The User Manual to KDevelop

The Reference Guide to the KDevelop Integrated Development Environment for Unix Systems, Version 1.0

Ralf Nolden <*Ralf.Nolden@post.rwth-aachen.de*>

The KDevelop Team

Version 2.1 , July 7,1999

Contents

Chapter 1

Introduction

As everything on earth has it's fashions, today's computer world seems to tend more to the use of free software even for commercial purpose. The most popular free software project is Linux. It is now generally agreed that Linux, (as well as other projects such as the Apache web server, the Perl language and GNU suite of tools), proves that free software can be of as high a quality as commercial software. But despite the quality, end users must still suffer under the cryptic commands of any Unix Systems.For Linux to thrive, it needs applications, both free and commercial, and ease of use.

The KDE project tries to close this gap by providing an easy to use desktop and the companion libraries to extend the variety of available GUI-based software. But especially as free software is often made in the author's free time, the question for many programmers is how much they like the current code development environment. KDevelop wants to take another major step: making the programmer's life easier and more efficient: products created with KDevelop can reach a higher level of reliability and functionality in the same development period.

To fulfill this goal, the KDevelop Integrated Development Environment provides many features that developers need as well as it wraps the functionality of third party projects such as `make` and the GNU C++ Compilers and makes them an invisible, integrated part of the development process. KDevelop manages:

- All development tools needed for C++ programming like Compiler, Linker, automake and autoconf,
- KAppWizard, which generates complete, ready-to-go sample applications,
- Classgenerator, for creating new classes and integrating them into the current project,
- File management for sources, headers, documentation etc. to be included in the project,
- The creation of User-Handbooks written with SGML and the automatic generation of HTML-output with the KDE look and feel,
- Automatic HTML-based API-documentation for your project's classes with cross-references to the used libraries,
- Internationalization support for your application, allowing translators to easily add their target language to a project,
- WYSIWYG (What you see is what you get) -creation of user interfaces with a built-in dialog editor,
- Debugging your application by integrating KDbg,

- Editing of project-specific pixmaps with KIconEdit,
- The inclusion of any other program you need for development by adding it to the "Tools"-menu according to your individual needs.

KDevelop makes it a joy to work with all programs in one place and saves time by automating standard development processes as well as giving you direct and transparent access to all information you need. The integrated browsing mechanisms are designed to support documentation requests that developers have in conjunction with their project.

The class viewer and error finder bring you anywhere in the project code with a mouse click without needing to search for files; File trees give direct access to the project files and the integrated help system offers superb access to online-documentation from anywhere within the IDE.

1.1 Changes

Since the last release, the main work was done in three different areas: the class viewer, the dialog editor and the "look-n'-feel" in general, including new editing functions and the creation of KDE-applications. In order to accomplish all this, we are proud to welcome the newest two team members, Jonas Nordin, who designed a full-featured class parser and -viewer, and Pascal Krahmer, who joined the team to implement an integrated dialog editor. Other work was done in various parts of the IDE, such as a new `grep`-dialog allowing expression search through all project files or even the whole system; extended configuration utilities; reviewed and extended documentation as well as a new project generator.

1.2 About this Handbook

This user manual gives the user a complete overview of the KDevelop IDE and describes the basic development process in brief. For more information about specific programming issues, we suggest reading the *KDevelop Programming Handbook* included with KDevelop, which covers themes such as understanding generated application frameworks and how to create full-featured KDE applications using example projects.

The design of this handbook is therefore separated into the following parts:

- Chapter 2 (Installation), covers the system requirements, installation and setting up of the KDevelop IDE.
- Chapter 3 (Programs), tells you how programs are created with standard GNU development tools and how they are built.
- Chapter 4 (Development with KDevelop), takes you on a short tour describing the main functionality of the environment.
- Chapter 5 (Overview), explains the menus and dialogs as well as keyboard shortcuts.
- Chapter 6 (The Help System), shows how to use the built-in documentation browser and additional help functions.
- Chapter 7 (Working with the Editor), explains the features of file management and editing.
- Chapter 8 (Projects), describes the creation and maintaining of software projects.

- Chapter 9 (Build Settings), contains a reference on how to set Compiler and liker flags as well as project options.
- Chapter 10 (The Class Browser), tells you how to make use of KDevelop's powerful Class Browser.
- Chapter 11 (The Dialog Editor), features the integrated visual GUI constructor and how it generates C++ output.
- Chapter 12 (General Configuration) shows how you can set overall preferences for using KDevelop.
- Chapter 13 (Questions and Answers), covers questions that regard to the usage of KDevelop under different flavors of Unix systems and problem solutions in general.

For programming beginners and new users of this product we recommend to read yourself into this manual before starting to actually work with the IDE as it covers the usage in depth. The understanding of how things are done the quickest way will save you a lot of time searching for functions and features as it will enable you to make use of first-class development tools even more simple.

Chapter 2

Installation

2.1 How to obtain KDevelop

KDevelop can be found either on the KDE Applications page at `<http://www.kde.org/current.html>` or on the KDevelop homepage at `<http://www.kdevelop.org>`. KDevelop is also available on Linux distributions, such as SuSE 6.1.

We're also offering snapshots of the KDevelop CVS repository on our homepage for those who want to stay up to date with KDevelop. Usually, the snapshots are not intended to be used for production but as a test for new features and to give an insight into development progress of the KDevelop team. Also we offer various third party software needed by KDevelop directly such as KDoc and KDbg.

If you're experiencing problems with compiling or using KDevelop, please read the 13 (Questions and Answers) section of this handbook or the FAQ-file included with the KDevelop package. If your problem is not addressed, please subscribe to the KDevelop mailing list at *kdevelop@fara3.cs.uni-potsdam.de* by sending a mail with an empty header and "subscribe" as contents. Requests and problem report should only target the usage of the KDevelop IDE and not towards any questions that regard any implementation problem you may have while coding your own application. Anyway, all mails send to the mailing list should be written in English, so that all participants can take part on discussions and are able to provide better help. The mailing list is also intended for those users willing to contribute and who found solutions for any problems they experienced, so we can fix errors and include that knowledge to give beginners an even more qualified first-hand help. An good way to report problems is to send the output you get by starting kdevelop from the console or to copy and paste the contents of KDevelop's internal Messages-window.

2.2 Requirements

In order to successfully compile and use KDevelop, you need the following programs and libraries which are available on most platforms as distribution packages and thereby can be installed easily.

Required:

- g++ 2.7.2/g++ 2.8.1/egcs 1.1 (or compatible), available at `<http://www.gnu.org>`
- GNU make (or compatible), available at `<http://www.gnu.org>`
- perl 5.004, available at `<http://www.perl.com>`
- autoconf 2.12, available at `<http://www.gnu.org>`

- automake 1.2, available at `<http://www.gnu.org>`
- flex 2.5.4,
- GNU gettext, available at `<http://www.gnu.org>`
- Qt 1.42, available at `<http://www.troll.no>`
- KDE 1.1.x, available at `<http://www.kde.org>`

Optional:

- a2ps or enscript for printing support
- ghostview or kghostview for printing preview
- glimpse 4.0 for the search index, available at `<http://glimpse.cs.arizona.edu>`
- sgmltools 1.0, available at `<http://www.sgmltools.org>`
- KDE-SDK (**KDE** **S**oftware **D**evelopment **K**it), which includes KDoc, KSgml2Html, KTranslator (available at `<http://developer.kde.org>`)
- KDbg, available at `<http://members.telecom.at/~johsixt/KDbg.html>`
- KIconEdit (available at `<http://www.kde.org>`)

KDevelop was tested with SuSE Linux 5.2 on an AMD K6 200, 64MB RAM and FreeBSD 3.0-Release as well as SuSE Linux 6.0 on an Intel 200 MMX, 128MB RAM.

As far as known to the authors, SuSE Linux and FreeBSD contain all necessary packages, including a2ps and enscript as packages or rpm's, so you should have no problem installing the required third-party software.

Documentation:

For creating the KDE library documentation, you need the kdelibs package in source available on your system as provided by the KDE project or included with the source packages of your distribution and **KDoc** (included in the KDE-SDK).

Also we're offering a C/C++ Reference on our homepage at `<http://www.kdevelop.org>` that is integrated into the documentation browser after it's installation. Download the package and copy the source file as `root` into your KDE directory and untar it with `tar zxvf c_c++_reference.tar.gz`, then the reference is available in the documentation tree; otherwise selecting the reference book in the browser shows an error page with the KDevelop homepage URL offering a download and describes the installation process.

2.3 Compilation and Installation

In order to compile and install KDevelop on your system, type the following in the base directory of the KDevelop distribution:

```
% ./configure
% make

(as root)

% make install
```

Since KDevelop uses `autoconf` you should have not trouble compiling it.

In order to compile the KDevelop CVS snapshot, type the following :

```
% make -f <idx/Makefile.cvs/
% ./configure
% make

change as ‘root‘ and type:

% make install
```

If your system's make-command is `gmake`, type `gmake` instead of `make`.

2.4 Starting KDevelop

If you use KDE as your window manager, KDevelop can be started by choosing "K"→"Applications"→"KDevelop"→"KDevelop 0.4". As KDevelop supports KDE-Mime-types, you can also start by selecting a KDevelop project file ("*.kdevprj", displayed with the KDevelop project icon) in the KDE File Manager which will start KDevelop and load the project. Under other window managers, open a console and type:

```
% kdevelop
```

To start KDevelop with an existing project, change into the project-directory and type:

```
% kdevelop  <yourProject>.kdevprj
```

Under each user account KDevelop will invoke the 2.5 (automatic installation) process on the first start, allowing a quick configuration of the most needed options. If your installation is messed up, you can reconfigure KDevelop any time either by entering

```
% kdevelop  --<idx/setup/
```

or, when using KDE, by choosing "K"→"Applications"→"KDevelop"→"Setup".

2.5 The Installation Program

KDevelop includes an automatic installation program module which is invoked whenever Kdevelop is started and the configuration file kdeveloprc does not exist. We suggest you follow the installation steps by choosing "Proceed" to automatically check your system and to set up your KDevelop environment.

The Installation dialog's buttons execute the following actions:

Help: Will open the KDEHelp program.

Proceed: This starts the installation process and executes the following actions:

1. Checking for make/gmake, autoconf, autoheader, automake and perl for the creation and compilation of new applications generated by KDevelop. If gmake is installed, the make-command will be set automatically to use gmake. Other commandline options to your make program can be set in the setup dialog accessed by the options menu later, an introduction into development under Unix is explained in section 3 (Programs).

2. Checking for KDoc and glimpse. Those will allow to create a new KDE-Library documentation and a search index automatically in a later setup step if found.

3. Checking for a2ps and enscript to ensure that printing is available. Either one of these programs have to be installed to allow correct printing. If neither program is installed, you can do this at any time later on at your option without having to run the setup again.

4. Checking for KDbg, KIconEdit and KTranslator. We encourage you to install those programs as they are good helpers for creating complete KDE-applications. Note that KDbg is used directly for debugging your current project within KDevelop; KIconEdit is used to display and edit pixmaps selected in the file-viewer trees. The programs KDbg, KIconEdit and KTranslator are then configured to be available in the "Tools" menu of the KDevelop menu bar if they are found. Other tools can be added to the Tools-menu by selecting "Tools..." in the Options menu later.

5. Summary of detected programs: the installer lists those programs it found and those it didn't. Additional hints are given, if a needed program is recommended.

6. Detection of your Qt-Documentation path: this checks for several standard paths in your system for the documentation and sets the path automatically. If your Qt-Documentation could not be found either because it is not installed or your system keeps it at a different location, a message will appear that asks you to set the correct path manually or to continue. Choosing the button to set the path will return to the main installation window and shows an editing field with a button on the right of it to choose the path. Usually this is in the qt/html directory. After doing so, the installation can continue with the selection of "Proceed" again.

7. If KDoc was detected, you are asked to create/update your KDE-library documentation. For that, you need to have the kde-libraries in source form. For Linux-Users who have installed KDE from a distribution, we suggest to copying and extracting the source of the kdelibs-package to your system; Free BSD users who installed the kdelibs as a package should look for their according distfile package of the ports-collection. If none of these cases match your situation, you should download the sources from `<http://www.kde.org>` and untar the sources on your system. If you wish to use the documentation package provided by `<http://developer.kde.org>`, cancel the creation and continue with creating the search index. After the installer finished and KDevelop has been started, set the path to the extracted documentation package in the KDevelop Setup dialog and run the search index setup again, which can be found at the same setup page of the dialog.

   The "Update KDE-Library documentation" dialog is set up to use a default path for the location of the documentation in $(HOME)/.kde/share/apps/kdevelop/KDE-Documentation. The only thing you have to do is choosing the path to your library sources you extracted on the system with the selection button on top of the dialog and press OK. As an example, if you got the kdelibs.tar.gz package from `<http://www.kde.org>` and downloaded it to your home directory, you should open a console or terminal and enter "tar zxvf kdelibs.tar.gz". This will untar the sources into a directory $HOME/kdelibs, which then contains the sources for each library in a subdirectory, e.g. for kdecore, this would be in /home/rnolden/kdelibs/kdecore. Now, the path you have to enter in the KDE-Library documentation dialog would be the path to all libraries, in the example /home/rnolden/kdelibs. After pressing the OK button, a message in the installation window shows that the documentation generation is in progress and you should wait for the next message.

NOTE: On a multi-user system or systems with disk-quotas for user accounts, the installation of a complete HTML documentation for each user would be a waste of disk space. In this case, ask your system administrator to run KDevelop under the root account to allow write access to the system's root directory. Then install the documentation within the KDE-directory, under $KDEDIR/share/apps/kdevelop/KDE-Documentation. The correct path can then be set up later in the KDevelop setup dialog, available in the "Options"-menu.

8. If the system check found the glimpse program on your system, you are offered to create a search database. The search database can be created with the options of including the KDE-Documentation and the Qt-Documentation (default). Additionally the KDevelop documentation is included and will be indexed. If you have other documentation that you want indexed you can select the directories and add them to the indexing process as well.
9. During the creation of the search index, the installation window will show a message that this is in progress.
10. If all installation steps were done correctly, a final message states that KDevelop will be started after pressing OK. Mind that you can set up additional options like auto saving in the 12 (KDevelop Setup) dialog that can be accessed through the options menu.

Cancel: Will show a warning message that the installation process is going to be canceled. This warning message allows you to return to the installation ("Back") or to start KDevelop with the default values ("Continue"). Mind that in this case you will have to set all options yourself with the configuration dialogs provided in the options-menu.

Chapter 3

Programs

Now that KDevelop installed successfully and the most commonly used options are set, you are probably wondering if it keeps what it promises. This Chapter gives you a guideline to how programs are created using the GNU tools in general and especially what part KDevelop plays in this game.

3.1 The Compiler

The Compiler is actually the program on your system that has to be installed as a minimum to create running programs; he is the one that compiles the source code into object files and creates the program.

Normally, you would start like this: Open an editor of your choice - don't use a word-processor. Type in something like this to create the source for your first program:

```
#include <iostream.h>

int main(){

cout << "Hello World" << endl;

}
```

Well, actually all the program will do is to print out the string "Hello World" to your standard output. But this is just the source code for the program to be build, not the program itself. Therefore, we need a Compiler, in this case a C++-Compiler like g++. Then we can save the file with the source code, as, let's say, myprogram.cpp and invoke the Compiler with the filename (on a console):

```
g++ -o myprogram myprogram.cpp
```

Then we can start our program- just type `myprogram` on the console, and the program prints out the string; then exits.

3.2 Make and Makefiles

I have everything I need: an editor, a Compiler and I can execute my own C++ program. But it isn't all that easy. What happens if you have more than one source file ? And, do you have to compile all sources over and over again only if you changed one file ? Compiling will become more and more complicated and time-consuming, because you have to type in all the commands and options yourself. Therefore, you could write a so-called "Makefile". You could also call it some other name except the name of the program to build. Then, you should have the tool `make` or `gmake` installed, or any other tool that is capable of keeping track of a project's compilation. Insert all your Compiler commands in a certain syntax into that Makefile and save it; then you will only have to type `make` or `gmake` on the console in the directory where your Makefile is located, and then make takes over, leading the Compiler to create your application. The make utility has many other advantages and can be used to a lot of purposes. To get a complete overview, open a console and type:

```
man make
```

or search for "GNU Make" in KDEHelp, "System GNU Info contents". At least, you have an insight, why a developer needs the make utility for making it easier to compile his application. Now, writing Makefiles is not only handwork until now, you also have to dig yourself into the whole syntax and options. But here is the good news about KDevelop and any Make-utility: You just have to set the Make-Command in the 12 (KDevelop Setup) dialog, and then you're done. All projects generated with KDevelop will use that Make command to build the target application, and no typing at all. Just hit the buttons on the toolbar of KDevelop, beginning with the one after the second separator line, or choose the desired function for Make in the "Build" menu.

The toolbar and the build-menu then offer the most-common functions that you need to let make do the dirty work:

- Compile File: is only active if you're working on a source file. It invokes make with the correct command to compile only the current source.
- Make: just calls make and creates your target.
- Rebuild all: rebuilds the whole project
- Clean/Rebuild all: cleans the project directory first and then runs make again.
- Stop Build: cancels the current process- this is mostly used if you watch make working and staring at your sources. Then- ahh- I forgot about this line...and you have to correct your code. Just hit Stop, correct the error you found by yourself and run Make again.

But this is not the only way how KDevelop works together with make- for KDE applications, there are some things that are special, like creating the message files for internationalization. These functions are also included, so no worry about these things anymore.

Until now, you know about sources, the Compiler and why make is needed. In the next section, we'll discuss how it comes that projects created with KDevelop automatically can be compiled on most other Unix-platforms using the `configure`- script.

3.3 Configure

The title of this section lets you probably question: Configure ? What has to be configured ? Or who ? Well, assume you have written a program including a Makefile. Then you like to distribute it, but the binary compiled does only run on your system or on systems that are compatible with yours. To support other platforms like different Unix-systems or machines like Alpha's or RISC's, you have to recompile the program. The easiest way would be to copy the source package to the target machine and run `make` again. But what if the target machine uses another Compiler command or has in some other way a problem to build your binary ? Not to mention more difficult issues like installation paths for your program and documentation- e.g. KDE can be installed in opt/kde/ on one machine, while it is installed under usr/local/kde/ on another. In this case, you would have to rewrite the Makefile each time to ensure a correct compilation and installation of your product.

Fortunately, GNU-tools have even more to provide than that mighty `make`- the commonly used automake and autoconf packages. It sounds good to hear something with "auto"- seems like something about application design can be done quick and easy, which exactly hits the point.

Automake's purpose is generally to create a so-called `Makefile.in` from a file `Makefile.am` which you have to write for your project. This Makefile.am consists of macros which can be interpreted and reduce the complexity that make offers, so a Makefile.am is written safer and quicker than the final Makefile.

Having this said, who is finally creating me my Makefile ? Now, here comes autoconf. Autoconf requires several macro files for the project. That are those Makefile.in's generated by automake and a file called `configure.in`, also containing macros. Hereby the Makefile.am and .in's are containing macros that are responsible for the way how to build the software in terms of which sources have to be compiled, which files belong to the package and what name the final binary or library will have after a build. Configure.in on the other hand contains macros for what the final configure-shell script will check for on the system configure is executed. Those could be e.g. the Compiler command, required libraries against which the final binary will be linked, include-files the project needs and their location.

