A C/C++ development environment for UNIX

EXCITING DEVELOPMENTS

RALF NOLDEN

For some time now KDevelop has been the standard development environment under Linux and Unix for writing portable KDE/Qt programs in C++. This article shows that graphical IDEs can make programming much easier under Unix too – and not only when it comes to programming applications with graphical user interfaces.

KDE has become popular on a great many Linux and Unix systems, not least due to its usability and quality. The number of KDE applications that are available which aim to make the user's life easier is rising all the time. Most KDE users don't bother to ask how these applications come about or why KDE is so popular with programmers (if it wasn't, there wouldn't be this huge number of programs.) However, there are some very good reasons for it.

The easy-to-learn API of the KDE and Qt libraries and the portability of KDE/Qt programs to most Unix platforms are two of the main reasons for the popularity of KDE. Qt even allows developers to compile their programs for MS operating systems. Added to this, KDE provides the relevant development tools that make this fast growth possible in the first place. One of the most important of these tools is KDevelop, an **integrated development environment (IDE)** which has not only established a name for itself among KDE developers but is increasingly being used for other programming projects too. KDevelop has provided the developer with a comfortable working environment, from which all KDE users will eventually benefit.

Objectives of KDevelop

The KDevelop team, which, like many other open source projects, consists of individual international programmers who participate in the project on a completely voluntary basis, is currently financed entirely by the developers themselves. This includes the cost of websites and development hardware. Contributions are welcome, so please get in touch!

Of course, each member of the team has his own, entirely personal reasons for participating in such a project. An important objective for some is to create applications that encourage a large number of ordinary users to become enthused about Linux/Unix. However, for many the aim is to make life easier for developers in particular. On the one hand

there are the developers who have discovered Linux as a development platform for free projects, not least thanks to KDevelop, and who wish to give something back to the Linux community. On the other hand, there are those who believe a high quality development environment is needed in order to convince programmers from the business world that it isn't just cool to program for Linux but that it is really simple and user-friendly too. Finally, of course, there is the purely selfish motivation that by creating such a fine tool as KDevelop the team make it easier and quicker to develop other programs themselves.

The more developers who are encouraged to develop a particular tool or start on a major project, the larger the number of available applications becomes over time. The old adage that there is no application software for Unix has finally been laid to rest. With KDE 2.0 approaching, we are currently in one of the most exciting free software development phases for the desktop environment, and everyone can be part of this by making a contribution.

KDevelop aims to provide the broadest possible support for compilers and platforms. This is actually becoming very easy as the use of autoconf/automake-compatible project frameworks and the universally available GNU tools always ensure that, firstly, KDevelop runs on most Unix systems (at least on those on which KDE also runs, i.e. Linux, BSD, SCO UnixWare, HP-UX, AIX,..) and, secondly, that the applications developed using it can be compiled and run on all these systems.

With this in mind, the fact that KDE is at the top of our list of priorities should be self-explanatory. The KDevelop environment provides complete support for the development cycle of KDE applications. From the generation of a basic KDE program through the creation of a graphical interface to the production of documentation, localisation and packaging, the IDE covers almost every need. KDevelop is now developed using KDevelop itself. With around 80,000 lines of source code this provides a convincing demonstration of the power of KDevelop.

Installation

We will now provide a few tips on installation so that you can get started with KDevelop later on in this article. Users running SuSE 6.4 or SCO UnixWare will probably experience the fewest difficulties. These distributions already contain version 1.1., which is similar to the latest version and can be used for development immediately without any problems. If you do not have KDevelop on your distribution or would like to use the latest version, you can download the source code, and in many instances precompiled RPMs too, from the KDevelop home page. The website also contains up-to-date information on the state of KDevelop, PostScript versions of the complete documentation, the Webforum and addresses of developers and mailing lists.

