KDE

Published : 2011-10-21

KDE FROM A DEVELOPER'S VIEWPOINT

1. DO YOU NEED THIS BOOK?
2. THE KDE PHILOSOPHY
3. HOW TO GET HELP

1. DO YOU NEED THIS BOOK?

You should read this book if you want to do any development for KDE. We're using the term *development* very broadly to cover anything that can lead to a change in source code. This includes:

- Submitting a bug fix
- Writing a new application powered by KDE technology
- Contributing to an existing project
- Adding functionality to KDE development libraries

In this book we'll give you the basics you need to be a productive developer. We'll explain the tools you should install, show you how to read the documentation (and write your own, once you've created new functionality!) and how to get help in other ways. We'll introduce you to the KDE community, which is key to understanding KDE because we are a free, open source project.

Regular users of the software do **NOT** need this book! However, they might find it interesting to help understand how the complex and richly featured software they use has come into being.

2. THE KDE PHILOSOPHY

KDE's success is based on a world-view that we've found to be both practical and motivating. Elements of this development philosophy include-

Using available tools rather than re-inventing existing ones: Many of the basics you need to do your work are already part of KDE, such as the core libraries or Kparts, and are quite mature. So check out all available resources before trying to solve your own problem.

When making a suggestion, change *we should..* to *I will..*: Grandiose plans are useless unless you are willing to put in the work to make them happen. You will find help once you start!

Improve code iteratively: Don't let the perfect be the enemy of the good. Try a small solution, get it to work, and improve it through testing and refactoring to produce an excellent patch.

THE KDE COMMUNITY

The KDE development platform is created and maintained by an international team that cooperates on the development and distribution of free, open source software for desktop and portable computing. Our community has developed a wide variety of applications for communication, work, education, and entertainment. We have a strong focus on finding solutions to old and new problems, creating an open atmosphere for experimentation.

As a community, we value all contributions, and can use all member talents, including artwork, graphics, design, communication, translations, documentation, testing, bug-reporting and bug-hunting, system administration, and coding.

WHAT MAKES KDE SO EXCITING?

The best thing about KDE is our amazing community! We welcome new members, offering help and allowing people to experiment, learn and grow. This book is a part of that mission.

Our products are used by millions of home and office workers, and are being deployed in schools around the world. Brazil alone has over 50 million school children using KDE-based software to browse, learn and communicate! As a complement to Google Summer of Code, we run our own Season of KDE, where people take on the responsibility of working on a project intensively for the summer, and get a cool T-shirt at the end, in thanks. In 2011, we had 100 proposals! What an amazing community.

IS IT DIFFICULT TO GET INVOLVED?

Not at all! Every day, more and more people join our ever-growing family of contributors. KDE has a strong infrastructure of web resources, forums, mailing-lists, IRC, and other communication services. We can provide feedback on your code and other contributions with the goal of helping you learn. Of course, a proactive attitude and a friendly personality helps.

THE KDE CODE OF CONDUCT

When communicating in official KDE channels please observe the KDE Code of Conduct.

Our Code of Conduct presents a summary of the shared values and *common sense* thinking in our community. The basic social ingredients that hold our project together include:

- Be considerate
- Be respectful
- Be collaborative
- Be pragmatic
- Support others in the community
- Get support from others in the community

The Code of Conduct can be found here : *http://www.kde.org/code-of-conduct/*

THE VALUE OF BEING FREE AND OPEN SOURCE

Our code can be copied, changed, and redistributed by anybody. This means you can be confident that the software will continue to be available even if the original developers move on. It also means that development is out in the open, with comments from anyone in the world who is interested. Finally, open source code makes it easier for a developer to find and fix problems. One of the practical effects of free software is that wonderful communities tend to develop around code. We have created all this within KDE.

The KDE community is involved with the larger free and open source movement. We cooperate with the GNOME community in FreeDesktop.org and the bi-annual Desktop Summit. We've been a leading participant in Google Summer of Code and Google Code-In since they began, and we regularly sprint with members of other FOSS groups, and participate in open standard efforts. We are supported in these efforts by the KDE e.V., who issue quarterly reports of their activity at *http://ev.kde.org/reports/*.


debian

3. HOW TO GET HELP

Of course, we hope our book is helpful to you! But inevitably, you will come up with questions or need help with a problem. The KDE community is helpful and friendly, so pick the most appropriate method in this chapter and ask away. Unless you have a specific need to talk to one particular person, please address the whole list or channel.

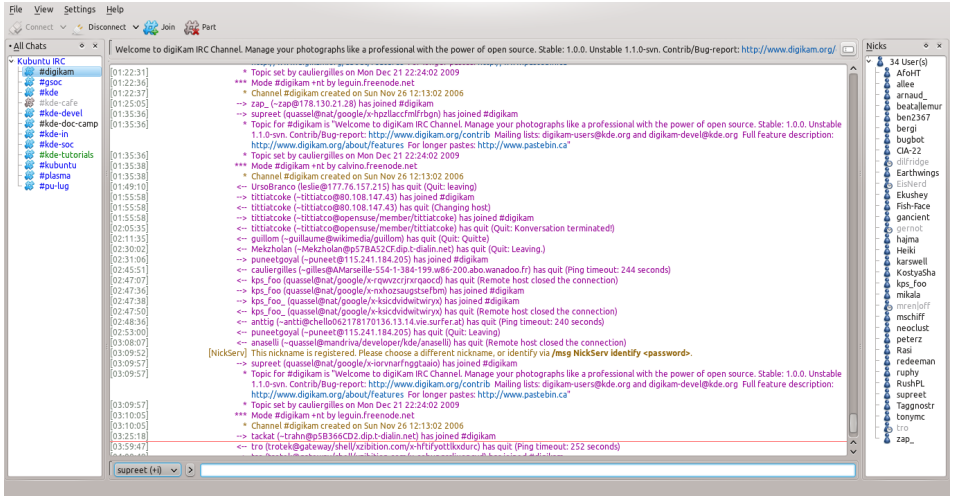
KDE MAILING LISTS

The KDE mailing lists are one of the main communication channels in the KDE Community. All developers will find the KDE-devel list useful. In addition, those working on the core of the KDE Software Compilation (SC) will want to subscribe to KDE-Core-devel. Those working on applications or KDE projects should subscribe to the project mailing list. The full spectrum of KDE lists can be found at https://mail.kde.org/mailman/listinfo.

Both help and information are available on the lists. However, if you need help quickly, IRC may be more useful.

KDE ON IRC

IRC is Internet Relay Chat, a text-only, real-time communication tool. There are a variety of IRC clients available from KDE, such as Konversation, Quassel, and Kvirc.



Almost all KDE developers show up or idle on KDE IRC channels on Freenode (irc.freenode.net). IRC is the best way to get quick help from the KDE developers. As a developer, you will want to be in #kde, #kde-devel and your project channel or channels. You'll find that the more time you spend in IRC, the more you will get to know your fellow developers and our KDE users. Life-long friendships have started in KDE channels. Help is available about the services such as chanserv and nickserv by using the commands */msg chanserv help* and */msg nickserv help*. More about using Freenode is available here: http://freenode.net/using_the_network.shtml.

Userbase keeps a list of channels current at http://userbase.kde.org/IRC_Channels, and you can also use the IRC command */msg alis list $searchterm*, where *$searchterm* is the subject in which you are interested.

When asking questions, please bear these tips in mind:

- If you have a question, just ask it. There's no need to ask first whether you can ask a question.
- Be prepared to wait for an answer. Even though IRC is a more real-time mode of communication than mailing lists, there may not be anyone available to answer your question immediately. In general, if you don't receive a response on IRC in about an hour, it's best to send an email.
- Don't ask your question more than once. Even though the channel is active, the right person may not be available to provide an answer. The one exception to this rule is that, if you are told to wait for a certain person to come online, ask again when you see him or her come online. Again, if you don't receive a response to your question in about an hour, it's probably best to send an email.
- Pasting large amounts of text is considered bad etiquette, so use a pastebin. KDE's pastebin is at http://paste.kde.org

KDE COMMUNITY PROBLEMS

If you encounter bad behavior on a list or in a channel, please contact the list owners or one of the ops. The list owner address is *$listname*-owner@kde.org, where *$listname* is the name of the list. Ops in a channel can be identified by issuing this command: */msg chanserv access #channelname list*. Ops will have a "+" next to their nicks. For general KDE community help, please write to the Community Working Group at *community-wg@kde.org*, or stop by our IRC channel at #kde-cwg.

FIRST STEPS

4. CHOOSING A PROJECT

When you come to KDE as a developer, you may already have a favorite project and know how you want to contribute. But it's worth looking over the various projects listed in this chapter, to find out all the ways you may be able to help. And even if you're really only interested in one project, it's useful to know what others are active because your work may interact with them.

FRAMEWORKS

These are general components underlying the applications and other visible parts of KDE. The team is working hard to make the libraries modular, clarify the dependencies, simplify, and increase the quality and stability.