For example you want to write a KDE application. After writing your sources, you want to distribute the program to the user community, and each user has to compile the binary on his own. Then you would write a configure.in file that contains the macros for a KDE-compliant application. That one macro finally expands to a check on the system whether the Qt-library is installed, checks for the Qt-header files, the KDE-libraries and headers etc.

Summary: To create a GNU-compliant application that is portable to different Unix-OS's and machines other than yours, you will need to do the following:

1. write the sources of your project
2. write a Makefile.am for each subdirectory, including the main project directory of your project
3. write a configure.in file placed in the main project directory containing the macros for system requirements
4. run automake
5. run autoconf

Then the main work is done. Automake creates the Makefile.in's, autoconf processes the configure.in and Makefile.in's and generates an executable shell script called `configure`. All you then have to do is to execute it with `.`configure/ and the script will run the checks of your choice. Finally Makefiles will be generated that allow a final execution of make (or gmake) that will process all Makefiles and then you're done.

This seems to be a lot of stuff for writing a small application and much to learn especially how to write correct macros. But even the fact that you provide a compilation on almost all Unix-systems will be worth this work sooner or later. Finally, you only have to do this work once for your project and in case your project's files increase you only have to add the filenames to the macros.

Now, how far does KDevelop support this kind of application development and how complicated does it get for the programmer ? The good news is, that you don't even have to know anything about macros and scripts. All details are hidden behind an easy to use graphical interface doing the work for you. An application is therefore created with GNU tools in a very user-friendly way:

Just generate your application with KAppWizard, by the choice of your application's needs- may it be a pure C++ terminal application or a kind of GUI program using Qt or the Qt/KDE libraries. All work is done automatically and your project already contains the Makefiles that are created by an auto-execution of the GNU-tools and the configure-script.

This is it- you're ready to extend the source of your project, may it be by adding classes, dialogs, translations or documentation, which is also completely automated. Just concentrate on the real work of the developer, which is providing functionality for the final application that you want to create. In most cases, you probably won't come in touch with Makefiles at all when using KDevelop.

3.4 Debugging

The following section covers a term that is widely used by developers: Debugging. It means, that, although your Compiler produces the final application, your application may not run or crash during execution due to a so-called "bug" in the code. A program error described by the name of this insect comes from the history of computing; one of the first errors that caused a machine to crash was not obviously a malfunction- bugs were inside the computer which were responsible for it. Therefore, an error not detectable on the first look is called a "bug", so "debugging" means to throw out bugs where they shouldn't be. Now, you don't have to hunt them for real; assuming that today's computers are designed to keep them out by some kind of outer protection. They have to be found inside the code, mostly ending the execution of a program with the message "Segmentation fault". GNU provides another tool called `gdb`, the GNU debugger. This terminal program allows to watch the internal values of an application and the execution step by step with setting "breakpoints" in the code. Gdb stops the execution every time the program comes to a breakpoint while executing. But like most other tools, the debugger is handled by another program providing a frontend to it, allowing to easily watch values and the setting of breakpoints in the code.

For this purpose, your project's application is by default created with a Compiler option for debugging, thereby storing additional data in the executable to allow the localization of values and lines in the code. As a third-party frontend to gdb, KDevelop makes use of KDbg, the KDebugger. To debug your program, you just have to select "Debug" in the Build-menu or press the according toolbar button displayed by a wheel having glasses over it, signaling that you want to watch the execution.

KDevelop then opens the Tools-window and starts your application with KDbg. The KDbg interface appears inside the Tools-window and allows the usage just like you started it from outside.

In general, the above steps are clearly showing the need of certain steps that a developer has to do when starting to write his own application, and cover issues that are common to all projects. Also, we explained what part KDevelop does for a developer and how it supports the idea of providing an easy way to Unix programming. To get further information about the role and purpose of GNU tools, you should read the documentation provided with them, commonly accessed via the `man` command or by the "System GNU Info contents" section in KDEHelp.

Chapter 4

Development with KDevelop

In this chapter you will get a general overview how you can use KDevelop and the internal dialog editor to create own applications. Thereby, all tools are described by their use during the development process. If you are a beginner in C++/GUI design and programming, you should read *The KDevelop Programming Handbook*, which describes some basics for program design and shows a more detailed use of KDevelop by sample projects which you can follow step-by-step to get used with the way applications are created using KDevelop. Also you should have read chapter 3 (Programs) to gain a general understanding why Unix applications should make use of provided development tools; this will make things easier when you hit references to what Makefiles are etc.

4.1 What are KDevelop Project Applications ?

Projects created with KDevelop make it very easy for developers to use the GNU standard development tools. In opposition to the self-creation of Makefiles, which have to be written anyway because of the project's complexity, those provide not only a better way of Makefile generation but also a good and safe way to provide a fast adaption towards different systems by autoconf-generated `configure` scripts.

The distribution of your applications does not require the end-user to have anything different installed than a C++ Compiler and the development libraries, which is most often the case; but you can as well distribute binary packages of your application. In either way, the end-user of your product does not have to have KDevelop installed. For giving away your sources, we advise to include the project file of KDevelop as well, as this makes it very easy for other developers to work with your sources if they use KDevelop as well. For projects where several developers, maybe working on different places, are involved, this should the case anyway to ensure consistency of the Makefiles so you don't run into trouble. Especially on multi language applications, translators won't actually work with the source code, except in cases that require corrections for enabling translation support. Anyway, those will be thankful as KDevelop simplifies their work to a minimum by adding their language and reducing the work to concentrate on the translation.

With using KDevelop for your own projects, you would start creating a new application with the KAppWizard. There, you can also set the project type according the goals of your program. Then, you would start developing the user interface by adding widgets that are already constructed in the libraries and only have to be called by your application, or by self-constructions that can be made with the dialog editor. Next comes the extension of the user interface by changing and extending those parts that are already made by KDevelop like the statusbar, menubar and toolbars. As resources are collected in a resource file, this can be extended simply by adding new values to the already existing one; the given resources can also be used as a guideline for adding your specific ones. After implementing the functionality to methods that are either generated empty by the dialog editor or already containing code for standard user actions, you should extend the User Manual by simply editing the provided SGML index file towards describing your applications capabilities. The last step would be to generate translation files and to distribute your project to translators doing the rest to enhance internationalization.

But even for non-GUI application designers the IDE offers an easy way to create new projects; the editor and Class Browser in conjunction with KDevelop's powerful project management will reduce the time for development to the minimum that is required for implementation of functionality.

4.2 Short Description of KDevelop's Tools

With KDevelop you have chosen a package, that, in conjuction with the use of other tools, will provide you a prefect and flexible environment for object-orientated application development under Unix-Systems. Generally, KDevelop consists of the programming environment and the dialog editor; besides that all needed programs that can be helpful for designing applications are embedded in one or the other way.

4.2.1 Programming Environment and Dialog Editor

The Programming Environment

The Programming Environment is the user interface that KDevelop provides for creating and maintaining projects, implicitly using many other tools by simplifying their use. It's build-in Editor and Helpbrowser as well as the Class Browser reduce the amount of work that development processes usually burden on the programmer. The ease of use make writing applications for Unix an enjoyment on it's own and will probably result in shorter release cycles and better development improvements. From here, you're managing your project throughout all its details, keep the overview over your classes and don't have to use additional consoles for your work, as well as the integration of all user interfaces results in the fact that you only have to work with one window under X and don't have to watch for several ones spread all over your desktop.

The Dialog Editor

KDevelop's build-in dialog editor offers a simple way of designing user interfaces with the Qt and KDE libraries. It's simple way to do almost everything with the mouse allows a rapid construction of dialogs and main views for applications and their direct transformation into C++ code, which is directly added to the project. As the dialog editor is embedded into the rest of KDevelop's user interface, you have the same commands available than working in Programming mode. This allows e.g. construction of a dialog, generation of the code output and it's direct testing within the program by the availability of the make-commands and you still can control the Compiler output without having to switch back to the programming view. For a detailed description, see 11 ().

4.2.2 KAppWizard and the Classtools

The KAppWizard and the Classtools provided for project generation and automatic code extension are intended to make the creation of applications as easy and safe as possible and offers a good way for beginners in Qt/KDE programming to start actually working themselves into the details of GUI application design, as results can be achieved very quickly.

KAppWizard

KAppWizard is intended to create full-functional application projects that are ready-to-run by using GNU-standard tools and, for GUI-programs, taking advantage of the KDE and Qt libraries. By specifying only the information that is required as a minimum, users are enabled to start working at a new project within seconds. Calling KAppWizard should be the first step to be done for creating a new application with KDevelop.

KAppWizard provides you a so-called application skeleton with all needed C++ source code files for a running application. After you specified a projectname and set the needed preferences, your new project will be generated and you have a complete application that already contains a lot of functionality (for KDE and Qt projects), without even having to type one line of C++ code yourself. When specifying the complete set of options the program generator offers, your application already contains:

- an SDI-Interface (for working with one document per application window), based on the document-view-controller model
- Menus and dialogs for opening and saving files as well as printing dialogs
- a toolbar and statusbar already providing complete functionality
- a helpsystem, containing an SGML-based user manual and statusbar help
- a complete HTML-based API (application programming interface) documentation for the current state of the project
- an installation routine though make

The KAppWizard is available by the Project-menu, entry "New...".

Classtools

The term "Classtools" describes a variety of functions that are available for the developer to work on a project not only by an object-orientated language but using the sources in an object-orientated way. Working with classes instead of having to watch for files offers a very flexible way to extend a project and lets the user concentrate on the objects the project contains. The tools that KDevelop offers are a Classgenerator and a Class Browser, where each one provides a lot of automated functionality.

Creating a new class with the Classgenerator After generating a project with the KAppWizard your work as a developer would be to add a set of classes that match the design of the application. The typical process without using KDevelop would be to create a new header and sourcefile, add them manually to the according Makefile.am and to start writing the classes declaration and implementation. To simplify the whole process, the Classgenerator lets you do this by only specifying the new class by its name, inherited class, inheritance attribute and further information such as class documentation by it's purpose for the project. Additionally, you can also change the filenames that are preset by the generator and if the class would be Qt-signal/slot enabled by default.

The rest of the work is done automatically- you don't have to take care for Makefile.am, files and the class itself. A new "make" invocation is enough to include the new class in the compilation process.

As the Class Browser updates itself after the addition of a class, the declaration and implementation is available at once and you can start working on the extension of the class. The Classgenerator is invoked by the Project-menu, entry "New Class".

The Class Browser The Class Browser displays all types of objects and functions of your project in a tree left of the editing windows. A class parser scans all files for methods and classes and makes them available by graphical symbols. Over those, context-menus offer a specialized functionality to work with the sources by focusing on the classes and functions.

Selecting a class will result in opening the file that contains the declaration; on methods and functions this will show you the implementation. For a full description, see 10 (The Class Browser)

4.3 The Development Process

The development of a new application with KDevelop can generally be divided into two major steps: first, you have to generate a program skeleton with KAppWizard, then the "rest" of the development has to be done using KDevelop's features, which only requires your abilities as a programmer. To describe the development process, we assume you want to create a Qt/KDE application, which implies using most of KDevelop's features for working with these types of project, so you have a general "HOW-TO".

4.3.1 Creating a Program Skeleton

To start creating your application, you normally would call the KAppWizard, enter the project name and generate the program. By this, the wizard automatically creates a project file for the application and inserts the KDevelop areas into the Makefile.am's which will be used later when extending the application. The project file is the file you have to load for opening the project in later sessions.

You call KAppWizard by the "Project"-menu, entry "New...". When the wizard appears, you have to specify the project type on the first page. Selecting "Next >>" brings you to the next page where you have to insert the project name, version, directory and your personal information. Page 3 offers more detailed options, page 4 and 5 allow editing file headers for header and implementation files that are inserted automatically according to your selection. If you already have file headers you want to use, you can choose those files as well. The last page contains an output window and an error message window. When selecting "Create", your application will be generated and the actions that are executed can be seen in the output window. If READY appears in the output window, the KAppWizard is finished and you can select "Exit" to return to the Programming Environment. To build the binary, choose "Make" from the Build-menu or choose the according button in the toolbar. You can as well test the functions already present by choosing "Execute" from the Build-menu.

4.3.2 Developing an Application

This section describes the development phase of applications with KDevelop and the dialog editor- and all work can be done within the IDE.

The development steps are generally to edit the source files, the compilation of the project sources followed by the Linker process binding all object files to the final binary. Then errors have to be located which either prevented a compilation or linking or are semantical bugs that can be found by a debugging session. Finally, the documentation has to be extended and translations have to be added.

But as all those steps can mix which each other, it is not that easy to describe a general way how to develop your own application. Typically, the usual procedure would be to first create all visible parts of the project like the main view and the dialogs for configuring any options, then generate the source code and implement all needed connections that still have to be done like changing the menubar and toolbars, adding statusbar help for new menu entries and fill the new member functions with objects that are displaying your dialogs. Then you would compile your application and test it's capabilities, fix errors and test it again. The following sections describe how this would be done; for your particular application you may choose another way- KDevelop leaves you enough flexibility to decide what to do at what time.

User Interfaces

The User Interface of an application is actually what the user sees and by which he interacts with the program. Therefore a program generally has a menubar, toolbars and a statusbar as well as a main view which would be e.g. a text input window for an editor. KAppWizard generated applications already contain a complex functionality on graphical interfaces- the frame structure takes care of standard elements such as bars and buttons. When developing your application, you have to extend the given structure to give the program the user interaction capability you want to provide. One part of the work is the design of dialogs e.g. for changing values like the brush width in a paint application. This can be done easily with the dialog editor inside KDevelop. Now, how do you create those good-looking dialogs ? As the Qt-library is a GUI toolkit, it provides a base set of so-called "widgets" like labels displaying plain text, lineedits for text input and buttons for navigation and selection commands. Besides these "low-level" parts of user interfaces, standard dialogs are provided and are ready to use such as printing dialogs. The KDE-libraries then are based on the Qt-library and contain replacements for certain dialogs as well as additional widgets that can be used in your application. So, when you start designing your applications behavior towards the user, you should make yourself comfortable with the widgets provided with Qt and KDE. The online-documentation of Qt offers screenshots for most of them, so you should have a look there first. Then we advise to have a closer look at the KDE-UI library, offering other widgets. Libraries like the KHTML library contains very specialized widgets that a lot of programs make use of. It just simplifies application design like the predefined dialogs provided with Qt. For your own dialogs, those low-level widgets are the one you need. A dialog merely consists by a collection of different widgets combined together building the interface and is technically programmed by C++ code in a class that is derived from `QWidget` or a more specialized dialog class that inherits `QWidget`. The program using the widget therefore needs an object of the widget's class- this is almost all you have to understand how dialogs or views are used. The dialog editor of KDevelop now offers a nice way to simply construct dialogs and user interfaces visually instead of you having to combine dialogs with pure C++ code together- this is the hard way to create interfaces. Switch to the dialog editor by choosing the according entry in the "View"-menu (or by the toolbar button). You will see that KDevelop's face has changed but for e.g. the menubar and the toolbar are very similar. This makes it very easy to switch back and forth between the two working modes and you will feel comfortable in both after some time. There you can construct your views and dialogs as you like and set all available preferences for the items the view will contain. When you're finished, select "Generate Sources" from the Build-menu; the dialog editor and the project management will take care of the rest. You can test if everything went OK by selecting Make or Execute, this will build your application including your new sources. But don't expect that dialogs are already usable- this is a matter of implementation that is the usual way a developer works. Don't worry- this isn't too difficult, too. For more information about creating dialogs, see 11 (The Dialog Editor), examples and guidelines can also be found in *The KDevelop Programming Handbook.*

Binding New Elements

After you have created your user interfaces and generated the sources, you are ready to make your application use them by objects. As described above, a usual GUI-based program contains bars and the main view; additionally the main view operates with the bars, which are part of the main application class, and the document object which it is connected to. In terms of object-oriented design, you would describe the given architecture as the "Document-View-Controller"-model. This model describes the basic functions for objects in an application towards their role within the program. The Controller represents the one coordinating the main application and provides user interaction through a menubar and additionally toolbars and a statusbar. The Document class takes the task of representing a document the user works with. Therefore, a document class should do all actions like loading files and saving them again. The view now is in the center of the application window, showing the user a part of the document visually and provides all functions to manipulate the data by the user. As the bars and toolbars are already present, your work would be to create the main view and additional dialogs by which the user can change any settings or has access to additional functions.

To build your main-view, the generated source code for your application already contains a class in the form of <YourApplication>View which inherits the `QWidget` class (this is the minimal class that handles visual interfaces in Qt and therefore in KDE as well). In general, there are three different ways how to extend the given class:

- remove the document-view structure and use one of the predefined "big" widgets already containing a lot of functionality- just remove the View-class and replace the view-object by another.
- change the inheritance of the View-class. Change the inheritance to e.g. QMultiLineEdit and your application would be an editor.
- if your main view will consist of several separated parts, create the widgets and their classes you need with the dialog editor and create objects of these classes in the view-class constructors that in combination will build the main view.

For dialogs, things are a bit different. The usual way would be to call a dialog through a menubar entry and additionally by a toolbar icon. After you constructed your dialog and generated the sources, the class is available for creating an object that will be the dialog. So you first have to look for a suitable place in the menubar to add an entry which will open the dialog when selected by the user. If the already given menus do not match your needs, create a new popup menu just like the others and insert your entry and the slot you want to call when the menuitem gets selected. Then implement the slot by creating an instance of the dialog class and call the dialog with the member functions given by the base class. Also you have to specify an ID for the menuentry. The frame applications have all IDs already given collected in a file resource.h, so you only have to add the ID and give it a new number. Then you're done- additionally you could construct another toolbar icon and add the statushelp message. Your application now offers a new functionality which is visible to the user. Now you have to add the implementation of methods that your dialog will be operating with to manipulate any values. Finally, call your make-tool or "Execute" and the changed files will be recompiled; the result of your changes can then be tested immediately.

The Complete Development Process

Above, we discussed the usual way how to start working on a new application using KDevelop and how to extend the user interface. Now, these are the standard steps where the IDE helps you, but KDevelop does more than providing tools for creating applications and their visual parts. The following gives a short description of the main functionality that is offered for enhancing application development.

Sourcecode Management The KDevelop IDE provides a lot of methods to the programmer to achieve his goals within the shortest time. As described above, the KAppWizard and the dialog editor shorten the time you would usually need to get to the same result manually. But this didn't cover the work a programmer has to do usually: working on the implementation of his application to provide a proper execution by the end-user. Now, why is KDevelop the IDE you would want to use as a coding environment, including it's use to create even non-GUI applications ?