IDE (Integrated Development Environment)

Integrated development environments (IDEs) *combine under one interface all the tools required by a programmer such as compilers, linkers, debuggers and editors. On top of this, they make the developer's work as easy as possible by automating repetitive stages of the development cycle. They take on tasks such as project management and the generation of makefiles, and provide help in searching for errors. IDEs allow programmers to develop their programs as rapidly as possible, reducing the "time to market" compared to those using conventional development tools.*

IDEs really come into their own when you are developing applications that have a graphical user interface. Programming a graphical interface by manual coding is time-consuming and tedious. IDEs allow the interface to be designed visually, and then automatically generate the required code.

If you compile KDevelop yourself you will ensure at the same time that you have as few problems as possible later on when you generate your own applications. Usually, all you need to do if you encounter a problem is to install missing packages such as library and include files.

What can you develop?

You can use KDevelop for any task to do with C and C++. Although KDevelop is specifically designed for KDE programming, there are a number of developers who use KDevelop for other projects. You might be interested to know, for example, that the inte-

Fig. 1: Selecting the project type using the KAppWizard

grated class parser doesn't have any problems with the Linux kernel code!

Development using KDevelop always begins by generating a project. To make this easy a "Wizard" is available. This is launched using the Project, New menu. On the first page of the dialog you select the type of framework, or program template, to be generated. Programmers have at their disposal a total of 13 different types which immediately create executable applications and can be edited in KDevelop after they have been generated.

The project types are divided into several groups:

- **KDE applications:** This group contains all the skeleton programs available for development for both KDE 1.x and the forthcoming KDE 2.0. Possibilities include a mini-application that consists only of a main window, an SDI (Single Document Interface) framework for a standard application with menu panel, tool bar and status bar (based on a document view model, as is common in GUI application development) and, a real gem, a MDI (Multiple Document Interface) framework for KDE 2.0, with which users can create Windows-style programs that manage several documents and their views simultaneously in one main window.
- **Qt applications:** The same skeleton programs are available as for KDE, the difference being that they are based exclusively on the Qt library. The use of the pure Qt API allows developers to implement applications that can also be compiled and used under Microsoft Windows because Qt, as a cross-platform toolkit, supports not only Unix but (in the "pro" version) the "other" operating system too. An example of this is the circuit board layout program Eagle, the next version of which is currently being developed with Qt in order to guarantee availability on all platforms as efficiently as possible.
- **GNOME applications:** To the amazement of the GNOME developer community and contrary to all the prophecies of doom, KDE offers its hottest competition the opportunity to profit from the advantages of an IDE too. KDevelop provides a framework that is based on the gtk+ library and can be used as a fully adequate framework for creating a GNOME application. Therefore, anyone who, for whatever reason, cannot acquire a taste for KDE as a Linux desktop does not have to forego the comfort of KDevelop if they wish to develop for GNOME.
- **Terminal applications:** Even if you only wish to write a command line tool you will be in good hands with KDevelop. For C and C++ there are frameworks with which minimal "Hello World" programs can be generated and then extended as you wish. These frameworks also offer beginners an easy way to take their first steps into programming under Unix.
- **Other:** This is how KDevelop defines its own project. For those of you who have already developed an application and would now like to edit it using KDevelop, this option provides a framework that allows you to convert to KDevelop without any major migration problems. There are already a number of projects that have achieved this quite successfully.

On the next page of the Wizard we define the project-specific options such as the program name, version, directory, author and so on, and what we want the Wizard to generate, from a complete framework down to the "blank" *autoconf/automake* layer.

On the third page you are offered the opportunity to develop the project using the version control system CVS. However, this is only possible locally when generating a new project. Users who wish to import their source code from a CVS server can do this with the help of the Cervesia program provided by co-developer Bernd Gehrmann, and then activate CVS support in KDevelop. CVS is used as standard on free projects in which more than one developer is involved. Otherwise it would not be possible to develop these projects independently at the same time and based on the same source code. However, the use of a CVS system can also be a sensible option for lone developers in certain situations. Any-

GNU tools, configure and make

In autoconf/automake-compatible software packages that are distributed as source code, a project is usually generated using the following instructions:

```
% make -f Makefile.dist
```