KDE Core Libraries (kdelibs)

Critical functions needed across the KDE platform

Widgets & Classes

Widgets and classes that are not in kdelibs but that are widely useful

D-Bus Web Service Proxy

A project to connect Web Services to the D-Bus notification framework on Linux

kdesu

Tools for gaining superuser privileges on different backends

KDE WebKit

A project integrating the QtWebKit browser engine into the KDE Software Compilation

KioFuse

This inserts KIO(K Input Output) resources (remote, archived, or compressed files) into the root filesystem hierarchy

Nepomuk

Basic desktop operations for annotation, indexing, search, and linking

Network Management

An applet and configuration tool for Solid Networking and KNetworkManager

Oxygen

Artwork for the KDE SC

Solid

The KDE SC hardware library

Telepathy

A communications framework for Instant Messaging, VoIP, and Collaboration

Akonadi

An extensible cross-desktop storage service for PIM data and meta data

Related projects

Non-central projects that are related to KDE technology in various ways, such as dependencies or build tools

PROGRAMS

This is a small cutout of the applications created and maintained by KDE developers.

Amarok

Amarok's tagline is *Rediscover Your Music*, and its development is based around this ideology. Amarok's core features such as the unique *context browser*, integrated Wikipedia lookup and lyrics download help users to find new music, and to learn more about the music they have.

Digikam

Photo management software

Gwenview

KDE image viewer

K3b

An optical disc writer

KDevelop4

Integrated Development Environment (IDE) for KDE SC

KWin

The KDE Window Manager

Marble

A visually appealing globe program

Okular

A unified document viewer

Rekonq

A lightweight web browser powered by WebKit and KDE SC

System Settings

The System Settings configuration tool.

SuperKaramba

A tool for creating attractive widgets and other interactive elements

SUITES

These projects group many related applications. The key concept is that data created in one part of the suite can be easily used by another application in the suite.

KDE Education Project

Educational software for KDE

KDE Finance

Financial applications

KDE Games

Desktop games for KDE

Kdetoys

A set of amusing diversions

KDE Utilities

A variety of tools to run on the desktop

KOffice and Calligra

Office suites based on KDE libraries

PIM

Personal information management tools

Plasma

Programs for the quick and easy creation of widgets, including interactive application launchers, and window and task managers

PLATFORMS

These projects ensure that KDE works on various operating systems.

KDE on Mac OS X

KDE libraries and applications for Mac OS X

KDE on Windows

KDE libraries and applications for Microsoft Windows

KDE on FreeBSD

KDE libraries and applications on FreeBSD and other BSD versions

Plasma Active

A project for porting KDE technology to mobile devices

WORKING WITH THE ORGANISATION

These projects deal with the people and processes that make KDE possible.

KDE Release Team

Schedules and coordinates releases

KDE Documentation Project

Creates and maintains KDE documentation

kde.org

Provides information around the *.kde.org websites

KDE Promotion

Promotes KDE and organizes conferences

Partner Program

Supports KDE partner Independent Software Vendors (ISVs)

KDE Usability project

Applies usability principles and practices to the K Desktop Environment

KDE Accessibility project

Builds on Qt features for making interactive environments more accessible to the disabled or others with special needs

KDE BugSquad

Keeps track of incoming bugs in KDE software, and goes through old bugs.

Summer of Code Projects

Google Summer of Code projects related to KDE

English Breakfast Network (EBN)

Provides tools dedicated to KDE Code Quality, including KDE API Documentation Validation, User Documentation Validation, Source Code Checking, etc.

KDE Research

Supports everyone who is interested in contributing to (funded) research projects with(in) the KDE community.

If you are still confused which project you want to work with then try hanging out with KDE SC developers on IRC to become familiar with the project.

5. THE QT FRAMEWORK

To start developing on the KDE Development Platform you will need to get familiar with the Qt framework, which is one of building blocks of KDE development.

Qt (pronounced officially as cute) is a cross-platform application framework based on C++, that is widely used for developing application software with a graphical user interface (GUI). Thus, it is largely a widget toolkit, but is also used for developing non-GUI programs such as command-line tools and consoles for servers.

Besides the KDE Development Platform, Qt is most notably used in Autodesk Maya, Adobe Photoshop Elements, OPIE, Skype, VLC media player, VirtualBox, and Mathematica, and by the European Space Agency, DreamWorks, Google, HP, Lucasfilm, Panasonic, Philips, Samsung, Siemens, Volvo, and Walt Disney Animation Studios.

ADVANTAGES OF QT

Writing code once to target multiple platforms

Qt allows you to write advanced applications and UIs that you can deploy across different desktops and embedded operating systems without rewriting the source code, saving time and development cost.

Creating amazing user experiences

Whether you prefer C++ or JavaScript, Qt provides the building blocks for modern, interactive interfaces: a broad set of customizable widgets, graphics canvas, style engines, and more. You can incorporate 3D graphics, multimedia audio or video, visual effects, and animations to set your application apart from the competition.

Doing more (and faster!) with less

Qt is fast to learn and to use, particularly when used together with the new Qt Creator cross-platform IDE. And Qt's modular class library provides much of the necessary infrastructure for interactive applications.

Blending web and native code in a single application

Qt's integration with the WebKit web rendering engine means that you can quickly incorporate content and services from the Web into your native application, and can use the web environment to deliver your services and functionality.

To learn how to use Qt, we recommend the tutorials here:

http://doc.qt.nokia.com/

6. USING GIT FOR KDE DEVELOPMENT

Git is a free and open source version control system designed to handle everything from small to very large projects with speed and efficiency. It provides lots of tools for figuring out where you have gone as you edit files, as well as merging your changes with those made by other developers.

You can find more about git (and download it if necessary) at http://git-scm.com.

GIT BASICS

There are several levels at which your changes can reside in git. You need to go through all the steps carefully in order to save changes correctly. A typical sequence of git activities starts with cloning a remote repository. Now you have a complete copy of what the original developer has. Then you should do the following:

1. Create and edit files.
2. Run git add to tell git about any new files you have created or files that you have just edited. Files with changes are in an intermediate state called a staging area, but not in any repository yet.
3. Run git commit to save your changes in your own local repository.

We'll show examples of all this later in the chapter.

Instead of having to remember and type in the different full git addresses for pulling and pushing, we recommend you manually add the following to your Git User Configuration (*~/.gitconfig*):

```
[url "git://anongit.kde.org/"]
   insteadOf = kde:
[url "git@git.kde.org:"]
   pushInsteadOf = kde:
```

CLONING REPOSITORIES

After setting up your *~/.gitconfig* as shown in the previous section, you need to clone the source repositories using the following command:

```
git clone kde:project_name
```

where *project_name* is the name of the project that you want to contribute to. For instance, to submit a patch to kdelibs, start with the command:

```
git clone kde:kdelibs
```

PULLING CHANGES

If you already have a git repository that you need to update with new changes from the original source, run the following:

```
git pull
```

WORKING WITH BRANCHES

Git is a very powerful revision control system that supports the concept of branches. In order to develop a new feature for a KDE Development Platform Project, it's best to use a separate branch for feature development. You can check out a new branch using the following command:

```
git checkout -b feature_name
```

for example:

```
git checkout -b myFeature
```

Later on you can switch between multiple branches using the `git checkout` command:

```
git checkout myFeature
```

Every git project has a branch called *master* that the owners of the repository consider the main branch. Usually, nothing gets added to this branch until it is tested in a variety of environments and the project leaders are sure it's both robust and useful.

Tracking Remote Branches

There are two types of branches in git, *local* and *remote*. Remote branches are branches that exist in the remote repository. You can also track remote branches using the following command:

```
git checkout --track remote_repository:remote_branch
```

For instance, to checkout the KDE 4.7 branch, use:

```
git checkout --track origin:KDE/4.7
```

COMMITTING YOUR WORK

Before pushing anything to the KDE codebase, you need a KDE identity and a developer account. Please visit http://identity.kde.org/ to register your account. Your account name will almost always be your surname; do not attempt to get around this rule. Developers with a lot of experience can get accounts with commit rights, but that is outside the scope of this book.

In order to generate a diff of your changes and review them, run the following:

```
git diff
```

You can pipe this output into a patch using the following:

```
git diff > my_patch.patch
```

In order to commit your work, you first need to add the files you changed to the staging area using the following command:

```
git add fileName1 fileName2
```

In order to commit your changes to your local repository, use the following command:

```
git commit
```

This opens up a text editor where you can type a commit message that describes your changes.

SUBMITTING YOUR CHANGES INTO THE MAIN TREE

You can submit your patches to the KDE Review Board at https://git.reviewboard.kde.org/. Log in using your KDE Identity account and submit a patch to a project there.

You can also directly send patches to review board using post review. The procedure is outlined at:

> http://community.kde.org/Sysadmin/GitKdeOrgManual#Using_Review_Board_and_post-review

TROUBLESHOOTING

For any issues related to git and the KDE Development Platform, you can seek help in one of the following channels on irc.freenode.net.