Implicitly, the IDE keeps track of your project in general; that means, you don't have to take care for saving changes, the Makefile generation and so on- this is providing full project-management, and KDevelop plays out all it's power here in any case of C++ application development. It is easy to understand that, after taking away the management from the programmer, he is more concentrated on working out the sourcecode. On the other hand, the code is usually spread over many different files across the project, so you can separate certain parts. But this means still working the hard way- being the developer, you still would have to struggle with creating these files and writing standard contents like file headers containing the date and the author's name as well as e.g. license terms for the code. Additionally, this requires that you have to remember where your functions, class declarations and implementations are in the project. Therefore, KDevelop contains the Classtools- a variety of actions, that allow fast work and moving the developer's focus from files to objects- classes, structures and methods. The classgenerator lets you create a new class including inheritance, attributes and documentation easily . For object-oriented work, the Class Browser brings you to the location of your objects; it doesn't matter any more where the code is actually. The Browser scans all sources automatically and rebuilds itself after additions to keep up with your work and enables you to access the new code directly. By context-menus, the browser offers even more functionality, like bringing you to the implementation or to the declaration of member functions. Then, the addition of members are done graphically by dialogs- no search for the file and the place you would have to add your entries. Finally, you can get an even more specialized view of your project's classes by the Classtool dialog, providing you trees that show the usage, contents and inheritance of classes and their objects. For more detailed information, see 10 (The Class Browser).

Building and Executing your Application The KDevelop IDE is specially designed to take away all those steps that you have to do periodically like building and executing your program as well as locating errors in the sourcecode.

You're starting the build-process by:

- clicking on the symbols "Build" or "Rebuild All" in the toolbar
- or selecting "Build"/"Rebuild All" from the Build-menu.

To execute your application, choose

- the symbol "Execute" or "Debug" (starting KDbg with your program) from the toolbar
- by the according menu entries in the Build-menu.
- or by "Execute with Arguments" to start your application with additional arguments

For more information about the build-process, see 8 (Projects).

Searching Program Errors As naturally errors occur either during the build-process (which are detected by the Compiler and are called syntactical errors because they result from a wrong syntax of the sources) or during the execution of the application, those have to be found and removed by the programmer. For locating errors, the developer needs the information what exactly caused its occurance. As mentioned, the Compiler is able to detect syntax errors himself, resulting in that the executable can't be build (this can also happen by the Linker when he detects "unresolved symbols"- see 9.4 (Linker Options)). As he gives out a description of the error as detailed as possible, the error can be found and removed. During the build-process, you will see the output window pop up showing you what your make-tool and the Compiler has to say. In case of errors or warnings, just press the mouse button over the error line and the editor will open the affected file and sets the cursor to the errorline. This can also be done by the entries in the "View"-menu, "Next Error" and "Previous Error" or by the according keyboard shortcuts. You will find that this is very functional and saves a lot of time to get to the error, thus for removing the cause your knowledge as a programmer is asked.

Runtime Errors, which occur during execution and will mostly result in a segmentation fault, are sometimes hard to find. Therefore, you can enable the Compiler to add information within the binary to watch the execution by the sourcecode. A debugger then is another program that lets you do this by starting the application and letting you set breakpoints in the code where the execution stops, so you can control critical implementations and by this detect at least the line that caused the error. Finding the real cause is another task; it depends on the programmer to locate this. The **gdb** is a debugger that is provided with the GNU Compiler and programs like *ddd* or *KDbg* are frontends that allow an easier use. KDevelop therefore uses KDbg and lets you invoke the debugging process by the commands "Debug" in the Build-menu or by the "Debug" symbol in the toolbar. For more information about debugging applications, see 8 (Projects) and *The KDevelop Programming Handbook*.

4.4 Additional Information

The topics addressed within this chapter are described im more detail within the documentation provided with the KDevelop IDE and in other documentations:

- *The KDevelop Programming Handbook*, covering a complete introduction into GUI application design and programming with the Qt and KDE libraries,
- this handbook, sections 5 (Overview) to 8 (Projects), describing all avaliable functions within KDevelop,
- this handbook, section 11 (The Dialog Editor), addressing the use of the dialog editor to create your own widgets,
- The *Online-Reference Documentation* to the Qt-library, which covers examples of using the Qt GUI-toolkit as well as a class reference and screenshots for the most important ready-to-use widgets included,
- The *KDE-Library Class-Reference*, generated automatically by KDevelop from the KDE-Library sources, containing descriptions for all classes and widgets including example code for their use,
- on the Internet, see:
  - `<http://www.troll.no>` for information about Qt and additional third-party widgets,

- <http://www.kde.org> for information about the KDE project and developer guidelines,
- <http://developer.kde.org> for additional references to KDE application development.
- on the KDevelop homepage <http://www.kdevelop.org>

Chapter 5

Overview

Looking at KDevelop, the user interface can be described by it's logical separation: the main window, the treeview and the output window, surrounded by the menubar, toolbars and the statusbar. This section describes the purpose of each part of the interface, starting by the windows, followed by the bars and the functions they provide.

5.1 Main Window

The main window consists of four tabulators for which the left two are for editing purpose. First comes the Header-/Resource window for headers and any other text files like the documentation SGML's, then the C/C++ window for sourcecodes. These are followed by the Documentation window displaying in HTML-format. At last is the Tools window for external programs like KIconEdit and KDbg which are embedded into the IDE. Any Tool that is registered can be accessed via the Tools-menu; whereas the registration of third-party programs can easily be done by a configuration dialog (see 12 (General Configuration)). For a description of the Editor windows functionality, see chapter 7 (Working with the Editor), and for the Helpbrowser, you should look at section 6.4 (Using the Documentation Browser).

5.2 The Class Browser and Fileviewers

5.2.1 The Class Viewer

The Class Viewer (CV) shows the classes of your project as well as global functions and variables. Opening the tree will show all member functions and members with symbols for attributes (private, protected and public, signals and slots), so you can see the properties of members visually without switching to the header file. Selecting the class name will open the according headerfile containing the class and set the cursor to the class declaration beginning. Selecting member functions will open the implementation file and place the cursor at the function header. For elements, KDevelop will look for the header file and place the cursor at the line the element is declarated. The classviewer also provides popup-menus that offer more specified options, see 10 (The Class Browser) for more detailed information.

5.2.2 The Logical File Viewer

The Logical File Viewer (LFV) allows sorting files by filters in groups. Individual groups can be added by the properties entry of the project menu or by a right click on the tree items. This allows a more specialized search for files that you want to access quickly, in the first place only showing project-included files. According to their Mime-type, the files will be opened when selected. You will like the LFV for things like selecting pixmaps- this will start KIconEdit (if installed on the system) in the Tools-window and opens the chosen picture.

5.2.3 The Real File Viewer

The Real File Viewer (RFV) shows the project directory tree with all files, so you can edit non-project files or files hidden by the LFV like configure.in and Makefiles. Popup-menus also provide functions like adding or removing files to the current project.

5.2.4 The Documentation Tree

The Documentation Tree (DOC) displays all available HTML-based documentation that is configured as books. Selecting a book will open the first page in the Browser window. Also, a popup-menu provides personal configuration for additional HTML-based documentation packages.

5.3 Output window

The Output window is separated as well into a messages window, a stdin/stdout window and a stderr window for the following necessities:

- messages window: displays all output of the Compiler. Clicking on error messages will change the edit widget to set the cursor to the line the Compiler found an error.
- stdout window: display for terminal based applications that send messages to the standard output of the computer. Note that Terminal applications are now started in an external console window.
- stderr window: displays all the error messages your program produces. This is useful for testing purposes. The Output window is programmed to show up each time an external process is called, such as make or a terminal application.

5.4 Menubar Commands

5.4.1 File Management and Printing

This section covers the functions that KDevelop provides about files; accessed via the File-menu in the menubar or by the according icons in the toolbar:

- **New Ctrl+N** Opens the "New File" dialog, allowing to create a new file. The file can be created using different templates and the filename has to be given as well as the path where the file will be created.
- **Open Ctrl+O** displays the "Open File..." dialog and lets you choose a file to be opened.

- **Close Ctrl+W** Closes the file in the top editing window
- **Save Ctrl+S** Saves the file opened in the top editing window. If the file has not been saved yet, the "Save File As..." dialog will be opened to let you choose a path and filename for the file to be saved.
- **Save As...** Opens the "Save File As..." dialog to let you save the current file under a new filename
- **Save All** Saves all changed files
- **Print... Ctrl+P** Opens the "Print File" dialog where you can set various options for printing either using a2ps, enscript or lpr
- **Quit Ctrl+Q** Exits KDevelop. If files are changed, you will be asked if you want to save these files.

5.4.2 Editing Files

Here, the "Edit" menu and the according icons in the toolbar are covered which provide editing files. Also the editing functions are available via a context-menu in the editor.

- **Undo Ctrl+Z** Reverts the last editing operation
- **Redo Ctrl+Z** lets you do the last undo step again
- **Cut Ctrl+X** cuts out a selection and copies it to the system clipboard.
- **Copy Ctrl+C** copies a selection to the system clipboard. This also counts for selections made in the documentation browser.
- **Paste Ctrl+V** inserts the clipboard contents at the current cursor position.
- **Indent Ctrl+I** Indent moves a selection to the right
- **UnIndent Ctrl+U** UnIndent moves a selection to the left
- **Insert File...Ctrl+Insert** Lets you select a file and inserts its contents at the current cursor position
- **Search... Ctrl+F**Opens the Search dialog that looks for an expression in the current file. For a search across several files, "Search in files..." should be used.
- **Repeat Search F3** Repeats the last search for an expression. This also counts for searches that were made across the documentation, where more than one hit was found in the same page. The next hit will be displayed by choosing "Repeat Search" or F3 and signed as marked.
- **Replace... Ctrl+R** Opens the "Search and Replace" dialog that allows the search for an expression and replaces the found text with a new expression.
- **Search in files... Ctrl+Alt+F**Displays the "Search in files..." dialog, that handles `grep` on whole directories with wildcards. Search results are displayed with a list of the filenames, line and expression. Choosing an entry will open the file and the cursor is set to the line of the search result
- **Select All Ctrl+A** Selects the whole text of the file currently opened in the top editing window

- **Deselect All** Deselects the whole text of the current file. This is often used for multiple selections so you don't have to deselect each one separately
- **Invert Selection** Inverts the selection, meaning that a selection becomes deselected and text which is not selected becomes selected.

5.4.3 View Configuration

The View-Menu covers closing and opening functions for windows and commands to enable/disable the tool- and statusbar as well as to jump errors in the code.

- **Goto Line... Ctrl+G**Opens a "Go to Line..." dialog that lets you insert a linenumber to show in the actual file. The last linenumber is remembered and marked, so you can either jump to that line again or just enter a new linenumber you want to see.
- **Next Error F4**Jumps to the next error KDevelop detects from the output. The output message of Make or other tools should give you a descriptive help what the problem is, so you can correct the error.
- **Previous Error Shift+F4** Jumps to the previous error that was reported.
- **Dialogeditor Ctrl+D** Switches to the dialog editor
- **Tree-View Shift+T**en/disables the Tree window on the left side of the main view containing the Class Browser, LFV, RFV and DOC-tree.
- **Output-View Shift+O**en/disables the Output window on the bottom of the main view.
- **Toolbar**en/disables the toolbar.
- **Browser-Toolbar**en/disables the browser toolbar, containing the back, forward and search buttons for the browser
- **Statusbar Ctrl+B**en/disables the statusbar.
- **Refresh** rescans all files to rebuild the Class Browser. The scanning progress is shown in the statusbar progress display.

5.4.4 Creating and Maintaining Projects

This section describes functions that are available in the "Project"-menu and are supposed to cover the creation and maintaining of projects

- **New...** Starts the KAppWizard and allows you to create a new project by choosing application type, name, version and other options.
- **Open** Shows the Open Project dialog, where you can choose a KDevelop project file to be opened. After selection, the project will be loaded.
- **Open recent project...** contains a submenu with the last 5 opened projects. You can open a project more easily using the recent project menu.
- **Close**Closes the current project. This is mostly done automatically when you quit KDevelop, invoke the KAppWizard or open another project.

- **New Class...**Starts the class generator to create a new class that will be added to the current project. The class generator lets you specify classname, inheritance and filenames for the new class.
- **Add existing File(s)** Opens a selection dialog where you can choose the files that you want to add to the current project and the path where they will be copied to. If you choose a destination that was currently not included in the project, e.g. creating a new subdirectory, KDevelop will copy the files to add to the new subdirectory and create a new Makefile.am. After the copy process, your project will be rebuild by automake and autoconf to include the new subdirectory into the make-process.
- **Add new translation file** Opens a language selection dialog that lets you choose the translation file language that will be added to the project. This is mostly used by translators who can easily add their language to the project and work on the translation without having to care about the Makefile.am's
- **File Properties Shift+F7** Opens the File Properties dialog showing a copy of the LFV and the project file options. Here, the properties for your files have to be set like installation path and file type.
- **Make messages and merge** This is intended to create the messages file for your project. The message file is the one containing all strings that are set up in the sources for internationalization and is used by translators to create the .po file for their target language.
- **Make API-Doc** Calls KDoc on your header files and generates a HTML output with the documentation of your project classes.
- **Make User-Manual** runs KSgml2Html on your manual SGML file, creating a HTML user manual. If KSgml2Html is not installed, sgml2html will be used to do this.
- **Make Distribution:**
  - **tar.gz** creates a distribution file of your current project in the project directory that is ready to ship to the end-users containing the sources of your project. The end-user has to `tar zxvf yourproject.tar.gz` to unpack it to a source directory and use the standard commands `./configure, make` and `make install`.
- **Options F7** Opens the Project Options dialog that lets you change various settings for your project. This could be a new version number or the compiler options to set to -02 for a release with optimization.

5.4.5 Building Projects

This section describes the Build menu that covers all actions to be done with make or actions like rebuilding the project's online help or API documentation.

- **Compile file... Ctrl+F8**Active, if the top editing window is the C++ Window. This only compiles the current implementation file and lets you save time in case you suspect errors.
- **Make F8** Invokes the make-command to your project and builds the target.
- **Rebuild All** Rebuilds all object files and the target of you project.
- **Clean Rebuild All** Cleans the project directory from all make-created files and rebuilds the target.

- **Stop Build F10** Stops the current process.
- **Execute F9** Executes your target after building the program with make.
- **Execute with Arguments Alt+F9** Executes your target with arguments. First, a dialog appears to let you specify the execution arguments (which are saved in the project for the next session), then your application uses the entered commandline. Mind that your application program is started from the project directory directly
- **Debug...** Opens KDbg in the Tools-Window to debug your application. In this case, KDbg automatically opens the file containing the main() function and executes your application.
- **Distclean** Removes all files that are generated by the project like object files etc. Distclean has to be done before distributing your project, so the distribution file doesn't contain any platform-specific files like those that were generated by your Compiler.
- **Autoconf and automake** Calls Make on the file Makefile.dist, located in your main project directory. Makefile.dist contains all commands for automake, autoconf etc. to build your project. If you added files manually or changed macros yourself, you should run Autoconf afterwards, followed by Configure to recreate all Makefiles.
- **Configure** Executes the configure-script generated by autoconf. If this command cannot be executed, run Autoconf and then Configure again.

5.4.6 Calling Tools

The Tools-menu contains by default the entries for the following programs if installed: KDbg, KIconEdit and KTranslator. These are checked by the KDevelop installation program and inserted in the given order into the menu. Invoking a tool will open the "Tools"-window and start the selected program inside this window. The Tools menu can be edited by the Tools-entry in the Options-menu; see 12.1 (Configuring the "<idx/Tools/" Menu).

5.4.7 Changing KDevelop's settings

The Options-menu contains all entries for invoking configuration dialogs with which you can change KDevelop's default settings. Major settings like those for the editor or printing have their own entries; general settings of KDevelop's behavior can be made with the 12.3 (KDevelop Setup) dialog.

- **Editor...** Allows the configuration of the editor's behavior like word-breaking, selections etc.
- **Editor Colors...** Here, you can set the color-configuration of the editor like e.g. the background.
- **Editor Defaults...** This sets the default display like font and fontsize for the editor
- **Syntax-Highlighting** This dialog lets you configure the fonts and colors for the highlighting of several programming languages, including HTML.
- **Documentation Browser** In this tab-dialog, you can set the fonts, sizes and colors for the internal Helpbrowser
- **Configure Printer...** Contains the entries for the printer configuration dialogs according to the used printing program, a2ps or enscript.
  - **a2ps** Configures the printer for use with a2ps. See Printing for more information.

  - **enscript** Configures the printer for use with enscript. See Printing for more information.

- **Tools...** Opens the Configure Tools dialog. Here, you can configure the "Tools" menu by adding or removing programs that will be started in the Tools-window.
- **KDevelop Setup** Opens the KDevelop Setup dialog. The first tabulator configures general settings, followed by the key-configuration and the Documentation settings. The Documentation settings also include the generation of a new set of HTML-library documentation and the rebuilding of the search index for the Helpbrowser.

5.4.8 The Window Menu

The Window menu contains a list with all currently open files. This allows a quick switch to another file you're currently working on.

5.4.9 Managing Bookmarks

The Bookmarks menu is intended for adding and removing bookmarks you want to set in the current editing file. As KDevelop uses two editing windows, each one configures it's bookmarks separately.

- **Set Bookmark** Opens a context-menu with up to nine configurable bookmarks. This allows setting a bookmark to a certain entry in the bookmarks menu by logical reasons.
- **Add Bookmark Ctrl+Alt+A** Adds the line of the current cursor position to the bookmarks menu as a bookmark. Mind that this could overwrite a bookmark set with the Set Bookmark-option. If the browser is opened, a bookmark will be added to the Browser-Window bookmarks menu.
- **Clear Bookmarks Ctrl+Alt+C** Clears the bookmark-entries for the top editing window or the browser, e.g. if the Header-Window is visible and you call Clear Bookmarks, the entries for the Header-Window are deleted.
- **Header-Window** Contains the bookmark-list for the Header-Window. Selecting a bookmark will set the cursor to the selected bookmarks' line. Mind that bookmarks are only assigned to the file they are set to, so if you change to another file, the bookmarks are not deleted but choosing a bookmark won't change to the file they are assigned to as well.
- **C/C++-Window** Contains the bookmark-list for the C/C++-Window. The preferences are the same as for the Header-Window.
- **Browser Window** Contains the browser bookmarks. Selecting a bookmark will open the browser with the selected page.

5.4.10 Online Help

The Help-menu contains entries for navigating in the Helpbrowser as well as entries for library and online-documentation for the most recently used cases. Accessing additional online-documentation can be achieved with the DOC-tree in the treeview automatically opened (if Autoswitch is enabled) when changing to the Documentation Browser window.

- **Back Alt+ Left Arrow** Opens the page opened before the actual one.

