Logger` objects in the client.

5.5.2 Setting the levels

The levels of the classes and packages of the level tree are set and cleared similarly to the levels in the filter tree (see section 5.4.1), i.e. through the keys **O** through **9**, **Delete** and **Ctrl-Delete**, the **Level** menu or the context menu.

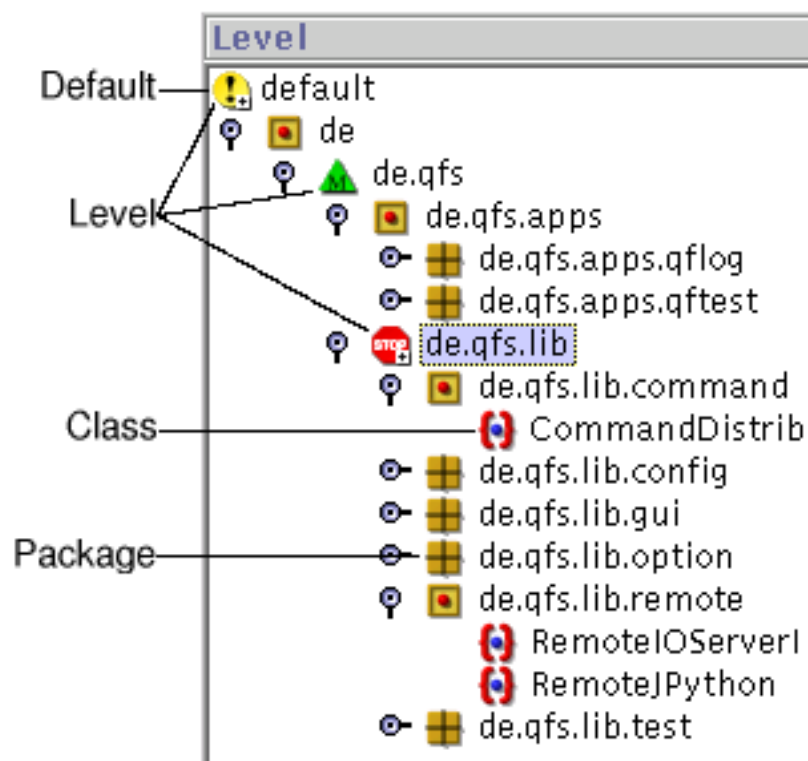


Figure 5.5: The level tree

5.5.3 Displaying defined levels

Also similar is the method to get an overview over which levels are set. Use the **Level→Show set levels** menu like in section 5.4.2.

5.5.4 Options

There are quite a few Options that can be set in the dialog accessible through the **Options...** menu item. Except for the last, these values correspond to the parameters used by the logging system in the client. Please read the documentation of the `de.qfs.lib.log.Log` class to learn what their use is.

The check button named "Override client's configuration on connect" is different. It determines the behaviour of *qflog* the next time a client with the same name connects. If the option is not checked, the client's settings will be read, otherwise the parameters for the logging system of the client and the levels from the level tree override the settings inside the client. This option will be more useful, once an option to reset the client to its original values on disconnect is also available.

Chapter 6

A sample application

qflog comes with a test client that can be used to test and demonstrate the various ways of interaction between a client and *qflog*. It generates random messages from a number threads and can redirect those to a log file, bring up an internal log window or communicate with *qflog* via RMI

6.1 Invocation

The test client can be invoked either directly with

```
java [java-options...] de.qfs.apps.qflog.TestClient  
[options...]
```

or with the startscripts provided in the `bin` directory of the *qflog* distribution:

```
testclient [options...]
```

After the launch a trivial window with a button that exits the client appears and logging starts depending on the arguments given.

6.2 Options

The following options are defined for the test client:

-allownonlocal

Allows a *qflog* server from a host other than localhost to access the client.

-clientname <name>

Sets the name under which the client registers itself with *qflog*. The default is "testclient".

-configfile <file>

Determines the file used for saving and loading the configuration.

-createregistry

Enables creation of an RMI registry, if none is available during the start of the client.

-internal

Creates an internal log window that is opened when the client starts.

-logfile <file>

Causes log messages to be written to a file.

-logserver <servername>

Tells the test client to connect to a log server. The name to use is "qflog", unless *qflog* has been started with a different `-servername` option.

-numloggers <number>

Determines the number of threads that create log messages. Default is 3. The first thread will create one message per second, the second thread one every other second, the third every 3 seconds and so on.

-outputlevel <level>

Sets the level up to which log messages will be printed on `System.err`. The value 0 suppresses all messages, 10 lets all messages pass. Default is 2.

-port <portnumber>

Sets the port number for the registry with which the client should register to wait for a log server.

-waitforserver

Tells the client to register with the RMI registry, so a log server is able to connect to it at a later time.

Additionally options of the form `-log-<name> <level>` may be used to set the levels of the Loggers. Please see the example section in the *qflog* documentation for details.

6.3 Example uses

Simply try out a few examples to get a feel for how *qflog* works:

- `testclient -internal` shows what the log window for the test client looks like. Play with the settings in the filter and level trees and watch the result.
- First bring up *qflog* then launch `testclient -logserver qflog`. You should see an entry named "testclient" in the *qflog* main window. Bring up the log window for the client, it is similar to the internal one.

- You can combine these examples with `testclient -internal -logserver qflog`, bringing up the internal log view as well as connecting to the log server. Notice how changes to the level tree in one window are immediately reflected in the level tree in the other window.
- Start the test client first with `testclient -waitforserver -createregistry` and then run *qflog*. Again you should find a "testclient" entry in its main window.

Appendix A

qflog and applets

The communication between an applet and *qflog* is not without problems. One reason is the missing RMI support in most versions of Microsoft's Internet Explorer. Another is the sandbox which restricts an applet's capabilities for security reasons, causing difficulties with RMI callbacks.

In any case, unless an applet is signed, the sandbox will prevent it from connecting to *qflog* on any host other than the one from which the applet was loaded.

Nevertheless, logging through *qflog* can already help a great deal when developing applets and it may get a little easier in the future, if other communication protocols than RMI are implemented.

A.1 Internet Explorer

The main problem when logging from applets running in Internet Explorer is the missing RMI support in older versions (probably before 5.0). At least for development this is not a big deal, since Microsoft is providing a `jar` archive under <http://www.microsoft.com/Java/resource/misc.htm>, which can be installed to get the necessary classes. With these in place, even RMI callbacks work, so the applet's log levels can be controlled at runtime.

A.2 Netscape

All Netscape versions starting around 4.02 support the full *JDK 1.1*, though only at version 1.1.4. Applets can connect to *qflog* without problems, but callbacks don't work, making runtime control over the log levels impossible for now. Signing an applet should help, which we couldn't test yet.

A.3 Plugin

We didn't have time yet to test applet logging with the *Java* plugin from *SUN*. We'd appreciate any feedback on this, be it for plugin version 1.1 or 1.2.

A.4 Example applet

An applet can be written in such a way that it can be deployed with only the `de.qfs.lib.log` package of *qflib*, while still using the full logging capabilities during development. The *qflib* examples¹ page has some example code for that.

¹<http://www.qfs.de/en/projects/qflib/examples.html>