The KDebugSettings Handbook

Laurent Montel

Contents

Abstract

KDebugSettings configures which debug messages Qt™ and KDE applications print, without recompiling them.

Chapter 1

Introduction

Qt™ applications group their diagnostic output into *logging categories*: named channels such as `org.kde.pim.kmail` or `kf.kio.core`, each of which can be turned on or off independently and for each severity (debug, info, warning, critical). By default most categories are quiet, so that applications do not flood the journal in normal use.

KDebugSettings is a graphical front end for those rules. It lists all the categories installed on the system, lets you choose how verbose each one should be, and writes the result to the configuration file Qt™ reads at application startup:

`$XDG_CONFIG_HOME /QtProject/qtlogging.ini` (usually `~/.config/QtProject/qtlogging.ini`)

Rules take effect the next time an application starts; already running applications are not affected.

> **NOTE**
> KDebugSettings does not decide *where* the messages go, only which ones are produced. The output still ends up wherever the platform sends it, typically the systemd journal or the terminal the application was started from.

Chapter 2

Using KDebugSettings

The main window is a dialog with three tabs and a row of buttons shared by all of them.

2.1 The KDE Application Tab

This tab lists every logging category declared by the applications and libraries installed on your system. Each row shows the description of the category and a drop down box with its current severity level. Hovering over a row displays the category name, its identifier and its default severity.

Use the **Search...** field at the top to narrow the list; the filter matches the description, the category name and the identifier, so typing **`kmail`**, **`org.kde.pim`** or **`KMAIL_LOG`** all work.

To change a single category, pick a level in its drop down box. To change several at once, select the rows first — the change is then applied to the whole selection. The three buttons at the bottom act on the selected rows, or on every row currently shown by the filter when nothing is selected:

Enable All Debug

Sets the categories to **Full Debug**, i.e. every message of every severity is printed.

Turn Off Debug

Sets the categories back to **Info**: informational, warning and critical messages are kept, debug messages are dropped.

Turn Off All Messages

Sets the categories to **Off**, silencing them completely.

The **Restore Defaults** button of the dialog resets every category of this tab to the default severity declared by the application that owns it (see chapter 3).

2.1.1 Severity Levels

A category is not a simple on/off switch: choosing a level enables that severity and everything more serious than it. The levels offered, from the most to the least verbose, are:

Level	Messages that are printed
Full Debug	debug, info, warning and critical
Info	info, warning and critical
Warning	warning and critical
Critical	critical only
Off	nothing

Each level is written out as an explicit rule for all four severities, so that a category always has a well defined state regardless of what the system wide defaults are. Selecting **Warning** for `org.kde.example`, for instance, produces:

```
org.kde.example.info=false
org.kde.example.debug=false
org.kde.example.warning=true
org.kde.example.critical=true
```

2.2 The Custom Rules Tab

Not every category is declared in an installed `.categories` file: third party applications, or code you are currently working on, may use categories KDebugSettings knows nothing about. This tab lets you write rules for them by hand.

Use **Add...** to create a rule, **Edit...** to change the selected one and **Remove...** to delete the selected ones. A rule consists of a category name, a type and an **Enable** check box, and is displayed in the Qt™ syntax, e.g. `org.kde.example.debug=true`.

The category name may contain the wildcard `*`, at the beginning of the name, at the end, or both. `org.kde.pim.*` therefore matches every PIM category at once.

> WARNING
> A rule whose category is just `*` with no type, i.e. `*=true` or `*=false`, matches everything and overrides all the other rules. KDebugSettings warns you at the top of the dialog when such a rule is found and it is best to remove it. If you really want a catch-all rule, write it as `*.*=true` or `*.*=false` instead, which is ordered so that the more specific rules still win.

2.3 The Rules Settings With Environment Variable Tab

Qt™ also reads logging rules from the `QT_LOGGING_RULES` environment variable, and those rules take precedence over the ones stored in `qtlogging.ini`. If the variable is set in the environment KDebugSettings was started from, this tab shows its content, so that you can tell why a category behaves differently from what the first two tabs suggest.

The rules are read-only here: to change them, change the environment variable itself.