- **Forward Alt+ Right Arrow** Opens the next page of the browser-history, available after a "Back"-action.
- **Search Marked Text F2** Scans the search-index for text currently marked in the editing windows or the browser-window. After the search, a result page is shown which lets you select the help-page to switch to. After choosing a page, the browser will highlight the found entry for the expression the search was invoked for. With the F3 key the next search result on the same page will be displayed if more than one hit was reported for the search on one page.
- **Search for Help on...**Opens the Search for Help on... dialog that lets you search for a specific expression you want help on.
- **User Manual F1** Opens the User's Manual to KDevelop index page so you have access to this manual.
- **Programming Handbook** Opens the *KDevelop Programming Handbook* index page to access the programming manuals
- **Tip of the Day** Opens the Tip of the Day dialog to inform you about KDevelop's features.
- **KDevelop Homepage** Opens the KDevelop Homepage in the browser window if Internet access is provided.
- **Bug Report...** Opens the KDevelop Bug Report dialog, where you can send a bug-report directly to the KDevelop Team by email. See 13.1 (Bug Reporting)
- **C/C++ Reference** Displays the language reference index page. If the reference is not installed, an error page shows you how to get the reference and how to install it correctly.
- **Qt-Library** Changes to the index page of the Qt-library documentation provided with your copy of the Qt-library.
- **KDE-Core-Library** Opens the class-index file of the KDE-Core library documentation.
- **KDE-GUI-Library** As above for the GUI-library
- **KDE-KFile-Library** As above for the KFile-library
- **KDE-HTML-Library** As above for the HTML-library
- **Project-API-Doc** Changes to the project's class-documentation index file.
- **Project-User-Manual** Opens the User manual's index file of your current project. This can be used to review the HTML-output generated by KSgml2Html
- **About KDevelop...** Shows the aboutbox of KDevelop containing the used version number and the authors' names and email-addresses as well as reference to the licensing of KDevelop.

5.5 Toolbar Items

KDevelop provides quick access to a various set of commands by it's toolbars. These are the standard and the browser toolbar; in dialog editor mode only the standard toolbar is visible. Both can be en-/disabled by the according menu entries in the "View" menu; also dragged out of the main window and replaced on each side of the working area.

5.5.1 The Standard Toolbar

The standard toolbar provides quick access to the most recently used functions for file processing and editing as well as building your application. The buttons execute the following commands from left to right:

- Open Project - shows the open project dialog
- Open File - shows the open file dialog and contains a delayed popup to fast select the current project header and source files.
- Save File - saves the currently opened file to disk
- Print File - opens the printer dialog
- (separator)
- Undo - reverts the last action
- Redo - executes a reverted action again
- Cut - cuts out the current selection
- Copy - copies the current selection to the system clipboard
- Paste - pastes the current clipboard contents to the actual cursor position
- (separator)
- Compile File - compiles the file currently visible in the source file window. In Dialogeditor mode replaced by the Generate Files button.
- Make - invokes make on the project
- Rebuild All - rebuilds the project
- Debug - opens KDbg with the application binary for debugging in the Tools- window
- Execute - runs the application target binary
- Stop - cancels the current process
- (separator)
- Dialogeditor - switches to Dialogeditor mode. There, the button is replaced by a Sourcecode editor button
- Tree-View - en-/disables the treeview and works as a toggle button to display the current state
- Output-View - en-/disables the output view and works as a toggle button to display the current state
- (separator)
- What's this..? help button - changes the cursor to a question arrow and lets you get information about GUI components of KDevelop

5.5.2 The Browser Toolbar

The browser toolbar is another toolbar that provides a lot of useful commands for browsing files and the documentation. The toolbar contains:

- the Class combo box - lets you select a class of the current project to browse to
- the Method combo box - lets you select a method of the current class and browses to the implementation of the method
- the Class-assistant button - on a single click will bring you to the declaration of the method currently selected. Contains a delayed popup to access the Classtools for adding Classes, Methods and Attributes as well as the browsing commands.
- Back - browses back in the documentation browser history; contains a delayed popup to select a certain page of the back history.
- Forward - browses forward in the documentation browser history; contains a delayed popup to select a certain page of the forward history.
- Stop - stops the browser from loading a documentation file request
- Reload - reloads the currently displayed page
- Home - opens the KDevelop User Manual index page in the browser
- Search Marked Text - searches the documentation index for the selected text; works with the browser and the editor windows
- Search for Help on... - opens the the Search for Help on.. dialog to let you enter a keyword to search in the documentation

5.6 Keyboard Shortcuts

This section handles the predefined as well as the standard values for configurable keyboard commands used in the KDevelop IDE. You should see section 12.4 (Changing Keyboard Shortcuts) for a detailed explanation how to change assigned values to commands.

5.6.1 Shortcuts for Text Processing

Cursor Movements

one letter to the left	Left Arrow
one letter to the right	Right Arrow
one word to the left	CTRL+Left Arrow
one word to the right	CTRL+Right Arrow
one line upwards	Up Arrow
one line downwards	Down Arrow
to the beginning of the line	POS 1
to the end of the line	END
one page up	PageUp
one page down	PageDown
to the beginning of the current file	CTRL+PageUp
to the end of the current file	CTRL+PageDown

Text Selections

one letter to the left	SHIFT+Left Arrow
one letter to the right	SHIFT+Right Arrow
one word to the left	CTRL+SHIFT+Left Arrow
one word to the right	CTRL+SHIFT+Right Arrow
one line upwards	CTRL+Up
one line downwards	CTRL+Down
to the beginning of the current line	CTRL+POS 1
to the end of the current line	CTRL+END
one page up	SHIFT+PageUp
one page down	SHIFT+PageDown
to the beginning of the current file	CTRL+SHIFT+PageUp
to the end of the current file	CTRL+SHIFT+PageDown

Inserting and Copying Text, Tabulators

en/disable insertmode	INS
copy selection to the clipboard	CTRL+C, CTRL+INS
insert text from the clipboard	CTRL+V, SHIFT+INS
delete current line	CTRL+K
insert line after current line	END, then Enter
insert line before current line	POS 1, then Enter
undo editing step	CTRL+Z
redo an undo step	CTRL+Y
tabulator	TAB

Deleting Text

delete letter left of cursor position	Backspace
delete letter right of cursor position	Delete
delete selected text	Select text, then Backspace or Delete

Searching Text within the Editor

open Goto Line... dialog	CTRL+G
open Find Text dialog	CTRL+F
repeat last search	F3
open Search and Replace dialog	CTRL+R
open Search in Files dialog (grep)	CTRL+ALT+F
search marked Text with Grep	SHIFT+F2
locate next error	F4
locate previous error	SHIFT+F4

Searching Text with the Documentation Browser

search selected editor text in documentation	F2
search selected browser text in documentation	F2

display next search hit on the same page	F3
search selected browser text within project	SHIFT+F2

Browser Shortcuts

previous page	ALT+ Left Arrow
next page	ALT+ Right Arrow

Managing Bookmarks

add bookmark	CTRL+ALT+A
clear bookmark list	CTRL+ALT+C

5.6.2 Shortcuts for Toolbar Symbols

Symbol "Open File"	CTRL+O
Symbol "Save File"	CTRL+S
Symbol "Print File"	CTRL+P
Symbol "Undo"	CTRL+Z
Symbol "Redo"	CTRL+Y
Symbol "Cut"	CTRL+X
Symbol "Copy"	CTRL+C
Symbol "Paste"	CTRL+V
Symbol "Compile File"	CTRL+F8
Symbol "Make"	F8
Symbol "Execute"	F9
Symbol "Execute with Arguments"	ALT+F9
Symbol "Dialog Editor"	CTRL+D
Symbol "Back"	ALT+Left Arrow, if browser opened
Symbol "Forward"	ALT+Right Arrow, if browser opened
Symbol "Search Marked Text"	F2

5.6.3 Window Management

To switch to a certain window, press ALT plus the underlined letter in the window title, e.g. Tools would be Alt+T

5.6.4 Shortcuts to Compilation Processes

Compile current sourcefile	CTRL+F8
Build current project target	F8
Execute target after Build process	CTRL+F9
Execute target with Arguments	ALT+F9
Stop the current process	F10

Chapter 6

The Help System

Most of KDevelop's strength is accumulated in the Help System. It is also to be understood as an example of how to extend your own KDE application with a complete set of help functionality, and is, in part, already realized in the template applications, so KDE/Qt application frameworks generated with the KAppWizard do already contain the basic functionality for statusbar help and documentation, that only has to be extended by the programmer. This section therefore introduces into the usage of the general help provided within the IDE as well as to show the usage of the Helpbrowser, which, by intelligent use, will make it very easy to get the information you need for development.

6.1 The "What's this ?"-Button and Quickhelp

Looking at KDevelop's top toolbar, you will see the "What's this?" button on the right end. By selecting the button, the cursor will change to a pointer with a question mark on the right, as is the same as the button in the toolbar. Now, you can select any visible part of the KDevelop user interface and click on it. This will result in displaying a help-window giving you a short description of the function this part gives you or what it can do for you. Included are the main view, the tree-views and the toolbars, for which every button help is provided. After another mouse click or a keyboard input, the What's this? help window disappears and your cursor is set to the last active position. Mind, that, if you place the mouse pointer over a button, it gets raised and after a short time, you will get a Quick-Tip-window, describing the function the button represents in the menubar; this will disappear if you move the mouse pointer away.

For user interaction dialogs, Quickhelp provides you help windows with a short description of the selected item you want help for. These are accessed by a right-button mouse click to the item, a context-menu pops up that allows the selection of "Quick-Help". Selecting this will pop up the help window. This is always useful for dialogs where you can't see the action's purpose for as long as you are not familiar with KDevelop. Mind that most dialogs offer a Help-button which will show you the detailed context-help for the dialog's options within the provided online-manual in the help-browser.

6.2 The Statusbar Help

The statusbar of KDevelop offers you many functions that inform you about the current activity state as well as "status messages" offering a short description for commands.

6.2.1 Statusbar Entries

The Statusbar contains:

1. a general message field, most left. This is used for help messages and displays the current activity.
2. the progressbar, indicating the progress for actions that require a comparably long time to be finished, such as saving files and the scan-progress of the Class Browser. The progressbar only appears during those processes are executed.
3. an Insert/Overwrite indicator. It shows the editor mode for insert actions by INS for Insert mode and OVR for Overwrite mode. The mode can be changed by the INS-key on the keyboard.
4. a line counter, displaying the current line the cursor is placed.
5. a column counter, indicating the current column position of the cursor in a line.

6.2.2 Help Messages

The statusbar informs you about the action of menu entries if you select a menu within the menubar and select an action without executing it. Further, if you press on a toolbar icon, but without releasing the mousebutton, the help message for the button is displayed in the same way as for the menu entries. You can prevent the execution of the selected icon by moving the mousepointer away from the icon, still holding the button pressed. If the cursor is away, release the mousebutton.

For actions that are currently executed, KDevelop displays the action. This counts for processes running in the background such as saving files as well as for dialogs. If the process has exited such as a make-invocation, the statusbar display changes back to "Ready" state.

Also, when using the documentation browser, the statusbar displays link urls for as long as the cursor is placed over an URL link in the browser window. Therefore, you can easily find out if the file is a local or a remote file only accessed by network connection.

The caption of the KDevelop Project Editor shows you the currently opened filename in the top window. This would be a HTML file for the documentation browser or a text file for the editing windows. Further, the Project name is displayed, so you have constant control where you are and what you're currently working at.

6.3 Configuring the HTML Browser

The browser included in KDevelop is completely HTML-based, therefore you can specify the usual options like background and the like. You can set all preferences by the "Documentation Browser"-entry of the "Options"-menu. The configuration dialog shows you two tabulators; the first for setting the font preferences, the second for color selection.

6.3.1 Font Preferences

The first option for the font display is the size. Available are small, medium and large. The best display for normal use is set to small by default. For the font selection you have to specify a standard font for normal text as it appears in HTML documents; the fixed font is the one used for displaying e.g. code within the HTML file.

6.3.2 Color Preferences

The color preferences dialog allows configuration of background, normal text, URL Link and followed link color. The colored buttons on the right are displaying the current settings; selecting a button will result in opening the "Select Color" dialog. There, you can specify the color either by choosing a System Color, a custom color, select the color by the multi-colored window or by setting the values directly. The middle contains a preview for the color selected.

Further, you can specify if the browser should underline links to easier detect them visually and if you like to use your own colors always independent of the page's preset ones. "Apply" will execute all changes, while "OK" will apply changes and close the dialog. "Cancel" leaves all settings untouched and exits the configuration.

6.4 Using the Documentation Browser

The documentation browser allows quick and easy access to all manuals and documentations that are provided with KDevelop or generated automatically, including online-documentation for the KDE-libraries and your project documentation. Plus, the Documentation Tree in the tree-view lets you customize a special "Others" folder that contains all individually added documentation.

6.4.1 Requirements

To make use of all features of the Helpbrowser, you should have KDoc and glimpse installed. KDoc will generate all kind of online class-documentation for the KDE-libraries during the setup process, but can also invoked by option in the 12 (KDevelop Setup). For the generation of an API (Application Programming Interface) Documentation for your project's classes, KDoc is also used and called by the Project-menu, entry "Make API-Doc". This will process all current header files of the project plus the addition of a cross-reference to the Qt and KDE-libraries if those are available in the Helpbrowser. For the generation of the project handbooks, you should have KSgml2Html (provided with the KDE-SDK) and, as a minimum, SGML-tools installed on your system. To extend and change your project's online-documentation, you have to edit the documentation file for your project by selecting it in the RFV. After saving your changes, call "Make User-Manual" from the Project-menu. In case SGMLtools detect formatting errors, those will be displayed in the output window allow you to find the error line directly. The program "glimpse" is used to create a personal search-index for your documentation automatically. The index is generally build during the installation process, but can also be build with the 12 (KDevelop Setup) dialog. For a description of the provided search functions see 6.4.4 (Using the Searchindex).

6.4.2 Provided Documentation

The KDevelop IDE comes with two sets of online-documentation, which can be accessed either via the Help-menu or by the Doc-tree in the Tree-View in the KDevelop folder. The first book provided is this online-handbook containing all information you need towards installation, configuration, available functionality and introduction to application development. The second book is *The KDevelop Programming Handbook* in it's first edition. The programming handbook covers most questions related to the creation and extension of projects that work with KDevelop. By tutorials the user gets an introduction to the rich facilities that applications created with the Qt- and/or KDE-libraries have to offer as well as given a guideline for ensuring KDE-compliance, which covers the same instructions given on the Internet site at `<http://developer.kde.org>`. However, the programming handbook cannot replace any additional documentation available in printed or electronic form about the C++ programming language as well as about the usage of the Qt-library in certain terms.

The C/C++-Reference used with KDevelop is currently only available on the KDevelop homepage at `<http://www.kdevelop.org>`. On distributions the reference may be included, see your distribution installation program index for more information. It can be installed easily by downloading and copying the sourcefile to the main KDE-directory ($KDEDIR). There, you have to untar it as root with `tar zxvf c_c++_reference.tar.gz`, the reference will then be copied the documentation directory of KDevelop. For uninstalling the reference documentation you just have to delete the "reference" folder under ($KDEDIR)/share/doc/HTML/default/kdevelop/reference.

The Qt/KDE-libraries documentation folder allows direct access to the HTML-online documentation of your copy of the Qt-library. The path to the library documentation is usually automatically detected by the KDevelop installation program, but can be set manually in the 12 (KDevelop Setup) dialog. Also, all available documentation for the KDE-libraries are listed in order of the library name, so if you would use classes of a certain library it is easy to determine the library type to be added to the Linker settings in the project. Mind that the whole KDE-Library documentation is only accessible when generated by KDoc- so this program, included in the KDE-SDK, has to be installed before the documentation can be generated. As the installation program of KDevelop does this automatically, it should be installed before running the KDevelop Setup. If this is not the case and the documentation cannot be build, you are able to create it afterwards at any time with the 12 (KDevelop Setup) dialog.

The documentation tree also contains the "Others" folder which is intended to contain all personally customized documentation as described below. Finally, the Doc-Tree allows access to the API and the Manual for your current project.

6.4.3 Adding Documentation to the Helpbrowser

To customize your Helpbrowser, open the Tree-View and select the "DOC" Tabulator. You will see an opened tree containing four folders. The "Others" folder is, by default, empty. On a right mousebutton click over this folder, a context menu opens that has an entry "Add Entry". Select this to open the Add Entry dialog where you have to enter two values: the upper one for the name that will be displayed for the documentation within the Doc-tree, and, below that, a file-entry line. Here, you have to enter the path and filename to the start page that will be opened by the selection of the entry later. You can enter the path and filename directly or by selection of the pushbutton on the right to open a file-chooser dialog. This allows a quick browsing on your system to the path for your start page. Mind that only HTML-documentation can be selected, so only HTML files are allowed to choose. Selecting OK will add the entry to the Doc-tree and is available directly.

6.4.4 Using the Searchindex

KDevelop includes a set of functions to help you search information within HTML documentation. To use these features, your system needs to have the program "glimpse", a free database generator, installed, which creates the search index and executes searches within the documentation. To set up the index, see 12 (KDevelop Setup). The index is also automatically generated during the installation program.

The search functions are available through several ways which you can choose from:

1. in the editor, select the text you want to have help on or place the cursor in the word that you

want to look for. Then press the right mouse button and select look up: "expression" This can also be done by selecting "Search Marked Text" from the Help-menu, the shortcut key F2 or by selecting the Search icon from the toolbar.

2. in the Documentation browser select the text you want to look up and press the right mouse button, select look up:"expression" or select "Search Marked Text" or the search icon as above.
3. if you want to search for a specific keyword, select "Search for Help on..." in the Help-menu or select the "Search for Help on..." icon from the toolbar, opening a search dialog where you can enter the expression you want to get information for.

Direct access to specific documentation is given by selecting the according icon in the "DOC" -tree-view or by the Help-menu entries.

The Helpbrowser also offers support for grep to search for a selection within your current project, e.g you've opened the documentation page for the `KTMainWindow` class (KDEUI library) and you want to know where in your code `KTMainWindow` appears. Mark `KTMainWindow` and either select "Grep: KTMainWindow" or press Shift+F2. The "Search in Files" dialog opens with the results about your grep search directly. Then you can choose an result line to jump to the according sourcecode.

Chapter 7

Working with the Editor

An important part of the integrated development environment is the editor. You're using it for:

- creating, opening and saving source and project files
- editing source and project files
- writing your SGML documentation
- printing your project files

Generally, the editor itself doesn't do much difference in comparison to other editors; especially as it is a build-in version of the popular KWrite, also known as the "Extended Editor". If you're familiar with that, you won't have any problems using it for managing your projects. In addition to normal editors, KDevelop contains a new printing system, that is far more extended for use with sourcecodes- and you can choose between the printing program you want to use.

This section gives you more information about how to handle your project files and to make you more familiar with the editor's functionality. A lot of Unix-hardliners would prefer using Emacs or XEmacs- they're fine with that. But it isn't necessary for easy programming, and especially for beginners, to work themselves into powerful editors which shall, in the result, only allow you to enter your code.

7.1 Managing Project Files

The following sections describe how to create, save, open and close your project files you need to edit- mind that this doesn't include translation files or pixmaps. Those are recognized automatically and the according editing program like KTranslator opens those files for you.

To give you an easy access to your files, KDevelop contains two easy-to-handle tree-views, similar to a filemanager, that recognize if you selected a file, and opens it in the according editor window. The main view contains two windows that are used independently, but connected via the Project Editor, so you can do all menubar actions the same way. The intention behind this is that it allows you to handle two windows at the same time, although only one is visible. The C/C++ window thereby takes another task, which allows you to easily compile your sources one by one, so you can check the implementations without having to run a complete build-process over your project.

This can be done when having the C/C++ window on top. Select "Compile File" from the Build-menu or press the according button in the toolbar. Your file gets saved and compiled, and you can control any errors by the Compiler output in the output window.