The command make, retrieves the instructions in the file Makefile.dist, usually the GNU tools automake, autoheader and automake. It uses them to generate from Makefile.ams the Makefile.ins; and from the configure.in file and the autconf macros the configure script. In the case of KDE/Qt projects, the perl script automoc or am_edit inserts the calls for the Qt-MOC (Meta-Object Compiler for the C++ signal/slot extension) into the Makefile.ins.

```
% ./configure
```

This executes the configure script, which checks the system to see that the paths for include files, compilers, linkers and libraries needed are present so that the generation of the project is guaranteed. It then generates the Makefiles from the Makefile.in files.

```
% make
```

The command make executes the instructions in the Makefiles generated by configure. In other words, it retrieves the MOC compiler, the C/C++ compiler and the linker in order to compile the source codes and link them to create a program or library.

```
% make install
```

This command executes the instructions under the "install" tag of the Makefiles. The effect should be that the programs, header files, libraries and documentation etc. are installed on the system (For this to work, you should be root.)

one who has already experienced the distress of accidentally destroying their project knows how valuable a CVS version can be as it can restore the program within seconds. The usual CVS functions such as add, delete, update and check are available later on in the file tree of the KDevelop project directory.

The next two Wizard dialog pages define the file headers which insert the date, author and licence information into the source text so that you can be identified as the author of the source at any time and your rights are covered. When you get to the last page, a click on the "Generate" button is all that is needed: the rest is trivial (or magic depending on how you view it!). As soon as the generation has been completed, you can exit the KAppWizard using the "Exit" button. The new project is then loaded automatically and ready for editing from in the source code editor. Although it won't do very much you can compile and run the program to check that everything works.