2.4 Saving, Loading and Groups

OK and **Apply** write the rules of the first two tabs to `~/.config/QtProject/qtlogging.ini`. Only the categories that differ from their default severity are written, which keeps the file short and lets applications change their own defaults later.

Preview… opens a read only window showing the `qtlogging.ini` that **OK** and **Apply** would write, without writing anything. Since it applies the same rule as they do, it is also the quickest way to see which categories were left out because they still sit at their default severity.

The remaining buttons manage rule sets you want to keep side by side, for instance one per bug you are chasing:

Save As…

Save As File exports the current rules to a `.kdebugsettingsrules` file anywhere on disk. **Save As Group** asks for a name and stores the rules as a group in `~/.local/share/kdebugsettings/groups/` .

Unlike **Apply**, both write out *every* category, not only the modified ones, so that reloading the file restores exactly the state you saved.

Load

Load From File reads back a `.kdebugsettingsrules` file, **Load Group** offers the saved groups in a submenu, and **Manage Group** opens a dialog where groups can be renamed, removed and exported. Loading only fills the dialog; use **OK** or **Apply** to actually activate the rules.

Insert…

Reads an additional `.categories` file and adds the categories it declares to the **KDE Application** tab. This is useful for an application that is not installed system wide, typically one you are building yourself.

Chapter 3

Declaring Categories: the `.categories` File

The list shown in the **KDE Application** tab is not hardcoded. It is read from the `.categories` files installed by applications, in:

- `${KDE_INSTALL_LOGGINGCATEGORIESDIR}`, i.e. `/usr/share/qlogging-categories6/` , where `kde.categories` is the shared file listing the categories of the KDE applications;
- the XDG configuration directories, for example `~/.config/` ;
- `$XDG_CONFIG_HOME /qdebug.categories/` , for applications outside KDE.

The format is one category per line. Lines starting with `#` are comments:

```
<category name> <description> [DEFAULT_SEVERITY [<severity>]] [IDENTIFIER ↩
    [<identifier>]]
```

The name and the description are mandatory, the rest is optional. `<severity>` is one of `DEBUG`, `INFO`, `WARNING` or `CRITICAL` and gives the level the category falls back to, the one **Restore Defaults** returns to; when it is absent, `INFO` is assumed. `<identifier>` is the name of the `QLoggingCategory` object in the source code, which makes the category searchable by the name developers actually type. For example:

```
org.kde.example.core Example core library DEFAULT_SEVERITY [WARNING] ↩
    IDENTIFIER [EXAMPLE_CORE_LOG]
```

If you build your application with the extra-cmake-modules, you do not need to write this file by hand: declare the category with `ecm_qt_declare_logging_category()` and install it with `ecm_qt_install_logging_categories()`, and the file is generated for you.

3.1 Renamed Categories

When an application renames one of its categories, the rules users already have in their `qtlogging.ini` still refer to the old name. A `.renamecategories` file, installed next to the `.categories` file, tells KDebugSettings how to migrate them. Each line maps an old name to a new one:

```
# old module name<space>new module name
org.kde.old.name org.kde.new.name
```

Chapter 4

Command Line Options

Besides the usual KDE options, KDebugSettings accepts the following, which list or change the rules without opening the dialog:

`--list`
: Prints the name of every known logging category on the standard output and exits, without changing anything. The categories are the ones read from the `.categories` files (see chapter 3), i.e. the same ones listed in the **KDE Application** tab:

```
% kdebugsettings --list | grep kmail
```

`--enable-full-debug`
: Sets every known category to **Full Debug** and exits.

`--disable-full-debug`
: Sets every known category to **Off** and exits.

`--debug-mode Full|Info|Warning|Critical|Off category...`
: Sets the given level on the categories named as arguments and exits. At least one category name is required:

```
% kdebugsettings --debug-mode Full org.kde.pim.kmail org.kde.pim. ↩
    messagelist
```

`--test-mode`
: Enables the `QStandardPaths` test mode, so that the settings of the unit tests are used instead of yours. Intended for testing KDebugSettings itself.

Chapter 5

The System Settings Module

KDebugSettings also installs a System Settings module, found under **System Administration** → **Debug Settings**. It offers the same three tabs as the standalone application and writes the same `qtlogging.ini` file.

Chapter 6

Credits and License

KDebugSettings