Within the IDE you can open as many text files as you like. All opened files are listed in the Window-menu, so you can switch between them by selecting the according filename from the menu. Further, the actual opened file is displayed with its name in the window frame of KDevelop.

7.1.1 Creating and Saving Files

To create a new file, choose "New" from the File-menu. This opens the "New File" dialog, where you can specify the filename and type. Additionally, you have to set the destination directory as well if the file is added to the project or not. Finally, you can also use your header template for the project in case you want to add a new sourcefile without generating a new class by the Class-Generator.

After the file has been created, you can edit the new file as usual; if you have to switch between different files, you can always turn back to the file either via the file-trees or by the Window-menu.

For saving any changes, KDevelop offers a whole variety of options. The standard way would be to save the file by selecting "Save" or "Save As" from the File-menu or by the Save-button on the toolbar. For saving all changed files at once, you could as well select "Save all" from the File-menu.

Now, when working on a project, it is very annoying if something critical happens while you have done a lot of changes on your files; sometimes you will bite yourself that you have forgotten to save the changes. KDevelop takes care of this by offering "Autosaving", which is by default enabled and set to save all files in intervals of 5 minutes. To select another saving period or to disable this, see 12.3 (KDevelop Setup), where the setup options are described.

Additionally, KDevelop takes care for all changes if you open another project or exit KDevelop. You will be asked for saving the changed files, where you also have the option to specify which one to save and which you want to stay unsaved. Further, when invoking any build-processes, your files get saved automatically, so you won't ever wonder why your application doesn't run the way you expected it to do after your changes to the source files. The only exception is the "Compile File" command, which only saves the currently opened source file visible in the source editor window.

7.1.2 Opening and Closing Files

To open a source file, you have a lot of options as well. One way could be, as the standard for editors, to select "Open" from the File-menu. You will be presented an "Open File" dialog, which allows you to choose the file you want to edit. Another, probably more frequently used way is to select the file from the LFV, the Logical File Viewer, or the RFV, the Real File Viewer (see 5 ()). The advantage of the file trees is that they provide a quick visual access to your files, especially the LFV, which only displays your project files by their type, collected into folders. You can as well configure the LFV towards sorting your files in another manner by a right mouse click on the tree. Select "New Group" from the popup-menu, and you can configure a new group, or select "Edit Group" to install new file filters by specifying the group's file extensions, separated by commas.

The standard file groups for a project generated by KAppWizard are the Headers, Sources, GNU and Others. Additionally, after adding a translation file, KDevelop adds a folder "Translations", containing your *.po files.

To close files, select the file to close from the Window-menu, which loads the opened file into the front editor. Then select "Close" from the File-menu. If your file has been changed, you will be asked to save it. When closing a project, all currently opened files will be checked for changes and you will be asked for saving as well.

7.2 Navigating within Files

The following gives you a general guideline how to locate certain positions within your files for a quicker access.

» How to find a certain line in a file

1. Select "Goto Line" in the "View"-menu or press CTRL+G. The dialogfield "Goto Line" appears.
2. Insert the linenumber you want to go to.
3. Press OK.

» How to set a bookmark

1. Set the cursor to the line you want to access via a bookmark
2. Select "Set Bookmark" from the "Bookmarks"-menu.
3. A popup-menu opens that allows you to select the bookmark number you want to set for the new bookmark.
4. Select the bookmark number.

Another way to set bookmarks would be to select "Add Bookmark" from the "Bookmarks"-menu. This sets a bookmark to the current line and appends it to the bookmarks-list. The documentation browser also offers setting a bookmark to the current page by a context menu entry "Add Bookmark".

» How to delete bookmarks

The bookmarks are set for each editing window separately- mind that your bookmarks aren't connected to a certain file you set the bookmark for. To delete all bookmarks, select "Delete Bookmarks" from the "Bookmarks"-menu. This deletes the bookmarks for the window actually on top, either the Header window, the C++ window or the browser window.

» How to go to a bookmarked line

1. Select the "Bookmarks"-menu and open the popup menu containing the bookmarks for the window containing the bookmark you want to view; either the C++-Window or the Header-Window entry.
2. Select the bookmarked line.

Selecting a bookmark for the browser window will open the browser and loads the page.

7.3 Working with Keyboard Shortcuts

For using the editor, you should make yourself comfortable with some keyboard shortcuts that make it easier to position the cursor and edit the file. The complete shortcut reference is also listed in section 5 (Overview).

`one letter to the left`	`Left Arrow`
`one letter to the right`	`Right Arrow`
`one word to the left`	`CTRL+Left Arrow`
`one word to the right`	`CTRL+Right Arrow`
`one line upwards`	`Up Arrow`
`one line downwards`	`Down Arrow`
`to the beginning of the line`	`POS 1`
`to the end of the line`	`END`
`one page up`	`PageUp`
`one page down`	`PageDown`
`to the beginning of the current file`	`CTRL+PageUp`
`to the end of the current file`	`CTRL+PageDown`
`one letter to the left`	`SHIFT+Left Arrow`
`one letter to the right`	`SHIFT+Right Arrow`
`one word to the left`	`CTRL+SHIFT+Left Arrow`
`one word to the right`	`CTRL+SHIFT+Right Arrow`
`one line upwards`	`CTRL+Up`
`one line downwards`	`CTRL+Down`
`to the beginning of the current line`	`CTRL+POS 1`
`to the end of the current line`	`CTRL+END`
`one page up`	`SHIFT+PageUp`
`one page down`	`SHIFT+PageDown`
`to the beginning of the current file`	`CTRL+SHIFT+PageUp`
`to the end of the current file`	`CTRL+SHIFT+PageDown`
`en/disable insertmode`	`INS`
`copy selection to the clipboard`	`CTRL+C, CTRL+INS`
`insert text from the clipboard`	`CTRL+V, SHIFT+INS`
`delete current line`	`CTRL+K`
`insert line after current line`	`END, then Enter`
`insert line before current line`	`POS 1, then Enter`
`undo editing step`	`CTRL+Z`
`redo an undo step`	`CTRL+Y`
`tabulator`	`TAB`
`delete letter left of cursor position`	`Backspace`
`delete letter right of cursor position`	`Delete`
`delete selected text`	`Select text, then Backspace`

7.4 Edit Windows Settings

The editor inside KDevelop can be configured towards special editing needs with a global effect on all editing windows. Thereby you can set the color modes, highlighting colors (also configurable in reference to the programming language of the file) and automatic text settings like tab-with and selection modes. The following describes how to set these options by the configuration dialogs provided in the "Options"-menu.

7.4.1 General Settings

The editor's general settings can be configured with the "Editor" entry in the "Options"-menu. Select the according values and press OK after you finished with your configuration.

Edit Options

Auto Indent:

this sets the editor to place the cursor below the first literal when a new line is entered.

Backspace Indent:

this option sets the cursor below the first literal of the line above when backspace is pressed.

Word Wrap:

words are taken into the next line after the column set in "Wrap Words At:"

Replace Tabs:

tabulators in the current text are replaced with tabulators of the "Tab Width:" value

Remove Trailing Spaces:

removes trailing spaces

Wrap Cursor:

sets the cursor to the end of the last line when backspace enters the beginning of a line

Auto Brackets:

creates a closing bracket in front of the cursor when a bracket (any kind) is opened.

Select Options

Persistent Selections:

selections made stay marked after setting the cursor to a different place

Multiple Selections:

allows multiple independent selections within the text

Vertical Selections:

allows vertical selections of text

Delete On Input:

deletes a selection when writing in the selection.

Toggle Old:

allows only one selection. A selection made gets deselected when another selection is made.

Wrap Words At:

sets the maximum columns that a line can have. The word that contains a letter that reaches over this value will be automatically broken into the next line.

7.4.2 Colors

For changing the editor's general appearance, you can define a set of colors that the editor uses by selecting "Editor Colors" from the "Options"-menu. You can configure colors for:

- **background:** the editor's background
- **text background:** the background of displayed text

- **selected:** the color of selected text
- **found:** the color of text found by a search via the menu "Edit"-"Search", "Repeat search" and "Replace".
- **selected + found:** the color of text selected to search for and found

7.4.3 Syntax Highlighting

The syntax-highlighting mode of KDevelop's editor can be configured by two dialogs; first you can set default colors for syntax-highlighting by the "Editor Defaults" entry in the "Options"-menu. There, you can set colors and fonts as well as the fontsize for e.g. keywords. Select the default item and set all needed options.

The second configuration dialog is accessed by the "Syntax-Highlighting" entry in the "Options"-menu. This allows you to set the file filters for the programming language, e.g. *.cpp, *.h for C++. Then select the item that you want to configure. If you wish to use the default values you've set in the "Editor Defaults" dialog, select the "default" checkboxes in the Item-style and Item-Font sections. This reads the default settings for the selected item. After pressing OK, your new values will be enabled and used by the editor.

7.5 Searching and Replacing

7.5.1 Single File Search

» How to find one or more characters in the actual editing window

1. Select "Search" from the "Edit"-menu. This opens the search dialog.
2. Specify the expression to search for in the edit field. The drop-down menu offers selecting a previous search expression.
3. Select additional options like "Whole Words only"
4. Press OK.

To repeat searching for an expression entered in the search dialog, press F3.

7.5.2 Searching over Several Files

As the search function only is referencing searches over a single file that is currently visible, you're limited to this. But often you want to search for the same expression overall your project. Therefore, KDevelop contains a `grep`-dialog, which lets you search over all files that you specify either by setting the directory to start the search from and/or by mime-type. Specifying the exact directory and the mime-type will therefore reduce the time that KDevelop needs to read your files and display the results. To start a search over several files, select "Search in files..." from the "Edit"-menu. The search dialog opens and lets you enter:

- The expression to search for (the pattern)
- The template used for the search

- The mime-type of files to search in
- The directory to start form
- If the search is recursive over all included subdirectories

By default, the grep-dialog is set to start at your project directory and works recursively over implementation and header files.

You can even extend your search pattern by using the following options:

1. **.** Match any character
2. ˆ Match the beginning of a line
3. **$** Match the end of a line
4. \< Match the beginning of a word
5. > Match the end of a word

For a repeating search, you can also use the available operators:

1. **?** The preceding item matches less than once
2. ***** The preceding item is matched zero or more times
3. + The preceding item is matched once or more times
4. { **n** } The preceding item is matched exactly *n* times
5. { **n,**} The preceding item is matched *n* or more times
6. {**,n** } The preceding item matches less than *n* times
7. {**n,m**} The preceding item matches at least *n* times but less than *m* times

Backreferences to bracketed subexpressions are also available by the notation \n.

After specifying your search, press "Search". The results are then displayed in the result-window. To jump to a file and line number, select the resultline and press Enter or double click the result. The editor will automatically open the according file and place the cursor to the line of the result. This allows a complete specification for any search action and give out exact results.

KDevelop offers also some more specialized functionality to use grep within the editors and the browser. Select the expression you want to search for in either of the windows and press SHIFT+F2 or select "grep:<your_expression>" from the right button popup menu. This will ask grep to search for the selection in your project directory's files and will show the results immediately. Switching to the result works as described above. From within the editor window, it lasts to place the cursor over a word and start searching; the word under the cursor will be the search pattern.

7.5.3 Searching within Documentation

While working on a project, you often need to have information about the parameters of member functions you want to use- most often you remember the function's name that matches your needs, but the parameters are a very hard thing to keep in mind. Therefore, and for other purposes that may occur, KDevelop contains a search functionality that combines searching expressions that appear in your files with the documentation browser. To make use of this search functionality, you should have set up the documentation browser correctly and created the search database. For invoking a search through the documentation, do the following:

1. place your cursor at the word you want to search or mark an expression
2. select "Search Marked Text" from the Help-menu or press the right mouse button to open the context menu; then select "Look Up: "expression".
3. after the search result page is displayed in the documentation browser, select the page that you think could contain the information you need.
4. the selected documentation page is displayed and your search result is marked. To display the next result within the same documentation page, press F3.

This allows you to easily find the information you need. For using the results, the documentation browser allows marking a selection and copying it to the clipboard. Then return to the file you're editing and select "Paste" from the "Edit"-menu.

For a full description on how to use the Documentation, see 6.4.4 (Using the Searchindex).

7.5.4 Replacing Text

For replacing an expression, select "Search and Replace" from the "Edit"-menu. The "Search and Replace"-dialog lets you specify the expression to be replaced as well as the replacement expression. Then press OK. The first expression which is found will be marked, so you can see where the expression is and in which context. Then you can specify by a dialog if the expression shall be replaced or not. When the search is finished by reaching the end of the current file, you will be asked if you want to start the search again from the beginning. If you're finished, select "Cancel".

7.6 Printing

As KDevelop is designed to give developers the best access to files and information to reduce development cycles, it also contains a new printing utility which makes use of two common printing programs available for Unix-Systems, *a2ps* (ASCII-to-Postscript) and *enscript*. Besides these, you can also print by directly using lpr (the lineprinter device). As using a2ps or enscript offers the most options towards printing, you should install either one of these on your system; both programs are usually shipped with distributions, so you shouldn't have any problem to get them. But before printing, you should have a look at the available configuration dialogs to prepare the output according to your needs. The following section describes how to configure KDevelop for printing files.

7.6.1 Configuring the Printer

The printing programs can both be configured by selecting "Print" from the "File"-menu; on the printing dialog, select the program by the drop-down-menu in the left upper corner. Then press the "Options" button on the right. This opens the configuration for the selected program that will be used. Another way to configure the programs are selecting "Printer Configuration..." from the "Options"-menu; then select "a2ps" or "enscript".

The a2ps Configuration Options

Printing

- **header:** adds a header frame to the page
- **filename:** if checked, the header frame will contain the filename as its text
- **login:** adds the User ID to the right upper corner of the page
- **borders:** adds frame borders to the text page
- **Date & Time:** adds the printing date and time
- **align files:** prints out files on the same page, available for two-page printing mode
- **set TAB size:** sets the TAB size for printing TABs
- **headertext:** available, if filename is deselected and allows inserting another text to the header frame
- **fontsize:** sets the fontsize for the text. The default fontsize is 9

Textprinting

- **cut lines:** cuts the line's contents if the line is too large to be printed. If deselected, lines will be broken.
- **interpret TAB, BS and FF:** interpret TAB, Backspace and FastForward characters.
- **replace non-printing character by space:** if the file contains non-printable characters, those will be replaced by space characters.
- **print non-ASCII character as ISO-Latin 1:** prints characters not included in the ASCII format in ISO-Latin 1 mode output
- **bold font:** prints the whole text in bold font mode.

Numbering

- **numbering lines:** numbers all lines top-down if enabled
- **numbering pages:** allows selecting page-numbering mode by:
  - **file single:** numbers each file's pages beginning with 1
  - **file together:** appends all following pages after the first for page numbering
- **lines per page:** sets the maximum lines per page to be printed.

The enscript Configuration Options

Header

- **Fancy Header:** adds a fancy header
- **Header Text:** enables adding a header text
  - **text:** sets the text's contents
  - **position:** sets the text's position to left, center or right
- **Login:** add the UserID to the header
  - **login:** enables adding UserID
  - **position:** sets the position for the UserID
- **Filename:** add the filename to the header
  - **Size of filename:** add filename as full or short, meaning full path or filename only
  - **Position:** sets the position for the filename
- **Hostname:** add the hostname to the header
  - **hostname:** enables adding hostname
  - **size of hostname:** sets the size for hostname
  - **Position:** sets the position for hostname

Date & Time

- **Current Date:** include the current date
  - **current date:** enables current date adding
  - **position:** sets the position for the date entry
  - **format:** sets the date format
- **Modification Date:** include the last modification date
  - **modification date:** enables modification date adding
  - **position:** sets the position for modification date entry
  - **format:** sets the date format
- **Current Time:** include current time
  - **current time:** enables current time adding
  - **AM**PM:/ use AM/PM or 24h format
  - **Position:** sets the position for the time entry
  - **Format:** sets the time format
- **Modification Time:** include the last modification time
  - **modification time:** enables adding modification time
  - **AM**PM:/ use AM/PM or 24h format
  - **Position:** sets the position for the time entry
  - **Format:** sets the time format

Layout

- **Numbering & Border:**
  - **numbering lines:** adds line numbers to the document for printing
  - **borders:** adds a border to the pages for printing
  - **numbering pages:** numbers pages for printing
  - **align files:** appends files for page numbering
  - **lines per page:** the maximum value for lines per page
- **Format and TAB:**
  - **set TAB size:** sets the TAB size for interpreting TABs
  - **font for header:** sets the font used for the header text
  - **font for body:** sets the font for the body text (file contents)
- **Textprinting:**
  - **cut lines:** cuts the lines if too large. If unchecked, lines are broken
  - **replace non-printing character by space:** replaces characters that the printing charset doesn't support by space characters
- **Other Options:**
  - **table of contents:** adds a table of contents page that contains information about printed files, page numbers etc.
  - **Highlight bars:**
    - **highlight bars:** highlights rows for printing
    - **cycle fo change:** sets the number of rows to change highlight style
  - **Wrapped line:**
    - **mark wrapped lines:** lines that are broken get marked for printing
    - **value for wrapped line:** the preset value for the new line the line is broken into.

Underlay

- **Text:** sets the text to underlay
- **Position:** sets the position of underlay text
- **Font:** sets the font used for the underlay
- **Angle:** sets the angle for the underlay text
- **Gray scaling:** sets the grayscaling for the underlay text
- **Style:** sets the underlay text to be printed as outlined or filled

7.6.2 The Printing Dialog

Direct Printing Options

- **Program:** sets the printing program to be used for printing: a2ps, enscript or lpr
- **Printer:** sets the printer to be used for printing
- **Output location:** for printing into a file, select the output location
- **Orientation:** sets the orientation for printing, either landscape or portrait
- **Copy:** sets the amount of copies to be made per page
- **Paper Size:** sets the paper size to be used
- **Output Format:** for using enscript as printing program, you can choose between postscript or html printing
- **Default Printsettings:** sets the default settings to use for printing
- **Outprinting:**
  - **Page Printing:** sets one or two pages on one sheet
  - **Pages:** select all,odd or even for using enscript
  - **Pretty Print:**
    - **pretty-print:** enables pretty-print mode for enscript
    - **color:** use color printing
    - **Pretty Print Mode:** sets the printing mode dependent on your file format

File Selection

The file selection dialog is accessed by the "Files" button on the printing dialog. The file selection allows specifying which files to print out by certain criteria:

- **File Selection:**
  - **current:** the currently opened file visible in the editing window
  - **all in project:** all files included in the project currently opened
  - **self chosen files:** allows choosing the files to be printed by file-selection
  - **all cpp files:** prints out all source files of the project
  - **all headers:** prints out all header files of the project
  - **changed files:** allows specifying files that are changed in a timespan:
- **Changed Files:**
  - **Between:** specifies all files changed after:
    - **Date:** the date the files were changed
    - **Time:** the time the files were changed
  - **And:** specifies all files changed before:
    - **Date:** the date the files were changed
    - **Time:** the time the files were changed

- **Self Chosen Files:** available if self chosen files is selected (see above)
  - **add:** press this to add a file selected for printing in the lineedit field left
  - **delete:** deletes a selected file from the printing list
  - **clear:** clears the printing list

Print Preview

Print Preview is available for you to control how the output will look like. Therefore, KDevelop uses the program *ghostview* or *kghostview*. When the preview-button was pressed on either printing dialog, you will be presented a template output that shows you the effect of the currently set options.