- #kde-sysadmin
- #kde-git

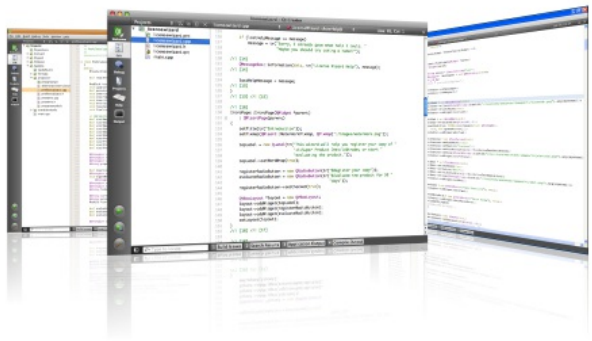
You can read more about KDE platform development using git at http://techbase.kde.org/Development/Git.

7. CHOOSING AN IDE

An integrated development environment (IDE) allows you to do project management, testing, and other activities in a convenient way alongside your coding. We recommend that you install one of the following IDEs and do your KDE development work within it.

We recommend QtCreator for its ease of use and features, especially its built-in text editor. But it's nice to know, if you're familiar with Eclipse already, that you can use that for KDE development too.

QTCREATOR

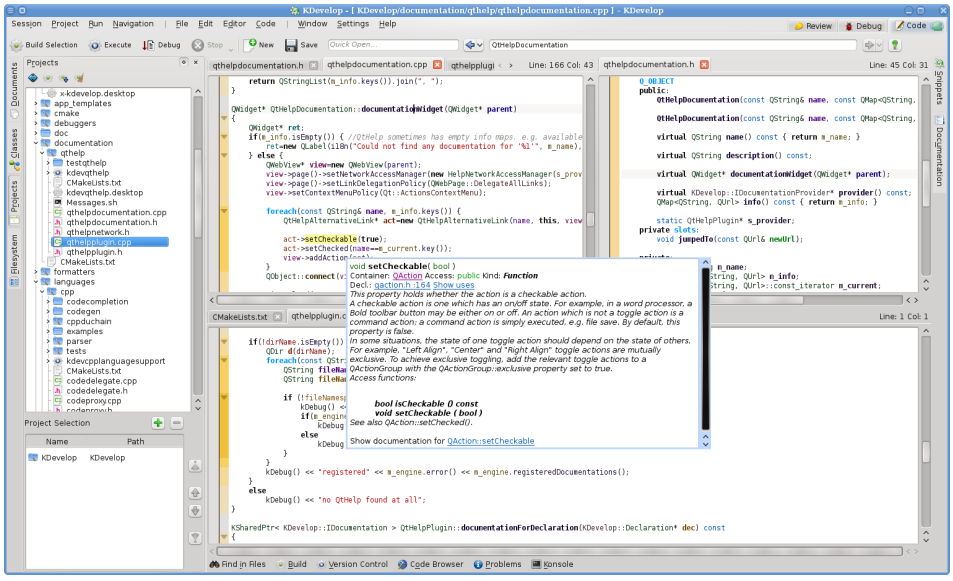


QtCreator is an integrated, cross-platform IDE for C++ and JavaScript that is part of the Qt SDK. It includes a visual debugger and a designer tool for GUI layout and forms. The editor's features include syntax highlighting and auto-completion. QtCreator uses the GNU C++ compiler and related tools. On Windows it can use MinGW or MSVC with the default install, and cdb when compiled from source.

You can find out more about using QtCreator with the KDE development platform at:

http://techbase.kde.org/Development/Tutorials/Using_Qt_Creator

KDEVELOP



KDevelop is a free, open source IDE (Integrated Development Environment) created as part of the KDE development project to support development on C/C++ and other languages on Microsoft Windows, Mac OS X, Linux, Solaris and FreeBSD. It is feature-full and can be extended through plugins. It is based on KDevPlatform and the KDE development platform and Qt libraries and has been under development since 1998.

You can find out more about KDevelop at:

http://userbase.kde.org/KDevelop4/Manual

ECLIPSE

Eclipse is a popular open source IDE. It is used primarily for programming in Java, but supports a number of other languages, including C++, with the appropriate plug-in modules.

You can find out more about using Eclipse with the KDE development platform at:

http://techbase.kde.org/Development/Tools/Eclipse

8. KDE APIS

An Application Programming Interface (API) is a set of functions exposed by a program or library that can be invoked by other programmers. An API greatly extends a program by allowing third-party developers to add new functionality. It is an ideal way to allow new features to be added without the need to modify the existing, core code.

The documentation for an API explains how things work and what methods can be called. For some nice tutorials to get started with using the myriad KDE development platform APIs, visit:

> http://techbase.kde.org/Development/Tutorials

The KDE development platform API documentation (apidox) can be found at:

> http://api.kde.org

This documentation comes in the qch format, an extended HTML format that can be viewed either in the QtCreator Assistant or via any browser.

The apidox are processed by the Doxygen documentation tool. This tool reads source code for an application (or library) and produces nicely formatted documentation from it. There is a good reference manual available, but hopefully you won't need to consult it just for basic documentation building. When you install the KDE development platform for development purposes, you should also install Doxygen. Then you can build documentation from any project by using the following command:

```
make docs
```

More information on the KDE API docs can be found at:

> http://techbase.kde.org/Development/Tools/apidox

9. DOCUMENTATION

If you write new functions for a KDE project, we certainly hope that you write documentation (apidox) for it, and try to explain as clearly as possible what can be done with the functions. The better you document your work, the more developers are likely to use it and the less likely they are to annoy you by asking basic questions about how it works.

API documentation is sometimes called an API reference manual. It needn't be just a reference manual, though. It can contain extensive additional material, such as tutorials, examples, and historical information. This page refers to all the material that documents and explains the API of a piece of code as "apidox", the term used in the KDE development platform documentation itself.

Basic apidox writing is fun and simple: add specially formatted comments in your code explaining what things are supposed to be for. These comments are nearly indistinguishable from stuff you would be writing in the headers of your code anyway, so that's not hard to do. Here is a sample.

```
/**
 * @author praxagora
 *
 * Returns a CHI square comparing two samples.
 * The arrays must be the same size.
 *
 * @param a First sample
 * @param b Second sample
 *
 * @return @size size of each sample.
 */
double chi_square(double a[], double b[], int size) {
double total_a=0.0, total_b=0.0, total, totals[size],
square_a[size], square_b[size], square_totals[size],
    terms[size],
    terms_total=0.0, sum_total=0.0, square_sum_total=0.0;
  int i;
  /**
    * This loop comprises the whole function and computes the CHI
    * square from the two arrays.
  */
    {
  for (i = 0; i < size ; i++)
```

Note that a comment was included before the function, and before the bracket that starts the next level of nesting, a for loop. All the parameters, as well as the return value, are marked.

To write good apidox, you first need technical knowledge: you should understand the code you are documenting--or at least know what it is supposed to do and how it is meant to be used. The other part of good apidox is plain discipline: apidox are most useful when they are exhaustive.

Actually, documentation doesn't usually have to explain what the code does at each step. The code should be so clear, with a clean layout and well-chosen names for methods and variables, that it is self-documenting. Rather, documentation should explain what code is *for*, when and why it is called, the purposes and ranges of its arguments, and perhaps the algorithm used and the trade-offs made in memory and time.

Look at the Qt documentation to get a feeling for what good apidox look like. They have a consistency of style and are permeated with a concern for thoroughness. You can learn a lot about Qt just from reading the documentation. You do not necessarily need to run the tutorial programs or read the source code of the library to find out what a parameter or flag does in some method of the library. It is all spelled out for you.

KDE DEVELOPMENT BUILD ENVIRONMENT

10. PREREQUISITES

Before you actually build the KDE development environment from source, think -- do you need to do this? Yes, if you will be working on the KDE SC core. You will not need the entire environment built from source if you intend to work on one application. We recommend confirming whether you need to build the environment by posting a question on the mailing list or chatting up on the IRC.

Assuming you need to build KDE from source, you will need to set up your build environment and install packages that contain header files used during the compilation of the KDE platform sources.

SETTING UP THE ENVIRONMENT

In order to set up the KDE platform build environment, you need to create a file defining environment variables and other settings by following the instructions at:

http://techbase.kde.org/Getting_Started/Build/Environment#Environment_Configuration

Save the file in the root directory for your build environment with the name .build-config. If you need separate build environments--such as one for stable releases and one for nightly builds--it is recommended that you create a separate script for each build environment and leave it in the environment's root directory.

BUILD REQUIREMENTS

This section details the software requirements you must install on your system before you can start building the KDE platform. For most of these requirements, it is best to use the packages provided by your operating system distribution, but in some cases you will need to build these requirements yourself.