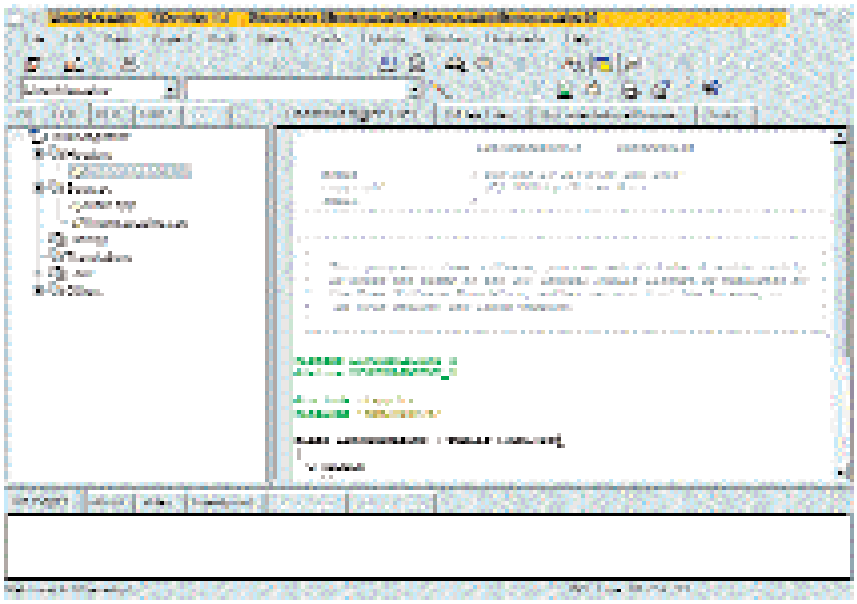
The Linux magazine browser

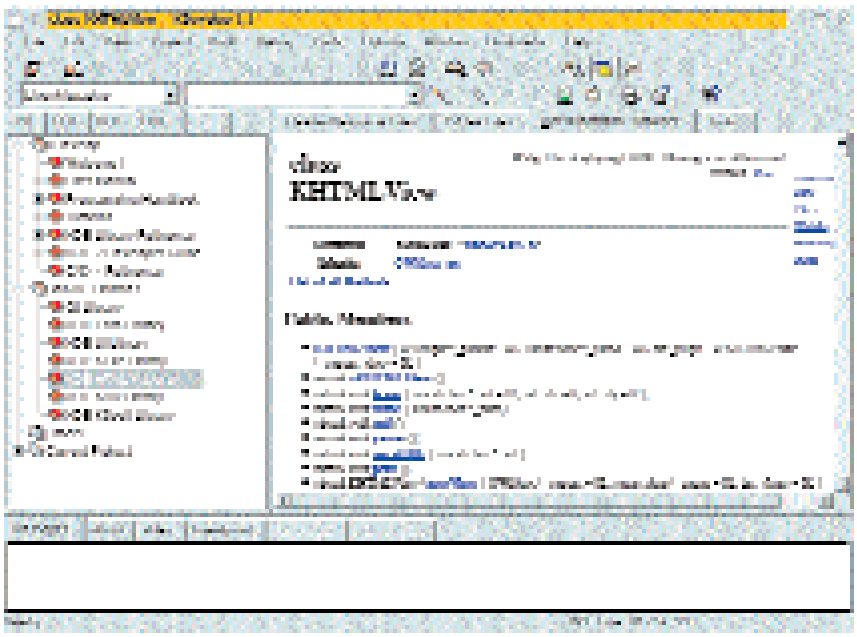
KDevelop has too many features to cover them all in the space of one short article. The best approach is to demonstrate the typical procedure for creating an application using a small example. The example program is rather nice – it is a web browser: one that is limited to viewing one particular web page, without any graphics, but a web browser nevertheless. This example will introduce you to the world of KDE programming without very much prior knowledge. Our program will be developed and extended step by step with the emphasis on how to use KDevelop while editing source code. We will also take a closer look at the concept of the Qt signal/slot mechanism, which makes GUI programming decidedly easier.

First of all, start KDevelop either via the "K" menu or by entering "kdevelop" in a terminal window. If you are starting KDevelop for the first time you will be guided through the basic configuration of the IDE by a setup assistant. You will immediately see which programs or packages still have to be installed. If you are unsure, KDevelop always offers you context sensitive help and, in the assistant, a "help" button using which you can move immediately to the description of the installation procedure in the manual.

To create the example program use the KAppWizard in KDevelop. Select "KDE-Mini" as the application type and enter "LinuxMagazine" as the project name. Fill in the other fields with your own details. After the program framework has been generated, KDevelop presents you with the example program. (Note: if you see any errors during the generation stage, you will need to install the missing packages indicated by the errors, delete the partly-generated project and try again.)

You can now compile the program to test whether everything is working as it should. To do this, use the "execute" button on the tool bar, represented by a small gear-wheel like the one on the "K" menu. KDevelop then retrieves *make*, which executes the commands in the makefiles. These files contain instructions for controlling the compiler and linker, enabling you to generate the program correctly. Once the compile procedure is finished, the program is started automatically and you have stepped successfully into a new world of program development! (Again, if the program doesn't compile, the reason is probably because of libraries and include files that are required and which have not been installed.)





[top]
Fig. 2: KDevelop showing our example project

[above]
Fig. 3: The documentation browser: all the information you need at a glance.

Information is all

Now we turn our attention to the finished product. You will soon realise that not only do you need to know what your program is supposed to do, but also how to find the information you need to create

the program as quickly as possible. Therefore, information is everything! Without documentation you will not be able to generate a program with KDE/Qt. But KDE would not be KDE if there were not a clever solution to this. With the help of the documentation tool KDoc, a set of HTML documentation (the API documentation) is generated from the header files. KDevelop relieves you of this task too. In the case of SuSE 6.4 and the KDK (KDE Development Kit, provided by the KDevelop team and also available from the KDevelop home page, this is fully precompiled and installed for KDE 1.1.2). What's more, you can generate your own new set of documentation (for the KDE 2.0 API for example) at any time using KDevelop. The documentation provided by TrollTech with the Qt library is integrated automatically.

The next step is to update the KDevelop search index so that you can get access to the documentation and can quickly find the descriptions of particular topics, classes and functions when you need to. You can also choose to use *htdig* or *glimpse* as a search engine. Commercial developers rely on *htdig* as *glimpse* is only freely available for non-commercial usage and is no longer shipped on most distributions.

KDevelop itself contains another five manuals in the form of online help, which should provide you with further help in almost any situation – at least as far as programming is concerned. User manuals, tutorials, programming manuals and KDE references provide you with a solid basis on which to learn about the IDE and how to use it efficiently to create more complex programs. A copy can be ordered in book form from the KDevelop home page.