Chapter 8

Projects

8.1 Projecttypes

8.1.1 Programs

KDevelop creates a projectfile with the .kdevprj ending. This file contains all your project information, so be careful not to delete it. It is stored in the project's base directory and has to be opened to load the project. The projectfile keeps all information for your files like the file properties, install path, distribution status and compiler options (CXXFLAGS). Setting file properties allows you to keep track of where the files should go.

With KAppWizard, you can create a new application project according to your choice of application type. For now, KAppWizard generates four kinds of frame applications, as:

- a single document interface (SDI) KDE-application including a menubar, a toolbar and a statusbar. It contains basic control resource management to allow extending the frame application into a unique KDE application. The application frame also contains statusbar help messages as known from commercial products just like KDevelop itself. From the programmer's point of view, it is based on three application-specific classes leaning on the MVC-concept (Model-View-Controller). Technically, the base classes may not be viewed that way, but it's construction is at least created most logical to create applications with a GUI.

- a KDE-based application frame window. This application type offers most flexibility to those wanting to develop their program from scratch, but can also be used as a basis for developing wizard applications or modules.

- a Qt-only based program framework. For those programmers who want to use the Qt-library as the GUI interface alone, we tried to offer you a smart framework to support your application development. As Qt programming is fully supported, you should have no problems to create a full-functional application by using Qt only.

- a C++ program framework. This application type is intended for those wanting to write a terminal based C++ program. Just remove the "Hello World" line in main() and construct your classes the same way with KDevelop as for KDE applications.

- a C program framework for C programmers. This is as well a terminal based application but only uses the C compiler.

Additionally, KDevelop enables you to work with already existing projects. Those can have any options set by the programmer himself by configure and Makefiles. As far as the execution and the build-process is concerned, the current state only allows the same structure as for the other baseclasses. Create a custom project with the application wizard and add your files to the project to allow scanning by the classbrowser.

To ensure the build process, your custom project has to have all sources in a subdirectory matching the lowercase name of your project; the execution of the binary is also restricted to this lowercase project name.

Mind that KDevelop does not write any information into Makefiles or configuration files. You are responsible for any project behavior and build settings yourself.

8.1.2 Libraries

A general project-type to create libraries is not available at the moment. Anyway, building libraries is not impossible with KDevelop. Here are a few guidelines and workarounds:

- Whenever your project subdirectory gets another subdirectory that contains source files, KDevelop will build a static library of these. That means static libraries are already supported by automatic creation in order to sort project sourcefiles. Mind that the static library is part of the binary later and won't get installed.
- to create a shared library, you have the option to create another project subdirectory. The sourcefiles that are created in this subdirectory are included in the project and are therefore available in the classviewer as root classes. To create the shared library, the *KDevelop Programming Handbook* offers a Makefile.am template. If the subdirectory's Makefile is added to the configure.in script, you only have to run "Autoconf and automake" and "Configure" to create the Makefiles. The build instead is only possible from a make-command within the subdirectory, as KDevelop invokes the build from within the original project subdirectory. Another possibility to create shared libraries is changing the Makefile.am of the original project-subdirectory according to the template in the Programming Handbook by hand following the rules for project modifications explained in chapter 8.4.5 (Project Hacking).
- for installing a shared library, you have to watch the KDE-File-system Standard as explained in *The KDevelop Programming Handbook*

8.1.3 Multiple Targets

For some projects, the facilities of KDevelop at it's current state will not last. Those are projects that include multiple targets like packages containing several applications. As commands like "Execute" require that only one target is build by the developer, those types of projects are only supported in the way that you have to write your own entries to the Makefile.am's and creating your directories for the additional libraries or binaries to build. Nevertheless, a build-process always invokes your make-program independent from what actually the targets are; so these functions still can be used (with the restriction that the build is invoked from the main project subdirectory).

Another way to still work with this type and to still have access to the binaries themselves are creating empty projects and move their subdirectories in conjunction with the project files to the directory containing all sources later. Then you could load each target independently by its project file; this also allows executing and debugging the target.

Multiple binaries or libraries within the main project subdirectory are possible with following the rules explained in section 8.4.5 (Project Hacking) and the following guidelines for editing the main project's subdirectory Makefile.am (all modifications outside the KDevelop write area):

- add your target to the bin_PROGRAMS if it is an executable
- add your library declaration line if it is a shared library
- add the same declarations like the original project binary is build:
  - newtarget_METASOURCES
  - newtarget_LD_FLAGS
  - DISTCLEANFILES
  - copy the messages: entry for the original binary and replace target_SOURCES with newtarget_SOURCES, target.pot with newtarget.pot
- add your sources like the KDevelop write area contains outside the write area for your binary or library
- for installing static libraries, create the library with KDevelop's auto-creation inside subdirectories. Then modify the Makefile.am outside the write area according to the needed settings

8.2 New Projects

The KDevelop Application Wizard allows the creation of four different types of projects, constructing a framework for each. All projects use the GNU standard development tools as described in the requirements section of this handbook. For KDE applications, the wizard offers two different frameworks, the KDE-Application, providing a complete application with base classes for document type, view and application. This includes the creation of a menubar, toolbar and statusbar, the mainwindow inherits the KTMainWindow class. The KDE-Mini-Application only gives a view which is empty. This type of project can be used for the creation of small desktop tools or other simple applications without too much change on the existing code provided by the framework. A Qt-Application offers the creation of a Qt-only program if you wish no dependencies towards the KDE-libraries for end-users. The Qt-Application also is created by using tree base classes like the KDE-standard-Application and provides a menubar, toolbar and statusbar.

The C++ -Application type offers a framework for creating commandline applications. It is ready to run and by default displays "Hello World" as the only action. This may be useful for C++ learners who would like to program applications without using a GUI first or for those working on commandline programs written in C++ or C. C programmers can also take advantage of the C-only project, which requires any C-compiler at the minimum.

The wizard asks for your project name, version and the location where the project directory will be build. Further, your Name and Email address are asked for inserting them in header and cpp templates on top of your source files as well as for entries in the *.lsm file.

8.3 Opening and Closing of Projects

KDevelop is by default configured to open the last project loaded when starting. This allows a fast start, but you may want to change that to just start the IDE without any project. To prevent the default behavior, disable the "Load last project" option in the 12.3 (KDevelop Setup) dialog.

To open another project, select "Open" from the "Project" menu or press the "Open Project" button on the toolbar. In case you've got another project currently open, this will be closed. If your current project then contains files that aren't saved, you are asked to save all changed files. Thereby, you can select which file to save and which you want to close without saving, or to save all at once.

Then you get an "Open Project" dialog, where you can change to the directory containing the project file to be loaded. KDevelop project files have the **.kdevprj* mime-type which is also displayed by a project icon. Select the project file and press "Open". When loading a project, the Class Browser scans all files and builds the initial classtree, so you can start working on the project by using the Class Browser directly.

Another comfortable way to open a project is to select the project file in the KFM, the KDE File Manager. This will start KDevelop with loading the selected project file. You could as well open a project by commandline, entering `kdevelop projectname.kdevprj`.

When closing KDevelop, your project file will be saved automatically and the IDE detects if you have changed any project files. Then you will be asked to save any changes before exiting. The available options for closing are the same than for closing a project before loading another.

8.4 Editing a Project

After you created a new project with the KAppWizard, the usual task is to extend the project by editing the already created sources and adding classes, pixmaps, pictures and whatelse you project needs. Now, editing a project means that you can change the default project after its generation by menus and dialogs according to your needs. The next section therefore describes how you can add existing files and classes as well as creating new files. This is needed for building your project, but this won't help you for the installation process by the end-user. Therefore, section 8.4.2 (Setting Project File Options) describes how to set File Properties especially for additional files that you want make to install, like documentation or pixmaps. Another part in project maintaining is adding translations for applications that support internationalization, which is described in 8.4.3 (Adding Translations).

8.4.4 (Extending the Project Documentation) covers questions on how to create a good set of documentation for online help enabling the end-user to help himself in case of troubles as well as how to use your product.

Finally, 8.4.5 (Project Hacking) describes how you can eventually work around the KDevelop project management in special cases.

8.4.1 Adding and Removing Files and Classes

Adding a new file is often used when you think that you should separate your class implementation file into several ones. Then you need to create an new file which will contain the part of the class implementation you want to move. You can do this by choosing "New" from the "File"-menu, opening the "New File" dialog. This enables you to specify the file-type, name and the location. When entering the filename, KDevelop automatically enters the extension for you, but you can as well change the extensions towards your own preferences. Further, you can include the header template for sourcefiles, so you don't have to copy this into your new file yourself. Also, you can decide, if the file is included in the project or not. Mind that this doesn't cover the installation destination; this has to be configured later by setting the file preferences.

After the new file has been created, the project file will be updated as well as the according Makefile.am's. To add a complete class to your project, you can construct a new class with the Classgenerator which is invoked by "New Class" from the Project-menu.

In case you have an existing project and you want to add certain classes that you want to re-use in your project, select "Add File(s) to Project..." from the "Project"-menu. Select "Add existing files", which will open a dialog to let you specify which files will be added to the project and the destination directory. Mind that the files will be copied into the specified directory and added to the project. You should call "Make" after adding sources; then your added files will be included in the build-process. In case you want to add e.g. pixmaps that have to be installed by the end-user, you should update the File Properties for the added file and specify the installation path (see 8.4.2 (Setting Project File Options)

To add a file to the project that is already in the project directory, go to the RFV, where all project-included files are displayed by their project status by a red ! over the file icon. Select the file you want to add to the project (which has a normal file icon) and press the right mouse button to get the popup-menu. Then select "Add".

Files can also be removed from your project. This could be needed in cases where you don't want to use pre-generated files that are already given by the KAppWizard. To remove a file, you have additionally the option between removing a file from the project or delete it completely. To remove a file from the project, select the file in the LFV or the RFV, press the right mouse button and select "remove". To delete a file, select "delete physically".

8.4.2 Setting Project File Options

The file properties dialog can be accessed via the project menu or within the LFV by a right mouseclick. It shows the project files in groups as they are sorted in the LFV and displays the file properties like file size, file type and if the file is included in the project as well as the installation path if the file is going to be installed by the end-user's make install command. It is important for documentation files as well as pixmaps to specify the location where the files should go when the project is build and installed by end users, so you have to set up those locations. For standard KDE location macros you should look in your Makefile.am where the location macros are specified.

8.4.3 Adding Translations

As KDE allows configuring your desktop and the behavior of your applications, you also have the option to choose the language that your application uses, in regards of the use of online-documentation as well as the application's look. For documentation files, this seems a trivial task. You would add a subdirectory labeled by the desired language, e.g. `de` for German, to the `docs` directory of your project and copy the english documentation into that directory. Then you would generate the documentation and set all project file options for the installation directories. Then you could start translating the SGML file to your desired language and regenerate the documentation; then you're done. For the application, this seems a bit more difficult for the programmer. Now, we want to explain how to enable internationalization support for your application and how to add the languages you want to support.

First, you have to enclose all visible strings of your application that appear in bars or dialogs with the `i18n()` macro. This macro is a replacement for the function `klocale->translate()` of the KLocale class and much easier to use. As this macro is declared in the `kapp.h` include file, you have to add `#include <kapp.h>` to the source file or the class declaration file of the class that makes use of the macro. Also it should be mentioned, that although `i18n()` is a macro and therefore you could think about using the original function, this won't work because the strings that are set up for translation have to be read out of the sources and get stored in the application's translation file

(<YourApp>.pot in the /po subdirectory). This task is done by the program xgettext, and to do this, you would enter `make messages` in your project directory containing the sources. As xgettext depends on the `i18n()` macro, the original function won't do the job.

For translations themselves, you first have to create the message file containing all strings that are used in your sources with the `i18n()` macro. This can be done by choosing "Make messages and merge" from the Build-menu. Then you have to add the languages that your application wants to support. Therefore, select "Add Translation File" from the Project-menu. This opens a language selection dialog. Select the language and press OK. This will build the ASCII file containing entries for the filename of the string and the line where the original string is placed. Then you will see a `msgid` line containing the string to translate, followed by `msgstr`. The msgstr line is mostly empty except for the translations already provided by the KDE-libraries. Those have to be filled with the according translations of your language.

You could think about writing the translations by hand, which can also be done. But the KDE-SDK offers the use of the program KTranslator, which ready the already existing files from other applications installed on the system, so you can reuse already translated strings to support your language.

To access KTranslator, the easiest way is selecting a <language>.po file in the /po directory either in the LFV or in the RFV. This opens KTranslator and lets you do the translating easily. Mind that you have to set up KTranslator's properties yourself to include the author name and the language as well as the destination file. KTranslator by default only opens your translation file.

For all translation files, `make` uses the program `msgfmt` to format your message files to use with the binary, but you don't have to take care for that, as well as specifying the destination directory for installing the translation files; this is all done by KDevelop automatically.

For more information about internationalization support, see `<http://www.kde.org>`; a lot of people are doing translations for you to support their language. You will find a list of email-addresses of the translators you could write to and who will help you with this. Also read 11 (The Dialog Editor) and *The KDevelop Programming Handbook* where questions about internationalization are covered again.

8.4.4 Extending the Project Documentation

All projects created with KDevelop contain a pre-configured documentation, which already contains standard chapters for installation, project name and version as well as the author name and email-address. As KDevelop uses SGML-templates, it's very easy to extend the documentation to a full descriptive helpsystem. The only thing you have to do is editing the SGML file, placed in docs/en as index.sgml. The reference documentation included with your sgml-tools package can be added to the Helpbrowser and allow you direct access to special tags as well as a short description of how to extend the documentation. SGML has a lot of advantages, whereas KDE makes wide use by this documentation type with the additional KSgml2Html tool. This creates the typical KDE-style documentation and makes it look nicer. Anyway, the sgmltools alone are enough to produce a html output that is already included in your application. To create the documents using KSgml2Html, install the tool and run "Make User-Manual" from the Build-menu. The Documentation Browser allows a direct controlling of the output by selecting "Project User-Manual" from the Help-menu or the according icon in the DOC-tree. Then you can browse the documentation online in KDevelop and have a better overview by the output over errors that result in missing tags.

Now, while extending the documentation, you can't avoid that additional files are produced that have to be included into the project as each `sect`-tag creates a new HTML file. The output generated by the KAppWizard is already included in the project, so you don't have to care for their installation path. What you have to watch out for is any index-xx.html file, where xx is higher than 6 (as the first six pages are already included in the project). After generating the documentation, switch to the RFV and browse to your documentation directory. Press the right mouse button over the files to add and select "Add". Further, KSgml2Html adds the KDE logo to the documentation directory. This file, `logotp3.gif`, has to be added to the project as well. Then you have to select the "File Properties" either from the Project menu or by the popup menu in the file-viewers. The easiest way to set the installation path is to select a documentation file already set up for installation such as `index.html`. You see that `Install` is checked and the Installdir+filename already contains the destination. Mark the Installdir and enter CTRL+C to copy the installation path to the clipboard. Then select the file you want to specify for installation. Enable `Install`, this will enable the installdir-entry field, already containing the filename. Place the cursor in front of the filename and enter CTRL+V to insert the clipboard contents (which was the installation path copied before). This is the fastest way to specify the installation path. For more options about specifying installation destinations, see *The KDevelop Programming Handbook*.

8.4.5 Project Hacking

When working with a project, you should never edit the project file by hand. This will prevent KDevelop from loading your project correctly under certain circumstances, as well as changes will not result in updating the Makefiles. To change any settings for your project, you have to use the given menu entries like e.g. for adding files or setting file properties. For experts that are not comfortable with certain options for e.g. the Linker or need additional project configuration, you should work yourself into the Makefile.am macros and add all changes in the Makefile.am's after the section separated with the entry "KDevelop write area". As the GNU-tools are using the commands at the end of all macro files, you can overwrite KDevelop's settings with this. Mind that this will prevent using KDevelop for any changes related to project configuration.

8.5 Compiler and Linker Flags for Projects

Each new project contains all needed options for the Compiler and Linker as well as general settings. By default, your project is set to use debugging by the -g flag, and warnings are set to the standard, -Wall. This ensures you can debug your application and detect constructions that may cause program errors. For some applications, you would need additional Compiler or Linker flags, especially if you're using libraries that are not currently included by the Linker. Then you need to update the project by configuring the correct settings with the Project Options dialog. See 9 (Build Settings) for more information how and where to set Compiler Options, Warnings and Linker Options.

8.6 External Projects

Existing projects can be converted to KDevelop projects by selecting "New" from the project menu. The following dialog creates an empty project file with your project name, version and type information as well as your name and email address. Then copy all your compilation and construction files to the new project directory and choose "Add Files" from the project menu. The files selected will be copied into your project directory and the Makefile.am's are updated. Please change all Makefile.am entries existing before the conversion towards the now created entries done by KDevelop in the KDevelop area. Test, if your program can still be compiled and installed after the conversion to ensure the project's consistency.

Chapter 9

Build Settings

The Project Options dialog, accessed by the project menu, lets you specify all needed parameters for your project. Those will be used for the Makefile.am's and the configure.in script (e.g. version number change or compiler warnings) and thereby set the compilation preferences as well.. After changing the project options, you should invoke "make clean" or "rebuild all" to compile your project with the new options. Please mind that debugging is only available if the project options are set to create debugging information, the amount can be set with the debug level (0-3). If you add functions that belong to a library that is not included in the Linker flags, your program will not link correctly if those are not updated, so keep track of your library linking.

For a release build of your application or for distributing the sourcecode package, you should watch the following standard settings:

- disable debugging
- enable optimization and set optimization level to **-02**
- set the compiler warnings to **-Wall**
- for each new release, increase the version number and update the project.lsm file for version and requirements.

9.1 General Options

The first page of the Project Options dialog sets the general options for your project. These are project name and number, the handbook sgml file which is used for generating a set of HTML files that are included in the project and specific information about the author. The short description field is for additional information that you want to include like the program's purpose and the like.

9.2 Compiler Options

The compiler options page sets the compiler flags towards your target, debugging and additional.

9.2.1 Target

The target box contains three options that can be set:

Target Machine: You have the choice to set the target machine option here by choosing between your machine (default) and i386v, which is the option if you configured your Compiler as a cross-compiler for an Intel 386-compatible machine running System V. This option sets the -b flag to the Compiler. Usually you would leave this to the default.

only syntax-check: If checked, the -fsyntax-only flag is set. This means that the Compiler will check your code only for correctness in regards to the syntax, but doesn't check anything beyond that.

Optimize: You can enable optimization for your build process by this option, meaning to set the -O flag. If not checked, the flag is set to -O0, so no optimization will be used. If you enable optimization by checking this option, you can as well specify the optimization level below from 1 to 3.

For a release version of your application, enable optimization and set the level to 2.

9.2.2 Debugging

Right of the Target box, you can see the debugging-section. This means that you can set your Compiler to include information for debuggers within the final binary, so the programmer can follow the execution of the application with the debugger in direct context to the sourcecode.