Debian/Ubuntu

Minimum build dependencies on Debian/Ubuntu for KDE 4.6 and above are as follows:

```
sudo apt-get install graphviz libxml2-utils libopenexr-dev libjasper-dev libenchant-dev \
libavahi-common-dev libaspell-dev libasound2-dev libldap2-dev libsasl2-dev \
libsmbclient-dev libxkbfile-dev libxcb1-dev libxklavier-dev libxdamage-dev \
libxcomposite-dev libbluetooth-dev libusb-dev network-manager-dev \
libsensors4-dev libnm-util-dev libcfitsio3-dev libnova-dev libeigen2-dev \
libopenbabel-dev libfacile-ocaml-dev libboost-python-dev libsvn-dev libsvncpp-dev \
libqt4-dev libqca2-dev libstreamanalyzer-dev libstrigiqtdbusclient-dev \
libcommoncpp2-dev libidn11 libidn11-dev libpci-dev libxss-dev libxft-dev \
libpolkit-agent-1-dev libpolkit-backend-1-dev libpolkit-gobject-1-dev git libpoppler-qt4-dev
libspectre-dev
```

Extra and optional packages for Debian/Ubuntu can be found at:

http://techbase.kde.org/Getting_Started/Build/Distributions/Debian

openSUSE

Minimum build depends for openSUSE are as follows:

```
yast -i alsa-devel automoc4 avahi-devel patch cups-devel kde4-filesystem \ libbz2-devel avah
```

Extra and optional packages for openSUSE can be found at:

http://techbase.kde.org/Getting_Started/Build/Distributions/openSUSE

Other distributions

Please refer to http://techbase.kde.org/Getting_Started/Build/Distributions

11. OTHER WAYS TO BUILD THE KDE SC

You don't always have to build the KDE SC from bare sources. Many pre-packaged builds of the KDE SC are available. This chapter will focus on installing KDE SC from the git master without compiling it from sources.

PROJECT NEON

The wonderful Project Neon team provides daily builds of the KDE SC for developers using Kubuntu. Project Neon installs binaries and libraries in */opt/project-neon/*, which gives you a sandboxed installation of the master KDE SC alongside your stable KDE SC installation.

Get Project Neon from its Launchpad home page : https://launchpad.net/~neon. You can install the packages using the following command:

```
sudo add-apt-repository ppa:neon/ppa && sudo apt-get update && sudo apt-get install project-neon-base
```

More information on Project Neon can be found at:

https://wiki.ubuntu.com/Kubuntu/ProjectNeon

http://techbase.kde.org/Getting_Started/Using_Project_Neon_to_contribute_to_KDE

OPENSUSE

This GNU/Linux distribution provides frequent snapshots of the KDE SC master. The target is to provide snapshots on a weekly basis, but this is not always possible. These packages may not always include openSUSE specific patches, and are not available for the oldest openSUSE releases. You must add the core packages repository to use Extra or Unstable:Playground.

Version: 11.4

Core packages: http://download.opensuse.org/repositories/KDE:/Unstable:/SC/openSUSE_11.4/

Released applications:
http://download.opensuse.org/repositories/KDE:/UpdatedApps/openSUSE_11.4/

Extra: http://download.opensuse.org/repositories/KDE:/Extra/openSUSE_11.4_KDE_Unstable_SC/

Unstable:Playground:
http://download.opensuse.org/repositories/KDE:/Unstable:/Playground/openSUSE_11.4_KDE_Unstable_SC/

Version: 11.3

Core packages: http://download.opensuse.org/repositories/KDE:/Unstable:/SC/openSUSE_11.3/

Released applications:
http://download.opensuse.org/repositories/KDE:/UpdatedApps/openSUSE_11.3/

Extra: http://download.opensuse.org/repositories/KDE:/Extra/openSUSE_11.3_KDE_Unstable_SC/

Unstable:Playground:
http://download.opensuse.org/repositories/KDE:/Unstable:/Playground/openSUSE_11.3_KDE_Unstable_SC/

Version: Factory

Core packages:
http://download.opensuse.org/repositories/KDE:/Unstable:/SC/openSUSE_Factory/

Released applications: Use the application packages from openSUSE:Factory

Extra:
http://download.opensuse.org/repositories/KDE:/Extra/openSUSE_Factory_KDE_Unstable_SC/

Unstable:Playground:
http://download.opensuse.org/repositories/KDE:/Unstable:/Playground/openSUSE_Factory_KDE_Unstable_SC/

You can find more information about openSUSE's master KDE builds at:

http://en.opensuse.org/KDE_repositories#Unstable:SC_aka._KUSC_.28KDE_trunk.29

12. SCRIPTED KDE BUILDS (KDESRC-BUILD)

The easiest way to build the KDE Software Compilation (SC) from scratch is to run a script named *kdesrc-build* (formerly *kdesvn-build*), written by Michael Pyne. This approach is highly recommended for those new to building the KDE SC, because it takes care of almost the whole process for you. The builds remain compatible with the manual methods of building KDE SC, so you can change the modules you install later if you want.

kdesrc-build automates the following tasks and more:

- Performing the initial checkout
- Handling updates for modules that are already checked out
- Setting up the build system for the module
- Performing the build and install
- Specifying your CMake options or configure flags (so you don't have to remember them every time)
- Logging build errors so you can review them more easily for troubleshooting

This is not the be all and end all for your troubles building the KDE SC. Troubleshooting may still be required. Many errors that occur using other methods occur here too: you should read the log files that are stored for you.

USEFUL LINKS FOR FINDING MODULES

When using *kdesrc-build*, you may find it beneficial to search the source repository for modules. Sources include the following:

- To browse any of the various KDE SC projects using git, you can go to https://projects.kde.org/ or to http://gitweb.kde.org/.
- To browse the KDE Subversion repository, use http://websvn.kde.org/trunk/KDE/

SETUP

These subsections cover all the things you need to get and install *kdesrc-build* itself.

Prerequisites

kdesrc-build is fairly easy to install and set up, but you need to have the right software installed to build KDE SC. The requirements to build KDE SC are available at:

http://techbase.kde.org/Getting_Started/Build/Requirements

kdesrc-build requires Perl 5.8 or higher. It is installed by default with most distributions, and is included in the previous link. Check your version of Perl with:

```
perl -v
```

You will also need libwww , a collection of Perl Internet-related modules from https://github.com/gisle/libwww-perl.

Downloading and installing kdesrc-build

Once your system is set up and able to compile the KDE SC, you can download *kdesrc-build* from its website, http://kdesrc-build.kde.org. The file you download will contain (at least) the *kdesrc-build* script and a sample configuration file.ut Installing *kdesrc-build* is as simple as saving the file and making it executable. If you'd like, you can move it to a directory in your PATH, however for this example we'll put it into the KDE SC source directory that we use (*~/kdesrc*).

```
mkdir -p ~/kdesrc &&
cd ~/kdesrc &&
tar xjvf ~/path/to/kdesrc-build-1.12.tar.bz2 &&
cp kdesrc-build-1.12/kdesrc-build .
```

Alternatively, the newest kdesrc-build script (and sample config file) can be pulled down directly using git:

```
git clone git://anongit.kde.org/kdesrc-build.git ~/kdesrc
```

SETTING UP THE CONFIGURATION

Configuration options for *kdesrc-build* are taken from a file named *~/.kdesrc-buildrc*. Directions for editing this file are at:

http://techbase.kde.org/Getting_Started/Build/kdesrc-buildrc

For the most part the defaults in the included kdesrc-buildrc-sample should be sufficient. You can copy it to your home directory as follows:

```
cp ~/kdesrc/kdesrc-build-1.12/kdesrc-buildrc-sample ~/.kdesrc-buildrc
# Now edit the ~/.kdesrc-buildrc
```

Note that the config file name begins with a leading dot (.), making it a hidden file. If you are using Dolphin or Konqueror from the desktop, you may need to show hidden files in order to find the configuration file and edit it. Alternatively, you can edit the sample file before copying it to *~/.kdesrc-buildrc*.

Also, make sure that the modules you'll want to build are included. You'll want the following at the least:

- qt-copy
- kdesupport
- kdelibs
- kdepimlibs
- kdebase

Modules are built in the order they appear in your *~/.kdesrc-buildrc* file, so the first module should be qt-copy. kdesupport should be before kdelibs, which should be before any other KDE SC module, and so on.

The sample configuration file does include these modules by default. So you won't need to make many changes unless you'd like to add some modules to the build by uncommenting them.

If a module you'd like to build isn't already present, simply add the following to the end of the *~/.kdesrc-buildrc:*

```
module module-name
end module
```

module-name is whatever the module is called in the software repository (for instance, kdemultimedia).

You may want to enable the `make-install-prefix` option if you are installing KDE SC or Qt to a directory that is not in your home directory. Through `make-install-prefix`, you can run *su* or *sudo* during the `make install` process so you can install files as root, or set certain programs to execute with higher permissions. (This is required for certain programs to execute properly.)

```
module kdelibs
  make-install-prefix sudo -S # sudo with no stdin
end module

module kdebase
  make-install-prefix sudo -S
end module
```

Git-based modules

Most of the KDE SC modules are in git, although a few of them still need to be ported from svn to git. Eventually, all KDE SC modules and projects will be in git.