Tree views...

There are a few navigation tips you should learn if you want to become a proper KDevelop power user right away. In the left-hand section of KDevelop you will find tree views placed on tabs with the following captions:

- CV (ClassViewer): This is the class browser for your project's classes and functions. You can use it to skip to class declaration or function implementation. Using a context menu you can access additional functions such as skip to method declaration, add files, classes, methods and attributes etc.
- LFV (Logical file tree): This is where you can sort your project's files in folders based on their filename extension so that you can access them more quickly. Note: in the KDevelop setup you can set the option Autochange (automatically changes tree view during programming) either to class browser or logical file tree.
- RFV (Real file tree): This is where you can view the project directory as you can in the file manager. It gives you access to all the files. Using a context menu, RFV provides you with extended functions such as delete, add to the project and the CVS commands.
- DOC (Documentation tree): The documentation view offers you access to the online help included with KDevelop, the KDE/Qt documentation and your project's documentation. In the KDE/Qt documentation you can also browse down to the functions of the classes available and find the information you require without a lengthy search.
- VAR (Variable tree): This view is available while applications are being debugged. It indicates the status of the runtime variables you use.

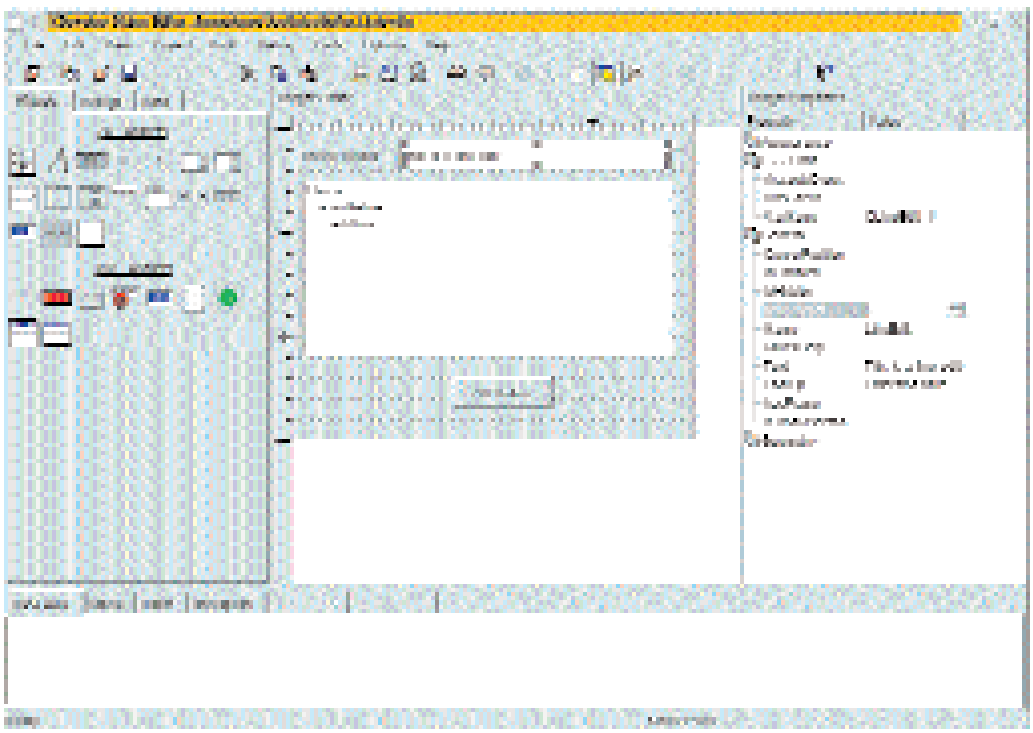
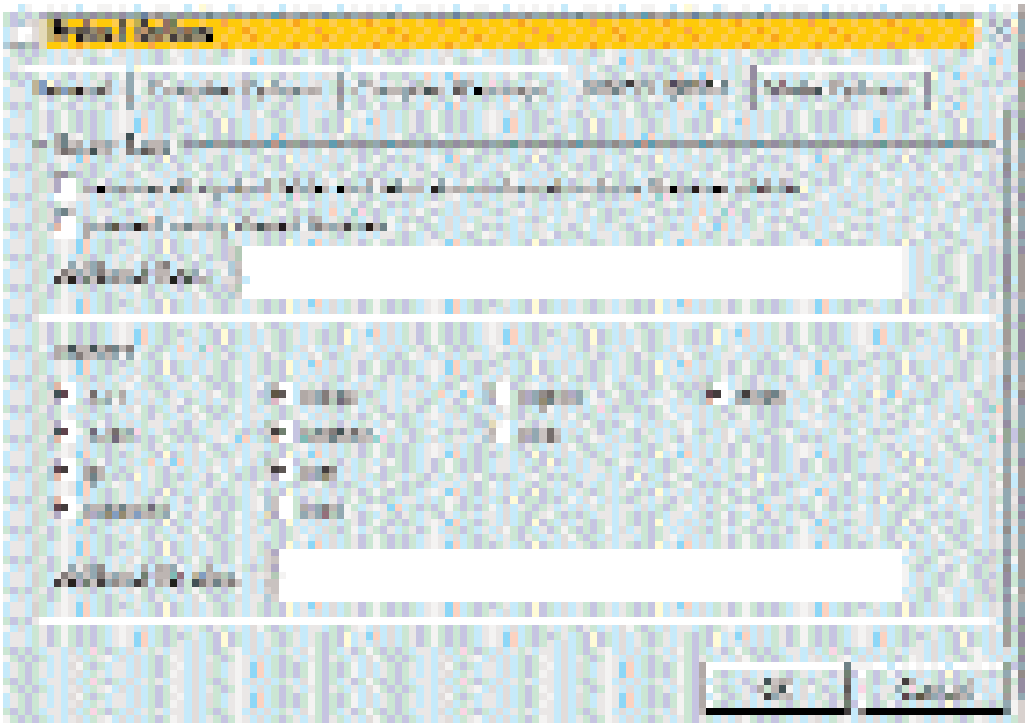
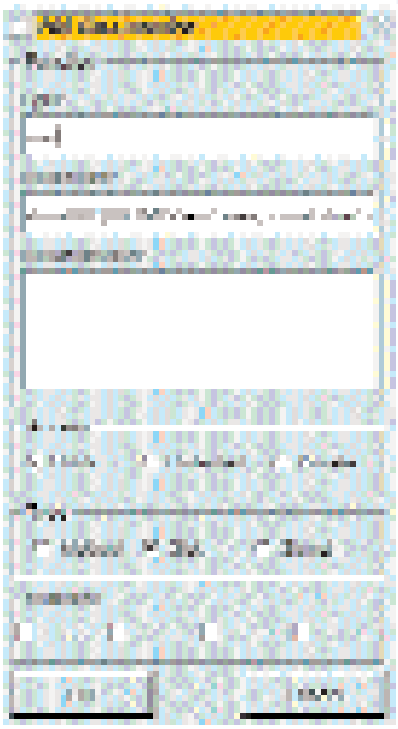


Fig. 4: The KDevelop dialog editor





[left]
Fig. 5: The Project Options dialog box

[right]
Fig. 6: Adding a member function

...and output views

Below the tree views are the output views. These are divided into windows:

- Messages: In this window KDevelop shows all the output of external tasks such as make, the compiler etc. If an error message is shown, a click on it takes you to the error (automatic output localisation).
- StdOut: Displays the program output during a debugging run at command line level.
- StdErr: Standard error output of project application during debugging runs.
- Breakpoint: Displays the breakpoint of the debugger and number of hits during a debugging session.
- Frame Stack: Displays the application's assigned frame stack during debugging.
- Disassemble: Assembler output of program code.

Work view

The work view is where you find the editor window for header, resource and other files, the source code editor window, the documentation browser and the tool window in which KDevelop starts any external programs that are required (such as KIconEdit for editing toolbar icons.) From the tool menu you can add other programs for use within KDevelop.