Enable debugging therefore sets the -g flag; the debugging level specifies the amount of information to be included in the binary. Available are level 1 to 3 to choose from. Mind that the binary execution will be slower by setting any debugging option and that the binary size will increase by the debugging level.

Generate extra information for gprof: sets the -pg flag, resulting that the Compiler will include information for the gprof program that displays caller graphs of your program's function.

Store temporary intermediate files: sets the -save-temps flag. This will result in storing the usually temporary files produced by the preprocessor and the assembler. A compilation of a sourcefile will therefore produce three output files: an *.o file which is the final output of the Compiler, an *.i file produced by the preprocessor and an *.s file as the output of the assembler.

For a release of your project, disable any debugging.

9.2.3 Additional Options

The text entry field on the bottom is intended for you to manually set any flags for the Compiler by setting the CXXFLAGS environment variable in the Makefiles, so make sets the flags before the build process and reset them afterwards. For a complete description of all available Compiler flags you should see your Compiler documentation; for gcc and egcs this can be done by **man gcc**; **man g++** will show you information about the c++ script that is used to lead the Compiler.

9.3 Compiler Warnings

The following gives a description about the Compiler warning options that can be set on page 3 of the project options dialog. The explanations are taken from the man page for GCC, egcs version 1.1.1. The warnings themselves are diagnostic messages that indicate that constructions may cause errors.

-Wall

Standard '-W' options combined.

-W

Compile with -W. This option sets options not included in -Wall which are very specific. Please read GCC-Info for more information.

-Wtraditional

Warn about certain constructs that behave differently in traditional and ANSI C.

-Wundef

"Warn if an undefined identifier is evaluated in an '#if' directive.

-Wshadow

Warn whenever a local variable shadows another local variable.

-Wid-clash-LEN

Warn whenever two distinct identifiers match in the first len characters. This may help you prepare a program that will compile with certain obsolete, brain-damaged Compilers.

-Wlarger-then-LEN

Warn whenever an object of larger than LEN bytes is defined.

-Wpointer-arith

Warn about anything that depends on the «size of» a function type or of void. GNU C assigns these types a size of 1, for convenience in calculations with void * pointers and pointers to functions.

-Wbad-function-cast

Warn whenever a function call is cast to a non-matching type. For example, warn if `int malloc()` is cast to `anything *`.

-Wcast-equal

Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a const char * is cast to an ordinary char *.

-Wcast-align

Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a char * is cast to an int * on machines where integers can only be accessed at two- or four-byte boundaries.

-Wwrite-strings

Give string constants the type const char[length] so that copying the address of one into a non-const char * pointer will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using const in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make '-Wall' request these warnings.

-Wconversion

Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

-Wsign-compare

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.

-Waggregate-return

Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

-Wmissing-prototypes

Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. The aim is to detect global functions that fail to be declared in header files.

-Wmissing-declarations

Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files. -Wredundant-decls Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

-Wredundant-decls

Warn if anything is declared more than once in the same scope even in cases where multiple declaration is valid and changes nothing.

-Wnested-externs

Warn if an extern declaration is encountered within an function.

-Winline

Warn if a function can not be inlined, and either it was declared as inline, or else the -fin*line-functions option was given.

-Wold-style-cast

Warn if an old-style (C-style) cast is used within a program

-Woverloaded-virtual

(C++ only.) In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

-Wsynth

Warn when g++'s synthesis behavior does not match that of cfront.

make all warnings into errors

(-Werror) Treat warnings as errors; abort compilation after any warning.

For a release of your project, it is recommended to enable **-Wall**.

9.4 Linker Options

The Linker Options for your current project can be set by the last page of the project options dialog. You have to enable those libraries that your application uses to link them to your binary with the Linker, e.g. your application uses the class KFileDialog. As the class KFileDialog is part of the KFile library, you have to enable kfile. For classes or functions that are not listed as checkboxes, use the **"additional libraries"** field.

9.4.1 Linker Flags

remove all symbol table and relocation information from the executable:

This means that all redundant information will be removed from the object files and the binary, resulting that debugging will not be possible. For as long as your application is in a development stage and not released as final, you should leave this option disabled.

prevent using shared libraries:

This option disables the use of shared libraries on systems that support this. On systems using no shared libraries, this option will have no effect.

additional flags:

Here, you can enter additional flags for the Linker, setting the LDFLAGS environment variable by make. The available options can be taken from the man page for **ld** or your Compiler manpage.

9.4.2 Libraries

The libraries section contains checkboxes for the most needed libraries in conjuction with Qt/KDE application development. You have to enable those libraries that your application uses, otherwise the Linker will complain about unresolved symbol tables.

X11

The X11 library. Recommended for all X-Window programs.

Xext

The X11 extension library. Also most X-Window programs depend on Xext.

qt

The Qt-library. Recommended for Qt and KDE applications.

kdecore

The KDE Core library; contains the classes for KDE Application frameworks.

kdeui

The KDE User Interface library; contains KDE-specific widgets.

khtmlw

The KHTML Widget library.

kfm

The KFM library containing classes for KFM functions.

kfile

The KFile library. Contains file dialogs etc.

kspell

The KSpell library. Contains an interface for programs to use Ispell for spell-checking.

kab

The KAdressBook library. Needed for access to the addressbook as well as providing addressbook widgets

additional libraries: Here you can enter additional libraries that your application needs, e.g. the KOM library. Set the libraries with the -l option; for the example -lkom.

9.4.3 Make

As GNU make supports some useful options, the Project Options dialog contains a page called "Make-Options", where those can be en/disabled. The available settings are:

Print debug information

prints out all information about the project files and what make determines for rebuilding them.

Continue after errors

tries to continue with the compilation after an error occurred (e.g. a file couldn't be compiled due to an error)

Print the data base

prints out the make-database for the current process which contains the changes from the last build-run.

Environment variables

give the current environment variables a higher priority than the currently used variables in the Makefiles.

No built-in rules

doesn't use built-in rules for make.

Touch files

don't run the Compiler on changed files; instead only touches them. This sets them as already processed by make.

Ignore all errors

Ignores all errors that occur

Silent operation

doesn't print out any information about the build-process

Print working directory

prints the current directory during the make-process.

job number

sets the amount of parallel processes for make. For a single-CPU system we recommend setting this to one or two.

set modified

sets the selected file modified. Choose the file by clicking the folder button on the right. Setting a file modified means that the file will be processed by make and compiled if it is a source file.

additional options

set additional options to make; those can be found in your local man page for "GNU Make".

Chapter 10

The Class Browser

10.1 The Classviewer

The KDevelop Classviewer is one of the most useful and important tools that the IDE provides a developer for managing his project sources. When a project is loaded, a class parser reads all project sources for functions, classes etc., then displays the results in the CV treeview. This chapter shows you how to use the Classviewer and the provided functionality and how it can improve your work.

Classes and their methods can also be accessed by the browser toolbar. There, the left combo selects the class; the right lets you choose the methods of the selected class. When you select a method, the classbrowser will automatically bring you to the implementation file and sets the cursor to the method. Finally, the class-assistant button on the right of the method combo will bring you to the declaration of the method on a click; on another click to the definition. The delayed popup menu, displayed by the down-arrow on the button, offers additional functionality that is also available in the classviewer's context menus, such as:

- Goto Declaration: browses to the declaration of the method
- Goto Definition: browses to the definition of the method
- Goto Class Declaration: browses to the class declaration
- New Class: opens the New Class dialog to construct a new class
- Add Method: adds a method to the selected class
- Add Attribute: adds an attribute to the selected class

10.1.1 Available Objects

By available objects, we describe a term that means that C++ code can be seen as a collection of objects- classes, their members, global functions and the like. The classtree displays these objects logically and orders them by character, so they are easy to locate in the tree as well by their according icon. Therefore the classtree contains a "Classes" and a "Globals" folder. Thereby the "Classes" folder generally contains the project's classes; if your project contains subfolders to manage your sourcefiles, these are also displayed by their original folder name and contain all classes that are stored in the files located in the subfolder.

Further, when popping up a class, the classtree displays the class contents by separating methods and attributes. As these can have attributes as well like public, private and protected, these are displayed by modified icons as well. You see that a class displayed in the Class Browser contains all the objects that are present in the class declaration.

Now, classes are a very common thing when programming in C++ and will contain most of the code. But applications also contain objects that have a "Global" appearance to the program. These would be structs, functions etc. Especially the `main()` function appears in every application and you need to modify it in one or the other way sometimes. To access these objects, the Classviewer provides the "Globals" folder, containing subfolders for the following object types:

- Structs
- Functions
- Variables

As the icons displaying these items are similar to those used in the class-viewer, their meaning is easy to guess and to remember by the programmer.

Finally, it can be stated that the classviewer displays your project graphically by their objects related to their appearance in the code. In the following section you will learn how to use the classviewer and it's tools in conjunction with your code.

10.1.2 Browsing Object Declarations and Implementations

The Classviewer's best strength is providing a fast and good access to his code by the code's contents independent of the location within files. Therefore, selections by a mouse click will result in the following actions:

- **On a classname:** Switches to the class declaration
- **On a class method:** Switches to the method implementation
- **On a class attribute:** Switches to the attribute's declaration in the class declaration
- **On a struct:** Switches to the struct's declaration
- **On a global function:** Switches to the function's implementation

Now, this provides you the most needed access towards code objects. Apparently, it may be needed to change a method's header, resulting that you have to change it's declaration in the class as well as the implementation. The classviewer supports this by providing right-button context menus over items. On a method or function, this means that you can select where to go:

- **Go to definition:** Switches to the implementation- this is the default for a left click as described above
- **Go to declaration:** Switches to the declaration of the method or function.

By this behavior, the Class Browser offers you access to every place you have to go for coding your C++ application.

The following sections give you a description of the other tools that the Class Browser offers- you will find them very useful when working with large projects as they enhance working object-orientated on C++ programs.

10.2 The Classtools

The classtools are dialogs that makes it even more easier for the developer to get more information about his project's classes. The classviewer displays all objects by their occurrence in the code, but you sometimes want to get more information about classes without having to look inside the code. Therefore, the classtool dialogs are specialized on displaying specific class attributes.

The classtool dialog is invoked by the popup menu over a class in the classviewer. Select "Classtool", and the dialog will appear. To get informed about a class, select the class in the combo box on top. Then the buttons in the toolbar provide functions that give you specific trees for your class. These are:

Parents:
: The parents of the selected class, which means the class it inherits. This is useful for multiple inheritance as well as to see why a class behaves in one or the other way, e.g. for dialogs your parent class could be `QWidget` or `QDialog`.

Children:
: Displays the children classes that inherit the current class.

Clients:
: Classes that make use of the selected class by an attribute in their class declaration

Suppliers:
: The suppliers that give attributes to the selected class.

Attributes:
: The attributes of the class by their name

Methods:
: The Methods of the selected class

Virtual Methods:
: The virtual methods that a class provides

Further, the selection of the attribute public, protected, private or all shows Attributes, Methods and Virtual Methods by their attribute value.

10.3 Managing Classes

The Class Browser additionally allows adding methods and attributes directly by dialogs. This means you don't have to type the classdeclaration and the implementation header yourself. After adding a method, you only have to set the formal parameters to the implementation header and, if the method requires an attribute, to the declaration.

» How to add a method to a class

1. select the class you want to add a method
2. press the right mouse button; the popup menu appears
3. select "Add member function".

4. the "Add member function" dialog appears.
5. insert the type, declaration and documentation for the method
6. specify the access and the modifiers for the method
7. press OK to exit the dialog

For adding a variable, this is the same action, just with selecting "Add member variable" in the popup menu.

The difference between the actions of these dialogs is that the adding of a variable will add the variable to the classdeclaration, the adding of a method will add the method's declaration and the method's implementation header to the sources. As the classviewer directly updates himself, you have a direct access to the new method implementation, so you only have to fill out the code for the actual purpose of the method.

Chapter 11

The Dialog Editor

The KDevelop integrated dialog editor allows the easy construction of widgets and dialogs your application uses all by graphical means. You see the direct appearance of your dialog as it will be presented to the user. Using the dialog editor is usually the first step you would take after creating a new project with the KAppWizard to create your main view, the user interaction dialogs and, after finishing the graphical work, the code generation. This way, your project will contain all the usually considered "difficult" parts that normally would take a long time to implement. Then, the "rest" of your work is implementing the functionality in the generated code. This chapter deals with how to use the dialog editor to create your project widgets as well as what to do in case you see your widgets need corrections or additions during the further development process.

You can switch to the dialog editor either by selecting "Dialog Editor" from the "Tools"-menu or by the according toolbar icon. To switch back to the Project Editor, select "KDevelop" from the dialog editor's "Tools"-menu or by the icon in the toolbar.

What else does the dialog editor's interface offer you ? Mainly, that its appearance is almost the same as the project editor in term of the main view separation as well as the menu- and toolbars. This allows you to make yourself accustomed with the dialog editor very quickly and, as he completely interacts with the project management, stay in the dialog editor if you want to control your build-process. Actions that require to switch back to KDevelop will do that automatically for you, like accessing the documentation browser. Just select the menu commands, and KDevelop reacts to your will.

The following chapters give you an overview of the dialog editor interface, how to create a new dialog initially and how set up the properties of the child widgets that your dialog contains.

11.1 The Dialog Editor View

11.1.1 The Mainview

The dialog editor's view is logically separated by:

- The Widgets Tabulator, containing the "Widgets", "Dialogs" and "Items" tabs. These are described in 11.3 (Adding Widgets).
- The Widget-Editor, representing the editing view for creating your dialog. See 11.4 (The Widget Editor)

- The Properties window, containing a list of properties and their values, dependent on the currently selected widget in the widget editor. See 11.5 (Setting Properties) for information how to specialize the widget's behavior and look.

11.1.2 Menubar, Toolbar and Statusbar differences to KDevelop

In Dialogeditor mode, KDevelop changes the menubar, toolbar and statusbar slightly to provide the functionality you need for creating widgets. These are:

Menubar

"File"-menu: replaces "New" with "New Dialog". "Open" allows to open a dialog definition file.

"View"-menu: replaces "Tree-View" with "Widgets-View", en/disabling the Widgets-View tabulators; adds "Properties-View" to en/disable the Properties-View and "Grid Size" to let you specify the grid size in pixels for horizontal and vertical values.

"Build"-menu: replaces "Compile File" with "Generate Sources". This lets you actually generate the sources for your dialog.

Toolbar

The toolbar contains a new icon for "New Dialog" as well as a replacement for "Compile File" by "Generate Sources".

Statusbar

The statusbar is providing you information about the currently selected widget, especially displaying the values for X and Y size in the coordinate system. For Statusbar help, you are provided the same functionality than in Project-editing mode.

While changing widget size, the statusbar shows the current values for width and height of the selected widget.

11.2 Creating a New Dialog

After creating your project skeleton, you are provided a ready-to-run application, according to your preferences. As KDevelop provides the project types KDE and Qt application, the dialog editor recognizes this and offers widget construction by the widgets that are provided by the used libraries. To save time, you already should have a design in mind that will do the intended actions. For information about widget design, see *The KDevelop Programming Handbook*.

To create a new dialog, select "New" from the "File"-menu or select "New" from the context menu in the "Dialogs" tab. The "New Dialog" menu appears where you have to give KDevelop the dialog-specific information about the baseclass and the source file names and destination.

11.2.1 Dialog Class

The dialog class you can select, is the class that is inherited by your new widget, which is technically represented by a class itself. Therefore, you have the following options:

1. **QWidget:** the base class for all user interaction widgets provided by Qt. Used for main views and top-level widgets.
2. **QFrame:** inherits `QWidget` and is used by numerous widgets as a base class. This is useful for widgets that already want to have a `QFrame` functionality in addition to `QWidget` methods.
3. **Custom:** inherits a custom class that has to be set in the "Custom Properties". This could be an already designed class provided by your project or the libraries.
4. **QDialog:** the base class for dialogs that you would inherit for user interaction like setting properties or changing values.
5. **QTabDialog:** inherits QDialog and provides a dialog with predefined buttons and a set of tabs, which you will provide by the widgets that you create.

Custom Properties

For the inheritance of a custom class as selected in the dialog class field, you have to specify the classname, which goes to "Custom Class". The dialog editor uses this for the code-generation; therefore, you also have to insert the "Custom Header", where the header filename of the custom class has to be set.

11.2.2 Files

In the "Files" section, you have to enter the widget's specific information. This is the classname (which would be e.g. KColorSelectDlg for a dialog that allows selecting the color of a pen), the header, C++ and Data filenames. When inserting the Classname, the filenames are given by the dialog editor, but you can as well change the filenames.

Now, what about the files ? When you are ready with constructing the widget visually, you will have to generate the files that contain the implementation for your widget. As this will be a class, the dialog will exist by the header file containing the classdeclaration, a C++ file containing the method implementation for your widget's methods and slots. The Data file is the file that will contain a function that is called by the constructor of your widget, the *initDialog()* method. The file itself shouldn't be changed as it will contain the generated code from the dialog editor to create the widget on the screen. If you ever have to change values, you should do this by the constructor or be sure that you won't change the dialog during the development process, as the Data file will be overwritten each time the code is generated for your widget. The header and C++ file contain sections, where the dialog editor writes; these are marked by comments. After file generation, you can change any values and settings outside these sections; otherwise your changes will get lost by the next code-creation.

11.2.3 Location

For generating the widget's sources, the dialog editor needs to know the location where these will go. The default value for the output directory is the current project subdirectory containing the sources already present.

After pressing "OK", your default values are generated and an empty Widget constructor is opened. Then you are ready to go for creating your own widget. Mind that the dialog editor currently only supports static widgets without geometry management. If you're about to use geometry management for your widgets, you should make yourself accustomed with the classes that Qt provides for this, create a new class with the classgenerator and write your widget by hand. For more information, see *The KDevelop Programming Handbook.*

11.3 Adding Widgets

After specifying the dialogs or widgets class and filenames, you are ready to start creating the widget and filling it with contents. Adding low-level widgets to your dialog is a very easy task. Just select the widget you want to add from the "Widgets" tabulator on the left by a single click over the according widget icon. The widget will then be laid on the left upper corner of the currently opened main widget. An added widget then gets the default size of 100x30 pixels on the editor view. To move a widget, click over it to activate the drawing frame, which is displayed in dark grey with hot spots on the corners and on the center of the top, bottom, left and right sides of the widget. A cross-cursor indicates that the widget can be moved. To move it, press the left mouse button and keep it pressed. then move the widget with your mouse to the place you want to have it displayed later.

To resize a widget, move your mouse cursor over one of the hot spots of the already activated item. The mouse cursor then changes to a double-arrow indicating the directions, in which resizing can be done. Press the left mouse button and hold it pressed. The widget item will change it's size when the mouse is moved to the direction indicated by the cursor.

Further, the widget editor contains a lot of context menus to help you coordinate your work. Those are available over all items in the widgets tabulator and give you a quick help message window that shows the class name of the selected widget with a short description. Over a selected widget, the context menu shows the class name of the selected item and offers:

- Raise
- Lower
- Raise to top
- Lower to bottom
- Cut
- Delete
- Copy
- Paste
- Help

After setting the size and position, you can edit the preferences for the selected item on the Preferences window.