To build these modules in *kdesrc-build*, you just need to add a couple of lines to the module configuration. For example, konversation is developed in the Git repository at:

https://projects.kde.org/projects/extragear/network/konversation/repository

So to add this module, write the following at the end of *~/.kdesrc-buildrc*:

```
module konversation
    repository git://anongit.kde.org/konversation
    branch master
end module
```

In this case I selected the "master" branch since that is the default git branch.

Now whenever you build konversation, *kdesrc-build* will use git instead of Subversion.

USEFUL KDESRC-BUILD COMMANDS

kdesrc-build is driven from the command line, so here's a guide to some of the more useful command line options:

Option	Effect
`--pretend` (`-p`)	This option is like a dry run. *kdesrc-build* will process the options and its configuration like normal, and run through the build as normal, but instead of downloading or running the build it will just print messages about what it would have done. You should always run with `-p` before running the script for real, to make sure it is doing what you expect.
`--nosvn` (`--no-src`)	This option skips the source code update step. This is useful if you're running *kdesrc-build* again soon after a previous update and don't want to wait just to find out there were no changes.
`--refresh-build`	This option causes *kdesrc-build* to delete the current build information for the modules given on the command line and build them again from scratch. This option adds a lot of time but offers the best chance of a successful build.

Any non-option arguments on the command line are assumed to be modules to build (and are built in the order provided on the command line). If no modules are specified, all of the modules listed in the ~/.kdesrc-buildrc are built, in the order listed in the file.

BUILDING THE KDE SC

We're almost there. If you're happy with your settings, it's time to test out *kdesrc-build*. In theory things are as simple as running kdesrc-build and then coming back later.

```
cd ~/kdesrc
./kdesrc-build
```

You may want to test it by building just qt-copy first, however.

```
cd ~/kdesrc
./kdesrc-build qt-copy
```

If the build failed (*kdesrc-build* will error out with a nice bright red error message), there are several possible things to check for:

1. You may be missing a key piece of required software (such as a development library)
2. The KDE SC code being compiled may be broken in some fashion so it won't build. This is commonly due to newly committed code that worked on the developer's machine, or occasionally because it is Monday (when incompatible changes are permitted to kdelibs).
3. *~/.kdesrc-buildrc* may not be set up properly. You may be trying to install to a directory that you do not have permission to access, for instance, or you may have specified a system qtdir that does not exist.
4. The module may depend on a newer version of qt-copy or kdelibs (or another module). In this case you'll have to run *kdesrc-build* to update the out-of-date module first.

How do you find out what the error was? The output of the failing command will be in the log directory. By default, all log output is in the *log* subdirectory of the KDE SC source directory. The log directory is laid out like this: *log/date-run/module/output-file.log*. To simplify finding the appropriate file, a couple of symlinks are created:

`log/latest`

> Contains the debugging output from the last time *kdesrc-build* ran (--pretend does not affect this).

`log/latest/module/error.log`

For instance, if qt-copy just failed to build, you could read the output like this:

```
cd ~/kdesrc
kwrite log/latest/qt-copy/error.log
```

Replace *kwrite* with your preferred editor. Hopefully the output can guide you to resolving the problem. For instance, if the failure is CMake output saying you're missing a library, install that library and try again. For link errors, you can try running with the `--refresh-build` on the module (or if that doesn't work, on required libraries like qt-copy and kdelibs).

If you're stumped by the error, you may want to wait a day and try updating again, and hope that the reason for the error has been fixed. You can also try mailing the kde-devel mailing list to see whether others know about the problem or have had similar issues.

RUNNING YOUR NEW KDE INSTALLATION

Assuming you got enough of the modules to build and install to have a working KDE SC installation, you'll still need to set up your environment correctly to run it. *kdesrc-build* doesn't help you out here (yet), so you should follow the instructions at:

http://techbase.kde.org/Getting_Started/Using_an_IDE_with_KDE4

Make sure to use the same paths as the ones you defined in *~/.kdesrc-buildrc*. For the KDEDIRS and KDEDIR variables, use the setting of the "prefix" option (in the global section). For the QTDIR variable, use the setting of the "qtdir" option.

KEEPING YOUR KDE INSTALLATION UP TO DATE

Keeping your KDE installation up to date is as simple as running *kdesrc-build* again. Every build has these phases:

1. Update the source code for all modules being built.
2. Build and then install all the modules.

Old build directories are not deleted by default, so the build after a small update will not normally take as long as the initial build of a module. This is called "incremental make". However it may be necessary at times to perform a full rebuild, due to inconsistencies between the build directory configuration and changes to the source directory. You can use the --refresh-build option to force a full rebuild.

For more information on how to take advantage of *kdesrc-build*, see the documentation at:

http://kdesrc-build.kde.org/documentation/

The site describes all of the module options and command line options available for *kdesrc-build* and gives tips on how to perform various useful tasks.

13. TROUBLESHOOTING YOUR KDE BUILD

Compile and Linking errors are frequent sources of discouragement. Make careful note of the first occurrence of an error in your build process. It could be as simple as a bad environment variable, an unexpected version of a library or missing prerequisite. Please read the instructions carefully. Check for spelling errors while using module names as well as commands.

Please review your logs and do searches for fixes. If you cannot find a solution then please ask for help on IRC or a Mailing List.

ADVANCED TOPICS

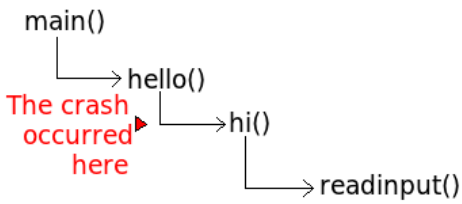
14. READING BACKTRACES

14. READING BACKTRACES

A backtrace (also called a *stack trace* or *stack traceback*) is a report of how the program has called different functions as it goes along. It is commonly used during interactive and post-mortem debugging. It can also be displayed to the user of a program as part of an error message, which a user can report to a programmer.

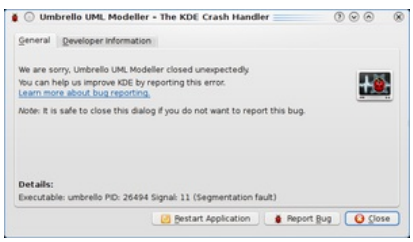
Each function puts a *stack frame* on the stack containing its arguments and other information it needs to run. The active stack frames reflect a certain point in time during the execution of a program. A stack trace allows you to track the sequence of nested functions called up to the point where the stack trace is generated. In a post-mortem scenario, the stack trace goes up to, and includes, the function where the failure occurred. Be aware, however, that the function where the failure occurred might not be responsible for the failure; an error could well have been embedded in a higher function (for instance, by passing an incorrect value to the function where the program failed).

The following figure illustrates a stack frame, where `main()` called `hello()`, which called `hi()`, which called `readinput()`. A stack trace is likely to work down from the last call to the first, so that `readinput()` might appear first.



Backtraces are essential. They may look meaningless to you, but they might actually contain a wealth of useful information. A backtrace describes which functions were called prior to the crash, so that developers may track down in which function the mess started. Exact memory addresses can also help locate problematic data, such as in a core dump (a file left behind when a program fails, containing the contents of live memory at the time of the failure). But producing good backtraces has a downside: libraries and executables occupy much more disk space than their optimized counter parts that can't provide the information to produce a backtrace.

The KDE Crash Dialog (Dr. Konqi) should appear right after a crash.



Opening the "Developer Information" tab will display the relevant backtrace. This process may take some time and a lot of memory, so things may go sluggish all of a sudden. But the result should look something like this:

```
Using host libthread_db library "/lib/libthread_db.so.1".
[Thread debugging using libthread_db enabled]
[New Thread -1232783168 (LWP 7604)]
[KCrash handler]
#6  0x0806be76 in TreeMapItem::parent (this=0x0)
    at /home/bram/KDE/kde3/kdeaddons/konq-plugins/fsview/treemap.h:285
#7  0x08065fea in TreeMapItemList::compareItems (this=0xbfec04a8, item1=0x0,
    item2=0x0)
    at /home/bram/KDE/kde3/kdeaddons/konq-plugins/fsview/treemap.cpp:720
#8  0xb7281619 in QGList::operator== () from /usr/qt/3/lib/libqt-mt.so.3
#9  0x0806d498 in QPtrList<TreeMapItem>::operator== (this=0xbfec04a8,
    list=@0xbfec0468) at /usr/qt/3/include/qptrlist.h:74
#10 0x08062e18 in TreeMapWidget::mousePressEvent (this=0xbfec03ac,
    e=0xbfebff1c)
    at /home/bram/KDE/kde3/kdeaddons/konq-plugins/fsview/treemap.cpp:1840
#11 0xb7004a63 in QWidget::event () from /usr/qt/3/lib/libqt-mt.so.3
#12 0xb6f6bca7 in QApplication::internalNotify ()
   from /usr/qt/3/lib/libqt-mt.so.3
#13 0xb6f6ca88 in QApplication::notify () from /usr/qt/3/lib/libqt-mt.so.3
```

```
#14 0xb7725a84 in KApplication::notify (this=0xbfec055c, receiver=0xbfec03ac,
    event=0xbfebff1c)
    at /home/bram/KDE/kde3/kdelibs/kdecore/kapplication.cpp:550
#15 0xb6f0bfd2 in QETWidget::translateMouseEvent ()
   from /usr/qt/3/lib/libqt-mt.so.3
#16 0xb6f0b8b0 in QApplication::x11ProcessEvent ()
   from /usr/qt/3/lib/libqt-mt.so.3
#17 0xb6f1b761 in QEventLoop::processEvents () from /usr/qt/3/lib/libqt-mt.so.3
#18 0xb6f82831 in QEventLoop::enterLoop () from /usr/qt/3/lib/libqt-mt.so.3
#19 0xb6f826b6 in QEventLoop::exec () from /usr/qt/3/lib/libqt-mt.so.3
#20 0xb6f6b72f in QApplication::exec () from /usr/qt/3/lib/libqt-mt.so.3
#21 0x0805181e in main (argc=134673960, argv=0xffffffff)
    at /home/bram/KDE/kde3/kdeaddons/konq-plugins/fsview/main.cpp:55
```