The dialog editor

You can call up KDevelop's dialog editor from the view menu or using the relevant tool bar button. Using it you can design the graphical interfaces of your applications and have them output as C++ source code. You can then edit the classes produced in the source code editor with the help of the class view. Currently the dialog editor in KDevelop only supports Qt 1.4x and KDE 1.1.2 APIs. If you only use standard components, however, you should not have any problem editing KDE 2.0 projects.

Implementing the browser

So to the creation of our example application.The Linux Magazine browser should be a simple HTML browser which, when started, loads the Linux Magazine home page at *http://www.linux-magazine.co.uk/* and displays it. First, the derivation of the main widget must be changed from QWidget to KHTMLView, and the *khtmlw* and *kfm* libraries must be linked to the program. Replace the derivation in the declaration of class LinuxMagazine and in the function stack of the relevant constructor. You must also change the include file from *#include <qwidget.h>* to *#include <htmlview.h>*.

Next, open the project options via the "Project" menu and switch to the "Linker options" tab. Enable the "khtmlw" and "kfm" checkboxes. After the dialog has been closed by clicking "OK" the *configure* script is automatically run so that the makefiles can be newly generated. Now allow the program to compile from scratch. When everything is running we can start implementing the actual functions. We implement a new function which downloads and displays an HTML page. At the same time we declare this as a "slot" so that we can continue to browse from this page by clicking on hyperlinks.

The HTML widget provides a signal, which we can link with this slot. The only things to note are the parameters of the signal: they must match the slot. Let's take a closer look at the API of the class KHTMLView (you will find these in the documenta-

Info

Kdevelop homepage:
http://www.kdevelop.org

KDE homepage:
http://www.kde.org

Homepage for KDE developers:
http://developer.kde.org

TrollTech AS (Qt library):
http://www.trolltech.com

tion browser in the *khtmlw* library). You'll see that a *URLSelected()* signal is available there. If the user of the program selects a link with the mouse, this signal supplies us with the URL. To generate the new function, select the "add method" function via the class LinuxMagazine in the class browser. When the dialog appears enter *void* as a return value and *showURL(KHTMLView* view, const char* url, int, const char*)* as a method name. The class browser adds the final semicolon automatically. Some description explaining what the function does won't hurt either.

Finally, we select "public" and "slot" as a modifier. OK the dialog box and you will be placed in the code implementing the new function. We must now consider how we wish to load the page. As KFM fortunately already provides this function, we can simply use it. For brevity's sake, only the general procedure is described. KFM loads the HTML page for us into a temporary file which we open using "QFile" and read into a string using "QTextStream". We then execute the view functions of the HTML widget using this string and remove the temporary file.

Finally, we have to execute this function using the URL of the Linux Magazine home page, which we execute in the constructor. We are not bothered by the fact that the method is also a slot as slots can be used as normal functions, the only difference being that a signal can also be used to execute them. Lastly, we insert a connect in the constructor, which links the *URLSelected()* signal to our slot *showURL()*.

You will find the source code of our example in the following listings. We hope that the brief example has given you your first insight into KDevelop and KDE programming: a taster before your first program of your own. Another small tip to finish on: If you are looking for ideas for your own applications, take a look at the source codes of other programs, which can be downloaded from the KDE home page. If your program will also be released under the GPL you can, of course, re-use this source code directly: you don't need to re-invent the wheel.

You can now start your first KDE program. If you like, you can install it on your system by opening a console, switching into the project directory and entering the command *make* followed by a *su -c "make install"*. After you have restarted the K-Panel you can access the Linux Magazine browser from the K menu.

Debugging included

Another technical highlight of KDevelop is the internal debugger. This can be seen in action in figure 7. Tool bar buttons to the top right of the window let you execute the program being tested a line at a time, stepping into or over function calls and so on. These buttons can be used as a floating tool bar, giving the advantage that you don't have to constantly switch between the IDE and the program window when debugging GUI applications. The program window remains in the foreground even if you debug step by step through the source code.