11.3.1 The Widgets Tabulator

The widgets tab represents the available widgets you can place on the dialog. If you want information about a certain widget, press the right mouse button over a widget icon and select "Quick-Help" from the popup-menu. Mind that the dialog editor automatically determines, if your project type is Qt-only or KDE. This prevents you from using KDE-widgets in a Qt application.

After you selected a widget item, it is placed with default sizes and values on the editing window and marked selected by a frame and darkened corners. To resize a widget, move your mouse over one of the dark spots and your cursor will change to display which resizing directions are possible. Then press the mousebutton and move the mouse while holding it. When you're finished with resizing the widget, release the mouse. While resizing, the statusbar displays the current position of the item by X and Y values and the current size by W(Width) and H(Height) values.

11.3.2 The Dialogs Tabulator

The dialogs tabulator is intended to let you open your project's dialogs by a mouseclick. As the dialog's structure is saved in a *.kdevdlg file within the directory that contains the generated files, only those dialog definition files are shown. Also mind that you don't delete these definition files.

On selecting a dialog, it will be shown as by the state it was saved in the last editing step in the Widget-Editor view.

11.3.3 The Items Tabulator

The Items tabulator lets you have an overview over the currently present widget items of the dialog hierarchically. This means, that, as your background represents the parent of all widgets within the dialog, it is shown on top of the tree. The children of the main dialog are then listed in the next tree-level.

On selecting an item, it gets marked in the editor view, as well as the properties are shown in the properties window. Using the items view is sometimes important if your widgets behavior depends on the parent-child relationship.

11.4 The Widget Editor

The Widget Editor is the main view that is placed in the middle and where you are constructing your widget. After adding items, those can be selected and resized, as well as moved to the place you need them. Over all items, popup menus provide a quick access to functions like cut, copy and insert.

11.5 Setting Properties

The properties window on the right is the place where you set the default behavior for the widget and its items. It displays the pre-set values for each selected item right away; changing values will result in direct changes on the Widget Editor view, e.g. naming labels or buttons.

To separate certain property values by their effect, the properties window contains four folders; selecting a folder will pop up all values for the properties group. All possible values are described below. Mind that the properties are dependent on the widget, e.g. a label and button will have a property for their on-screen name, while lineedits will have properties for methods like *setText()*.

For a complete list of the available values per item, you should see the class-reference of the widget which explains the used methods and all possible values. Note that most values are implemented in `QWidget` and are used for all widgets that inherit QWidget. Also mind that the final code does not contain any method calls that are unchanged by the user and therefore use the default values as given in the widget's constructors.

A complete list of the supported properties that can be set in the properties window for each widget item.

11.6 Generating Files

After creating a widget, you have to generate the sourcecode to make it available in your project. This can be done either with the "Generate Sources" from the "Build"-menu or by the according icon in the dialog editor toolbar. Your Makefiles will be updated automatically to include the new widget in the compiling process; therefore, after calling "Generate Sources", you can build your project again within the dialog editor. The output window pops up below the Widget-Editor window as in Project Editor mode.

Now that your project contains a new widget, your work as a programmer is to implement functionality to the used slots and eventually add other methods you may need. [1]

[1] See *The KDevelop Programming Handbook, The Dialogeditor* for more information about widget properties and sourcecode generation.

Chapter 12

General Configuration

This chapter describes how you can set your individual preferences about how KDevelop works. All settings addressed below can be found by the according entry in the Options-menu.

12.1 Configuring the "Tools" Menu

As KDevelop supports the use of third-party programs within it's user interface, you are able to configure any program that suits your needs towards application development. This can be done by adding programs to the already pre-defined ones in the "Tools"-menu. To change the tools-menu, select "Tools" from the "Options" menu. This dialog allows to specify the entry name, program and additional command line options you want to pass to the execution. To remove a program from the menu, select the entry name and choose "Delete". To add a program, specify the menuentry, where a & is used as a menu-accelerator; you may compare the already configured entries with the entry list. Select the binary and pass your commandline options. Then hit "Add" and the entry is added to the list. After leaving the configuration dialog, the tools-menu updates itself, so the new configuration is already usable without restarting KDevelop.

12.2 File Viewer Options

The Logical File Viewer can be configured by context menus completely. As it's intention is to separate files logically to keep a better overview over complex projects, one of the most used configurations is to create file groups. Those can be set by opening the context menu with a right mouse button click over the project icon displayed at the root of the tree. The menu offers:

- New File: Opens the New File dialog. Equals to the menubar command "File"-"New"
- New Class: Opens the Classgenerator to create a new class. Equals to the menubar command "Project"-"New Class"
- New Group: Opens a dialog to create a new group. There, set the group name and the file filter for the project files that will be displayed in this group.
- Show relative path: displays the files with their path name starting from the main project directory if checked; otherwise only the filename is shown.

Over a group folder, the according context menu offers:

- New Group: Opens the New Group dialog as in the context menu described above.
- Remove group: removes the group from the LFV.
- Properties: Opens the properties of the group. There, you can edit the file filters by a list of wildcards separated by commas.

12.3 KDevelop Setup

Make-command: The General Options dialog lets you configure KDevelop's general settings. First, you should set the make-command available on your system. If the selected program does not exist, KDevelop will warn you the next time you're invoking a make command.

Autosave:

If Autosave is checked, KDevelop will save all changed files periodically. The autosaving time range can be set to 3, 5, 15 or 30 minutes.

Autoswitch:

If Autoswitch is enabled, the KDevelop windows will switch on and off according to the usage context, e.g. if you switch to a documentation in the Help-menu, the Documentation browser will be opened, together with the documentation tree and the output window turned off. Startup:

For starting KDevelop, you have the option to enable/disable the start-logo to be shown during the time KDevelop loads. Further, if you don't like the last project to be opened on startup, you can disable the default behavior.

12.4 Changing Keyboard Shortcuts

The Configure Keys dialog lets you configure the KDevelop key bindings. Note that global keys can be configured in the KDE Control Center, such as open file and print. A key function can be configured by choosing the menu entry. Then the configuration can be changed by checking values like the Alt / Ctrl key etc.

12.5 Documentation

12.5.1 Directories

For setting up the documentation browser to work correctly, KDevelop needs some information about where the HTML-documentation is placed on the system. Therefore, the Documentation Path properties dialog needs the path of the Qt- online documentation in HTML as well as the path to the KDE-Library documentation.

Usually, the Qt-Documentation is placed in the same directory where Qt is installed; e.g. if Qt resides in /usr/local/qt, the path you have to enter is /usr/local/qt/html. For the KDE- Documentation, you have to set the directory to the root of the documentation, assumed all KDE-Libs documentation resides in the same directory. Both path's can be selected by pressing the according buttons, displaying a path-selection. If your system doesn't contain the documentation for the KDE-libs, you should first enter the next configuration dialog, Update KDE-Documentation. This will create the documentation to a path of your choice, also setting the KDE Library Doc path automatically.

12.5.2 Options

Update KDE-Documentation

For those users who don't have a recent documentation of the KDE-libraries, especially the documentation for the files installed on the system, the Update KDE-Documentation dialog creates a new one or updates existing documentations. This function requires your system to have KDoc and qt2KDoc installed, included in the KDE-SDK package. First of, you have to set up the path to your recent kdelibs sources, which is not the include-path for KDE ! Just enter the path to the sources, like: /home/rnolden/kdelibs-1.1/.

Then, you can choose three different installation modes, as:

- Delete old Documentation and install to recent Documentation-path: this assumes that you already have a documentation installed and is placed in the path the Documentation Path-dialog was entered. This will delete all documentation and install the newly generated documentation into the recent path.
- Delete old Documentation and install to new Documentation-path: this will result in deleting the old documentation as well as above, but gives you the choice to set up a new documentation place.
- Leave old Documentation untouched and install to new Documentation path: This is recommended for a new generation of the kdelibs documentation for users who didn't have one before and for those who want to keep the last documentation for an older kdelibs version.

The "new KDE Libs Documentation path" is to be set for option 2 and 3 of the installation mode. This is also recommended for users who generate a new documentation from scratch.

After pressing the OK button, KDevelop will create a subdirectory "KDoc-reference" in the documentation path containing the KDoc reference files. First, the qt library documentation classes will be indexed to connect the Qt documentation with the documentation to be generated for the kdelibs. So it is important that you have set up the Qt documentation path first to ensure that it can be found by `qt2kdoc`. Finally, the KDE libs will be indexed and the documentation will be build with cross-references to give browsing the most functionality.

Create Search Database

The Create Search Database dialog, accessed via the create-button, allows the programmer to create a database to search for a keyword interactively. To create and use the documentation search function, you must have the program glimpse 4.0 installed. Preset are the options to index the given KDE-Library documentation as well as the Qt-Documentation, assuming the path to the documentation files were set in the "Documentation Path" dialog of the options- menu. Additionally, the index can include directories the user can set up himself by the "additional directories to index"-field. After setting a path to an additional directory, the "Add" - button must be pushed to set the path. A path once set can be removed from the index by selecting the path in the path field and pushing "Remove". Furthermore, the user is offered three different modes for the index size: tiny, small and medium. The higher the index size, the more the index files will grow. On the other hand, a search in a bigger search-database will be faster and more successful, so we suggest choosing a "medium" size. For using the search function, see section 6.4 (Using the Documentation Browser).

Chapter 13

Questions and Answers

This section addresses questions by users that were answered by the KDevelop Team or by their supporters on the KDevelop mailing list during experiences with the current versions of KDevelop as well as bug reporting in general.

13.1 Bug Reporting

Another improvement of KDevelop is the integrated bug-reporting system via email. If you experience a bug, you have the option to send the KDevelop development team a bug report either by your email-client or by the bug-report dialog. All bug-reports are collected on the KDevelop web site and can be reviewed on `<http://fara3.cs.uni-potsdam.de/~smeier/kdevelop/bugarchive/maillist.html>`. You can also receive all bug-reports via subscription to the bug-report mailing list by sending an empty email to *kdevelop-bug-report-request@fara3.cs.uni-potsdam.de* with "subscribe *your_ email_ address*" as the body contents.

To send bug reports, please use this email address by your mail program. If you want to use KDevelop for direct bug-reporting, choose "Bug Report" from the Help-menu. You are presented the report dialog that lets you enter all necessary information about the bug you found. After pressing "OK", the dialog's contents is sent to the mailing list automatically.

13.2 Where to get Information

Q: I have a question which is not addressed in the FAQ file, nor in the manuals of KDevelop. Where should I turn to ?

A: In any case send all requests that are regarding KDevelop by subscribing to the KDevelop mailing list at *kdevelop@fara3.cs.uni-potsdam.de.* Send a mail with an empty header and "subscribe" as contents; then you can participate on the discussions. All questions should go there and will be addressed there as well. If you stick to that, you will get the most help by the developers and all users having the same problems, as well as helping to keep the FAQ up to date.

The KDevelop Homepage at `<http://www.kdevelop.org>` also contains a mailing-list archive that allows you to browse the mails already send by the subscribers, so you should look there first as most problems should have been addressed already by the team or other users.

13.3 Library and System Problems

Q: Wrong JPEG library version: library is 61, caller expects 62

A: There are 2 ways.

1. When the kdelibs are installed it installs header files for the jpeg libraries, these are version 61, however most distributions (Redhat) use version 62 libraries. To fix this just remove jpeglib.h from /opt/kde/include. The pukka include file for version 62 should then be picked up. However looking at the error message above it may be the other way round, in any case ensure you only have on version of the header file, the library and that they are consistent.

   It is useful to use the locate command to verify that I have the correct version of a library and header files e.g. updatedb locate libjpeg locate jpeglib

2. You must recompiled kdesupport without jpeg library (./configure –with-libjpeg –with-libgif).

Q:

```
make[2]: Entering directory ‘/usr/local/src/kdevelop-0.3/po’
cd .. && automake --gnu --include-deps po/Makefile
aclocal.m4: 2709: ‘AM_PROG_INSTALL’ is obsolete; use ‘AC_PROG_INSTALL’
make[2]: *** [Makefile.in] Error 1
```

A: Workaround for automake-1.4/automake-2.13 users: Just run "aclocal" manually, then it will compile.

Q: What must i do, if configure said ,that i need giflib23.

A: Try a newer snap of kdesupport, or maybe you have another giflib installed?

Q: How can I convert a KDevelop 0.2 project to a 0.3 one?

A: Please change the AC_OUTPUT in the configure.in to a oneline version

for example: old version:

```
AC_OUTPUT(Makefile \
kdevelop/kwrite/Makefile \
kdevelop/templates/Makefile
)
```

new version:

```
AC_OUTPUT(Makefile kdevelop/kwrite/Makefile kdevelop/templates/Makefile)
```

Q: I get the following Linker errors when using SuSE Linux with KDE 1.1, what do I have to do to get KDevelop linked ?

```
/usr/lib/libqt.so:
warning: multiple common of ‘QArrayT<char> type_info node’
ckdevelop.o: warning: previous common is here
ckdevelop.o: In function ‘CKDevelop::slotFileSaveAll(void)’:
ckdevelop.o(.text+0x784): undefined reference to ‘kdebug(unsigned short,
unsigned short, char const *,...)’
ckdevelop.o(.text+0x839): undefined reference to ‘kdebug(unsigned short,
unsigned short, char const *,...)’
```

```
ckdevelop.o(.text+0x89d): undefined reference to ‘kdebug(unsigned short,
unsigned short, char const *,...)’
ckdevelop.o: In function ‘CKDevelop::slotFileSaveAs(void)’:
ckdevelop.o(.text+0xd28): undefined reference to ‘kdebug(unsigned short,
unsigned short, char const *,...)’
ckdevelop.o: In function ‘CKDevelop::slotFileClose(void)’:
ckdevelop.o(.text+0x1216): undefined reference to ‘kdebug(unsigned short,
unsigned short, char const *,...)’
ckdevelop.o(.text+0x1263): more undefined references to ‘kdebug(unsigned
short, unsigned short, char const     *,...)’ follow  collect2: ld returned 1
exit status  make[2]: ***
[kdevelop] Error 1  make[2]: Leaving directory
’/home/LinuXDaten/Programme_Updates_Packete/KDE_Updates/Kdevelop_actual_snapshot/kdevelop-0.3/kdevelop’
make[1]: *** [all-recursive] Error 1  make[1]: Leaving directory
‘/home/LinuXDaten/Programme_Updates_Packete/KDE_Updates/Kdevelop_actual_snapshot/kdevelop-0.3’
make: *** [all-recursive-am] Error 2
```

A: If you have the SuSE rpm´s of KDE-1.1, you must recompile the kdelibs without the patch commited by SuSE and reinstall them or get an updated rpm of the kdelibs from `<ftp://ftp.suse.com>`

13.4 Usage Questions

Q: I see the KDevelop does not allow for usage of the delete key (or backspace deleting when text is marked).

A: go to "Options"->"Editor" and make sure that "Delete on Input" is enabled, then backspace and delete works.

Q: If I add files to my project, will they be automatically included and compiled ?

A: Yes, they are included in the Makefile.am’s then and if you make a "Rebuild All" (./configure updates the Makefiles), your new added files will be included as well.

Q: If I removed a file, I get some weird Linker messages. What is wrong with my project ?

A: If the removed file is a header file, that is automatically processed by automoc (running the Qt-Meta-Object-Compiler automatically on all headers), your removed header is still present as a moc-generated *.moc.cpp file and compiled. Remove the according *.moc.cpp file and rebuild the project.

Chapter 14

Authors

Main Developers:

Sandy Meier <*smeier@rz.uni-potsdam.de*> (maintainer, development coordinator and homepage provider)

Main development of: frame structure, IDE look'n feel, project management.

Stefan Heidrich <*sheidric@rz.uni-potsdam.de*>

Main development of: KAppWizard, printing functionality

Ralf Nolden <*Ralf.Nolden@post.rwth-aachen.de*>

Main development of: KDevelop<->Dialog Editor interface, configuration functionality, online-help and handbooks

Jonas Nordin <*jonas.nordin@cenacle.se*>

Main development of: Classviewer and -parser

Pascal Krahmer <*pascal@beast.de*>

Main development of: Dialog Editor

Bernd Gehrmann <*bernd@physik.hu-berlin.de*>

Main development of: Grep Dialog, CVS integration

Stefan Bartel <*bartel@rz.uni-potsdam.de*>

Main development of: Real-File-Viewer

Translation coordination:

Martin Piskernig <*martin.piskernig@stuwo.at*>

Program and Documentation Translations:

Martin Spirk <*spirk@kla.pvt.cz*> -Czech

Steen Rabol <*rabol@get2net.dk*> -Danish

Martin Piskernig <*martin.piskernig@stuwo.at*> -German

Salvador Gimeno <*salgiza@eui.upv.es*> -Spanish

Sami Kuhmonen <*sami@iqs.fi*> -Finnish

Herve Lefebvre <*hlefebvre@easynet.fr*> -French

Pahan Szabolcs <*szabczy@bigfoot.com*> -Hungarian

Duarte Loreto <*dnloreto@esoterica.pt*> -Portuguese

Ilmar Habibulin <*ilmar@ints.ru*> -Russian

Jacek Wojdel <*wojdel@kbs.twi.tudelft.nl*> -Polish

Jan Prokop <*jprokop@ibl.sk*> -Slovak

Patrik Adolfsson <*patrik.adolfsson@iname.com*> -Swedish

Startlogo:

Jacek Wojdel <*wojdel@kbs.twi.tudelft.nl*>

Additions, patches and bugfixes:

Walter Tasin <*tasin@e-technik.fh-muenchen.de*>

Jost Schenk <*Jost@Schenk.de*>

David Barth <*dbarth@videotron.ca*>

Matthias Hipp <*Matthias.Hipp@gmx.de*>

Martin Piskernig <*martin.piskernig@stuwo.at*>

Matthias Hoelzer-Kluepfel <*mh@caldera.de*>

Steen Rabol <*rabol@get2net.dk*>

Matt Koss <*koss@napri.sk*>

Jochen Wilhelmy <*digisnap@cs.tu-berlin.de*>

Bernd Gehrmann <*bernd@physik.hu-berlin.de*>

Torsten Uhlmann <*TUhlmann@debis.com*>

KDevelop contains sourcecode from the following applications:

KWrite 0.98 © by Jochen Wilhelmy <*digisnap@cs.tu-berlin.de*>

KDE Help © by Martin R. Jones <*mjones@kde.org*>

GrapeFruit © 1999 by Bernd Gehrmann <*bernd@physik.hu-berlin.de*>

KSwallow © by Matthias Hoelzer <*hoelzer@physik.uni-wuerzburg.de*>

kcmlocale © 1998 by Matthias Hoelzer <*hoelzer@physik.uni-wuerzburg.de*>

Chapter 15

Thanks

We like to express special thanks to all of our family members and friends who supported us in several ways to let us construct and improve KDevelop.

Further, we thank Jochen Wilhelmy for offering his program kwrite and his help on integrating it into KDevelop.

Thanks also to the KDE team and Stephan Kulow, who gave us the possibility to work on KDevelop via CVS

We hope that our free work on this product will lead to a better acceptance of Free Software and its development. Without the help and idealism of many, the KDevelop IDE would never have been realized in such a short period of time and we're proud that so many users already have reported good experience and brought KDevelop to it's destiny: helping free software programmers to build a better world where users have the freedom of choice.

Chapter 16

Copyright

Index