In this backtrace, the first stack frame is shown on line #6. Because the stack is unwound from the end back to the beginning, we can see that the call that crashed the program was `parent()`, which was called by `compareItems()` on line #7, which in turn was called by the overloaded == operator on line #8, and so on.

After the line number, the hexadecimal number starting each line is the address in memory where the stack frame starts for each function. Unless you have a core dump, this is not useful to you. More interesting are the lists of arguments and the addresses of their data in parentheses. Thus , line #6 shows that `parent()` was called with a single argument, `this`, whose value was 0 (`0x0` in hex). Of course, the name `this` is assigned to the object on which the method was invoked. So the `parent()` method was actually called without arguments. Methods in object-oriented languages are passed the pointer to the object on which they were invoked as their first argument. So `compareItems()` on line #7 was called with two arguments, but because `this` was passed as the first argument, three are shown in parentheses.

On line #6, the string "(this=0x0)" indicates that the `parent()` function is being called with a NULL pointer. Of course, any program will crash if it tries to retrieve data from, or put data into, an address to which it doesn't have access. The address 0x0 on virtually every computer system is reserved and unavailable to the program, so you can tell that reading from or writing to a NULL pointer will cause a crash. You can also see, in the documentation for the Qt function `parent()`, that it is called without arguments (so the problem was not caused by a bad argument) and returns a pointer to the parent of the object on which it is called. Therefore, the developer should try to figure out what object `parent()` was called on and why the parent could not be returned.

APPENDICES

15. USEFUL TOOLS
16. KDE DEVELOPER GUIDE FREQUENTLY ASKED QUESTIONS (FAQ)
17. GENERAL GLOSSARY
18. USEFUL LINKS
19. ABOUT THIS BOOK

15. USEFUL TOOLS

This chapter lists some of our favorite tools for software development, debugging. building, and other activities. Some are discussed in other parts of the book as well.

CORE TOOLS

These are absolutely required to build and develop KDE software. On Linux, they will typically be provided by your distribution. On other platforms, packages should normally be available, often for download directly from the home page of the tool. Of course, the standard development tools, such as a C/C++ compiler and some sort of text editor, are also required.

CMake - CMake is the build system of choice for the KDE SC. Once you have this, you can use it to configure a software project for building, and that process will tell you of any other requirements you are missing.

Git - Most KDE SC projects are developed in Git, so you will need it to get the source code in the first place. You can find the relevant Git URLs at the KDE SC projects directory. It will be helpful to follow the git configuration instructions.

Subversion - The KDE SC still uses Subversion for some things, notably translations.

DEBUGGING AND ANALYSIS TOOLS

These are used to analyze a program and do such tasks as profiling.

Valgrind - Valgrind helps to find memory leaks and uninitialized memory blocks. It also includes a profiler and more. Valgrind is one of the most important development tools.

The GNU Project Debugger (GDB) - GDB helps you find problems in source code by allowing you to set breakpoints, step through the code, look at stack traces, etc. It keeps evolving, but we currently recommend version 6.x. Graphical frontends are available. See also the debugging tutorial "Debugging with GDB"
at http://techbase.kde.org/Development/Tutorials/Debugging/Debugging_with_GDB

KDbg and DDD - KDbg and DDD are graphical user interfaces to GDB.

MS Windows tools (Process Explorer, Console, WinDbg, DebugView, etc.) - More information about these is available on the "KDE on Windows" page on Techbase:

http://techbase.kde.org/Projects/KDE_on_Windows

DEVELOPMENT TOOLS

These are more basic tools that will help you with your development. We have discussed some of these earlier.

Qt Creator

This is the recommended IDE for KDE SC development, making it easy to write new programs and to integrate existing ones into an IDE. Further details can be found in this tutorial:

http://www.developer.nokia.com/Community/Wiki/How_to_use_Qt_Creator_IDE

KDevelop

KDevelop is the IDE created as part of the KDE SC for developing KDE and Qt C++ applications. It includes a an integrated debugger, a powerful editor with syntax highlighting, a Project wizard to create applications from templates, the automake/autoconf suite, and even the class documentation. Further details can also be found on the KDevelop wiki.

http://kdevelop.org/

Eclipse

Eclipse was developed for Java programming, but designed with a view toward extensibility that has made it popular for many programming languages. You can find out more at http://www.eclipse.org.

MS Visual Studio® Express IDE (Windows only)

Visual C++® Express is the free version of the Microsoft Visual Studio compiler from, and is officially supported by Nokia. It is a choice for compiling Qt and KDE applications on Windows. More information can be found at:

http://techbase.kde.org/Projects/KDE_on_Windows

Internationalization (i18n) Tools

These help you create applications that can be viewed and used in different languages and with the local conventions of different countries for currency, time, etc.

Lokalize

Lokalize is a computer-aided translation (CAT) system that focuses on productivity and quality assurance. It has the usual components for CAT tools: translation memory, a glossary, etc. It also includes a unique translation merging (synchronization) capability. It is targeted at software translation and integrates external conversion tools for freelance office document translation.

Dr. Klash

This reports conflicting shortcuts in menus. It's helpful for both translators and developers.

The x-test language

This language helps you find untranslated strings in applications. If you start your application with the "x-test" locale, all translated strings will appear with trailing and leading xx's.

HELPER TOOLS

These provide a variety of useful functions.

kde4-config

This helps a user to find out more about a KDE SC installation.

kfmclient

This lets you control the Konqueror browser through scripts.

kioclient

Command-line tool for network-transparent operations

kconf_update

This updates configuration files.

apidox tools

These help develop and generate API documentation for your code.

Automoc4

This is a helper tool that builds KDE SC4. It automatically generates moc-files.

svnmerge.py

This tool helps you keep track of merges between different SVN branches.

QUALITY ASSURANCE

Code Review

KDE uses the Review Board for performing code reviews; see the page on the Review Board on the kDE Techbase for more information:

> http://techbase.kde.org/Development/Review_Board

Continuous Building - Dashboards

Dirk's dashboard has all KDE modules, while the win32 dashboard has a selection available on Windows.

English Breakfast Network - Static Analysis

The English Breakfast Network is a collection of machines that do automated KDE source artifact quality checking. Basically, that means they have a git clone of the entire KDE codebase (including documentation and whatnot) and they run checking tools on that.

EBN uses:

- Krazy - Code Analysis
- APIDOX - API Documentation
- Sanitizer - DocBook Checker
- Usability Checks

16. KDE DEVELOPER GUIDE FREQUENTLY ASKED QUESTIONS (FAQ)

Q. I want to help/develop the KDE SC. How do I start?

A. A great way to get familiar with the codebase is to fix some bugs. You can find a number of "Junior Jobs" on the KDE Bugzilla. Search for the "Junior Jobs" link on the KDE bugzilla (http://bugs.kde.org) on the left.

Q. I do not understand what [insert strange term here] means.

A. Check the term in the *General Glossary and KDE Jargon*.

Q. What project should I contribute to?

A. Find something that interests you, and start hacking! Most people start by "scratching their own itches" first, which means that they try to fix the bugs that irritate them the most. See also the list of KDE SC projects in the section *Choosing a project*.

Q. How do I set up the Development Environment?

A. This is documented in the *KDE Development build environment* section of this book.

Q. What programming languages do I need to know?

A. Most of the KDE SC is written in C++, but KDE SC has bindings for Python, Ruby, etc. The Qt toolkit will be very useful to you in most of the KDE SC codebase.

Q. What programming concepts do I need to know?

A. An understanding of object-oriented programming (OOP) is valuable if you want to hack on the KDE development platform, but is not necessary for developing applications. You need also to know about software version control and the Qt communications concepts of signals and slots.

Q. How do I read backtraces?

A. This is covered in the *Reading Backtraces* section of this book.

Q. How do I use Valgrind?

A. Please refer to http://techbase.kde.org/Development/Tools/Valgrind for a brief overview of Valgrind.

Q. What's the best distro for KDE SC development?

A. Any distribution that provides the latest and greatest KDE SC packages is just fine for developing the KDE SC.

Q. How do I report a bug?

A. Use Bugzilla, our bug tracking software. A good bug report is thorough (containing all details that could be relevant, such as the operating system you're using, the versions of all related software, and the precise actions you were doing when the bug occurred). It should also be factual, polite, and clearly written. For a quick introduction to bugzilla, please refer to:

http://techbase.kde.org/Contribute/Bugsquad/Quick_Introduction_to_Bugzilla

For more Frequently Asked Questions visit http://techbase.kde.org/Category:FAQs

17. GENERAL GLOSSARY

A

Accessibility (a11y): The ability of all people, regardless of disability or severity of impairment, to use the features of a program.

Akonadi: KDE extensible cross-desktop storage service for personal information management (PIM) data and metadata providing concurrent read, write, and query access. Provides desktop-wide object identification and retrieval.

Akregator: KDE open source feed aggregator, supporting both RSS and Atom. Feeds can be sorted into categories, and there is an incremental search feature for the titles of all the entries in the database.

Algorithm: step-by-step procedure for calculations, data processing, and automated reasoning.

Amarok: *Rediscover Your Music* using KDE's Amarok. Core features such as the unique *context browser*, integrated Wikipedia lookup and lyrics download help users to find new music, and to learn more about the music they have.

Application programming interface (API): A particular set of coding rules and specifications that software programs can follow to communicate with another. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.

apidox: API Documentation

Applet: Program written in Java to be embedded in another environment, such as a Web page.

B

Backtrace: also called *stack backtrace, stack trace* or *stack traceback*. This is a report of the active stack frames at a certain point in time during the execution of a program. When using KDE software, one gets the best backtrace from Dr. Konqui, if it pops up after a crash. Otherwise, GDB can be used to get a backtrace.

Bot: Software applications that run automated tasks over the Internet, or in IRC channels.

Bug: An error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

Bug Squad: The team that keeps track of incoming bugs in KDE software.

Bugzilla: A web-based, general-purpose bugtracker and testing tool, used by the KDE community.

Build: Short for software build, which refers either to the process of converting source code files into standalone software artifact(s) that can be run on a computer, or the result of doing so. One of the most important steps of a software build is the compilation process, which converts source code files into executable code.

Build system: Software tools that script or automates a wide variety of tasks that software developers do in their day-to-day activities, such as compiling computer source code into binary code, packaging binary code, running tests, deploying code to production systems, and creating documentation and/or release notes.

C

Channel, IRC channel: The basic place to ask questions and get help in IRC. It is rude to direct questions to one person rather than asking the help channel in general.

CMake: an open-source build system that enables developers to automate compiling, testing and packaging of software based on specifications written in text files.

Code: Text written in a computer programming language.

Commit: To make a set of tentative changes permanent.

Compile: To use a compiler to process source code into executable code. Also a destination for messages logged by programs or the operating system, where administrators or developers can view the messages.

Console: A command-line interface (CLI).

C++: The coding language in which KDE software is primarily built.

D

Debugger: A computer program used to test and find bugs in other programs (the *target* programs).

Desktop: In graphical computing, a desktop environment (DE) commonly refers to a style of graphical user interface (GUI) derived from the desktop metaphor that is seen on most modern personal computers. The most popular modern Linux desktops are the KDE workspaces and GNOME.

Dependency: A package you need to install in order for your application to build and run.

Diff: A file comparison utility that outputs the differences between two files. Also refers to the output of such a program, which can be called a patch (since the output can be applied with the Unix program *patch*).

digiKam: an image organizer and editor that uses the KDE Platform. It runs on most known desktop environments and window managers, supports all major image file formats, and can organize collections of photographs in directory-based albums, or dynamic albums by date, timeline, or tags. Users can also add captions and ratings to their images, search through them and save searches for later use. With the plugins one can also export albums to 23hq, Facebook, Flickr, Gallery2, Google Earth's KML files, SmugMug, Piwigo, and Simpleviewer, or burn them onto a CD, or create a web gallery.

Distributed repository: A peer-to-peer approach to sharing and maintaining code or other collaborative work, in contrast to the client-server approach of centralized systems. Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a separate repository bound by a web of trust.

Distribution (Distro): a selection of packages that make up a working software system, and provided together to the user. Often applied to the GNU/Linux system in particular.

Docs: *documentation*, an essential part of the development process.

E

EBN: English Breakfast Network, a site dedicated to the contemplation of tea, KDE API Documentation Validation, User Documentation Validation, Source Code Checking, omphaloskepsis, and star-gazing.

Eclipse: A multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R, Ruby (including the Ruby on Rails framework), Scala, Clojure, Groovy, and Scheme.

Environment: A set of configuration scripts and bash commands provided as a recommended configuration when building KDE software manually.

e.V.: *Eingetragener Verein*, a registered voluntary association in Germany. The KDE e.V. is a registered non-profit organization that represents the KDE Community in legal and financial matters.

F

Forum: An online discussion site where people can hold conversations by posting messages. They differ from chat rooms in that messages are archived. A discussion forum is hierarchical or tree-like in structure: a forum can contain a number of subforums, each of which may have several topics. Within a forum's topic, each new discussion started is called a thread, and can be replied to by anyone who wishes to.

FOSS: Free and open-source software (F/OSS, FOSS) or free/libre/open-source software (FLOSS, FL/OSS) is software that is liberally licensed to grant users the right to use, study, share, change, and improve its design through access to its source code.

Framework: An abstraction in which software providing generic functionality can be selectively changed by user code, thus providing application specific software. It is a collection of software libraries providing a defined application programming interface (API).

Frameworks: Beginning with KDE Frameworks 5.0, KDE has a roadmap for the next major releast of KDE's libraries and runtime requirements, with an emphasis is on modularity, dependency clarity, simplification and increasing quality.

Freenode: An IRC network used to discuss peer-directed projects.

G

GDB: GNU Debugger, usually called just GDB and named gdb as an executable file. This is the standard debugger for the GNU software system.

Git: a distributed revision control system with an emphasis on speed and support for multiple branches shared among many developers. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Free software distributed as GPL v.2.

Gluon: a way of creating and playing games, and a means for players and makers of games to get together and talk about their shared interest. You can use the powerful Gluon Creator to build the games, interact with other makers and players of games on the GamingFreedom.org network site, and play them on any of the many supported platforms with one of the Gluon Player applications.

GPL: The GNU General Public License is a free, copyleft license for software and other kinds of works.

H

I

Integrated development environment (IDE): a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, a compiler and/or an interpreter, build automation tool, and a debugger. (Also known as *integrated design environment*, *integrated debugging environment*, or *interactive development environment*.)

Internationalization (i18n): The insertion of constructs that make it easy to change the interface and language of a program for different cultures and countries (see also **Localization**).

Internet Relay Chat (IRC): A text-based real-time communication tool. KDE channels are on *irc://irc.freenode.net*.

J

K

Kate: a text editor included in the KDE SC. The name Kate is an acronym for KDE Advanced Text Editor.

KDE PIM: KDE Personal Information Management, such as Kontact, KMail, KOrganizer, etc. Also, a work group within the larger KDE SC project that develops the individual Kontact applications in a coordinated way.

KDE SC: KDE Software Compilation, the sources for the KDE distribution

KMail: KDE email client that supports folders, filtering, viewing HTML mail, and international character sets. It can handle IMAP, IMAP IDLE, dIMAP, POP3, and local mailboxes for incoming mail. It can send mail via SMTP or sendmail.

Konqueror: KDE web browser and file manager. Provides file-viewer functionality to a wide variety of things: local files, files on a remote ftp server and files in a disk image.

Konsole: a free terminal emulator that is part of KDE SC. The KDE applications Konqueror, Krusader, Kate, Konversation, Dolphin and KDevelop use Konsole to provide embedded terminal functionality.

Kontact: KDE's personal information manager and groupware software suite. Supports calendars, contacts, notes, to-do lists, news, and email. Uses KParts to embed the various applications (KMail, KAddressBook, Akregator, etc.) into the container application.

Kopete: KDE's multi-protocol, free software instant messaging client.

Kpackage Kit: KDE's frontend for PackageKit. PackageKit is an open source suite of software applications designed to provide a consistent and high-level front end for a number of different package management systems.

Kparts: component framework for the KDE SC. For example, Konsole is available as a KPart and is used in applications like Konqueror and Kate.

Konversation: user-friendly Internet Relay Chat (IRC) client built on the KDE Platform.

KWin: the window manager that is an integral part of the KDE SC. It can also be used on its own or with other desktop environments.

L

LAMP: acronym for a software bundle or platform consisting of Linux, Apache, MySQL and Perl/PHP/Python.

Licensing: legal instruments (usually by way of contract law) that govern the usage or redistribution of software. All software is copyright protected, except material in the public domain.

Localization (l10n): making the changes required to display a program's interface using the language and conventions of a particular country (see also **Internationalization**).

M

Mailing list: A collection of names and addresses used by an individual or an organization to send material to multiple recipients. Often extended to include the people subscribed to such a list, so the group of subscribers is referred to as *the mailing list*, or simply *the list*.

N

Nepomuk: KDE workspaces project that supports annotations, indexing, search, and linking.

Nick: a user's screen name or online handle.

Nightly: a neutral build that reflects the current state of the source code checked into the version control system by the developers, as built in a neutral environment (that is, in an environment not used for development). A nightly build is a neutral build that takes place automatically, typically each night. Project Neon is such a project for KDE.

O

Object-oriented programming (OOP): a programming paradigm using *objects* – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs.

Ocular: KDE's universal document viewer based on KPDF.

Operators (IRCops): Channel operators have powers over the IRC channel, including moderating or kicking out disruptive users. *IRCops or sysops* control the IRC server, so they control the channels as well as having control over who can participate. On most systems, ops are identified with a symbol next to their nicks, but Freenode discourages ops from appearing as such unless they have work to do in the channel.

P

Package, packaging: There are two types of packages that may be downloaded from the KDE FTP site: binary packages (rpms, debs, and the like) and source packages. Binary packages are compiled ("runnable") versions of KDE SC that are built to run on a specific OS or distribution. Source packages are the raw code that makes up KDE SC, and need to be compiled before they can be used. KDE software packages available from the distributions may be slightly different from the pure KDE source packages.

Pastebin: a web application that allows users to upload snippets of text, usually samples of source code, for public viewing.Use is encouraged in IRC channels, where pasting large amounts of text is considered bad etiquette. KDE's pastebin is hosted at http://paste.kde.org

Patch: software designed to fix problems with, or update a computer program or its supporting data. This includes fixing security vulnerabilities and other bugs, and improving the usability or performance.

Phonon: multimedia API provided by Qt; the standard abstraction for handling multimedia streams within the KDE SC.

Plasma Active: the latest initiative of the Plasma team, bringing KDE functionality to mobile devices.

Plasma: KDE SC framework to facilitate the creation of widgets. These cover interactive application launchers, window and task managers, and more.

Plasmoid: widget in the Plasma Desktop environment.

Post-mortem debugging: Debugging after a crash report has been filed.

Q

QtCreator: An integrated, cross-platform IDE for C++ and JavaScript that is part of the Qt SDK.

Qt: Cross-platform application framework that is widely used for developing application software with a graphical user interface (GUI).

Quassel: cross-platform, distributed IRC client, meaning that one (or multiple) client(s) can attach to and detach from a central core -- much like the popular combination of screen and a text-based IRC client such as WeeChat, but graphical.

R

Rekonq: KDE web browser based on WebKit.

Reviewboard: web-based collaborative code review tool, available as free software under the MIT License. An alternative to Rietveld and Gerrit, Review Board integrates with Bazaar, ClearCase, CVS, Git, Mercurial, Perforce, and Subversion.

Review: systematic examination (often as peer review) of computer source code. It is intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills.

S

Script: small program written for a command interpreter or another scripting language.

Server: computer program running to serve the requests of other programs, the *clients*.

Solid: device integration framework for KDE SC. It functions on similar principles to KDE's multimedia pillar Phonon; rather than managing hardware on its own, instead it makes existing solutions accessible through a single API.

Source: Human-readable instructions in a programming language, to be transformed into machine instructions by a compiler, interpreter, assembler or other such system.

Sprint: face-to-face meeting of team members who usually work together remotely.

Suite: collection of computer programs, usually application software and programming software of related functionality, often sharing a more-or-less common user interface and some ability to smoothly exchange data with each other.

Summit: in KDE and FOSS, a large meeting for members who usually work remotely. Team sprints may take place before, during and after a large summit.

SVN (Subversion): A software versioning and a revision control system distributed under a free license, part of the Apache Foundation.

T

Techbase: KDE's developer documentation wiki.

Telepathy: A realtime communication framework that supports instant messaging, VoIP, and collaboration.

Terminal: interface for serial entry and display of textual data. See also **console**.

Testing: investigation conducted to provide stakeholders with information about the quality of the product or service under test.

Text editor: program used for editing plain text files.

Toolchain: set of programming tools that are used to create a product (typically another computer program or system of programs). The tools may be used in a chain, so that the output of each tool becomes the input for the next, but the term is used widely to refer to any set of *linked development tools*.

Toolkit: set of basic building units for graphical user interfaces. KDE SC uses the Qt toolkit.

Trunk: the unnamed branch (version) of a file tree under revision control. The trunk (or *master*) is usually meant to be the base of a project on which development progresses.

U

Unit tests: method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In object-oriented programming a unit is usually an interface, such as a class.

Usability: ease of use and learnability of a human-made object, in this case, our software.

Userbase: KDE's user documentation wiki.

V

Valgrind: GPL licensed programming tool for memory debugging, memory leak detection, and profiling. The name valgrind comes from the main entrance to Valhalla in Norse mythology.

Variable: symbolic name given to some known or unknown quantity or information, for the purpose of allowing the name to be used independently of the information it represents. A variable name in computer source code is usually associated with a data storage location and thus also its contents, and these may change during the course of program execution.

Version control: Revision control, also known as version control and source control (and an aspect of software configuration management or SCM), is the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision".

W

Widget: element of a graphical user interface (GUI) that displays an information arrangement changeable by the user, such as a window or a text box. The defining characteristic of a widget is to provide a single interaction point for the direct manipulation of a given kind of data. In other words, widgets are basic visual building blocks which, combined in an application, hold all the data processed by the application and the available interactions on this data.

Wiki: website that allows the creation and editing of any number of interlinked web pages via a web browser using a simplified markup language or a WYSIWYG text editor. Wikis are typically powered by wiki software and are often used collaboratively by multiple users.

Word processor: computer application used for the production (including composition, editing, formatting, and possibly printing) of any sort of printable material.

X

X, X window system: computer software system and network protocol that provides a basis for graphical user interfaces (GUIs) and rich input device capability for networked computers. It creates a hardware abstraction layer where software is written to use a generalized set of commands, allowing for device independence and reuse of programs on any computer that implements X. **X.Org** serves as the canonical implementation of X, and is what KDE SC uses.

Y

Yakuake: drop-down terminal emulator based on KDE Konsole technology.

Z

Z-machine: virtual machine used by Infocom for its text adventure games. Kwest is a Z-machine interpreter for KDE.

18. USEFUL LINKS

KDE: http://kde.org/

The KDE Project home page

KDE Techbase: http://techbase.kde.org/

The KDE Development wiki

KDE Userbase: http://userbase.kde.org/

The KDE User wiki

KDE Identity: https://identity.kde.org/

A single sign-in system used across several KDE websites

KDE Projects Page: https://projects.kde.org/

An overview of all projects within git.kde.org that are based on KDE technology

Qt Tutorials: http://doc.qt.nokia.com/

An extensive reference to Qt documentation

Git in 30 minutes: http://blip.tv/scott-chacon/git-in-30-minutes-4064151

A useful video that introduces the concepts revolving around git

The Dot: http://dot.kde.org/

The official KDE community news outlet

Behind KDE: http://behindkde.org/

People Behind KDE interviews the people who work on KDE

Planet KDE: http://planetkde.org/

Aggregation of KDE community member blogs

The KDE Release Schedule: http://techbase.kde.org/Schedules

19. ABOUT THIS BOOK

Acknowledgments

A few people have really helped us make this book what it is so this book is incomplete without a vote of thanks and a hug of appreciation to them. So , in no specific order:

- Gunner, from Aspiration (http://www.aspirationtech.org/), inspired and made us all smarter, as did the other teams at the sprint, namely OpenMRS, OpenStreetMap, and Sahana Eden.
- Nóirín Plunkett & Belinda Lopez provided amazing writing and editing resources for all of us.
- Adam from Floss Manuals was endlessly helpful, and the Booki team worked with us on needed functionality and bug-fixing as the week and work went along.

We also thank the KDE community, who provided for us both an atmosphere in which we could grow and thrive as contributors and human beings, but also for providing wonderful documentation from which we could steal with wild abandon.

In particular, Ingo Malchow, Lydia Pintscher, and the KDE-Promo team gave us great help in our time-pinch.

The Team

This book was first created at a three-day book sprint in October, 2011, at the Googleplex in Mountain View, California. Carol Smith of Google Summer of Code fame got us the needed funding to fly Karan, Supreet, and Rohan from India, and to house and feed us while we worked. Rohan turned 21 during the sprint, and the group celebrated with an X-box party!

Rohan Garg - rohan16garg@gmail.com
Supreet Pal Singh - supreetpal@gmail.com
Karan Pratap Singh - wizard.karan@gmail.com
Valorie Zimmerman - valorie.zimmerman@gmail.com
Andy Oram - andyo@oreilly.com

KDE team at the Doc Sprint, October 2011