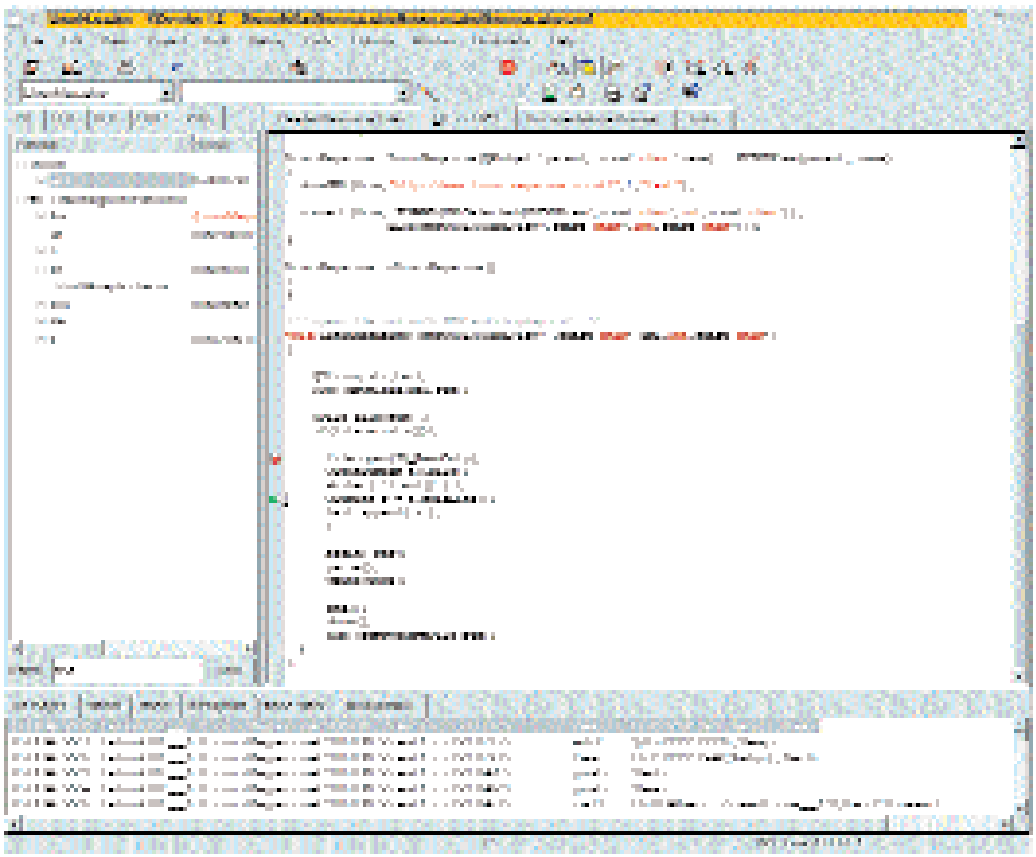


Fig. 7: Debugging the Linux Magazine browser with KDevelop

In the tree view the status of variables and the function stack can be observed in the VAR window. In the output window you can find windows for observing breakpoint hits and stack frames. The program can also be executed at machine instruction level using the disassembly window. Information about the contents of memory and processor registers can be obtained using an additional view in which the status of the libraries linked to the application can also be viewed.

For advanced users, KDevelop offers the option to set breakpoints in library calls via what are known as "Pending Breakpoints" even if the libraries are not loaded yet.

KDE 2.0 and KDevelop 2.0

Let's finish with a quick look at the near future. With KDE 2.0 approaching, there is going to be some more action in the Linux/UNIX domain, and not only in terms of innovation and speed. The KDevelop team is currently developing the second version of the IDE and is, of course, making substantial use of the new technical opportunities presented by KDE 2.0. This mainly concerns the user interface, which will support an MDI interface in future. The tree and output views can also be separated from the main view and used as self-contained windows. This will particularly please those developers who use XFree 4.0 in Multi-Monitor mode as they will be able to distribute KDevelop to all monitors. Work is also being done on the interchangeability of the editor so that *vim* fans can use their one true love.

The fact that KDevelop 2.0 isn't ready yet shouldn't stop you from developing for KDE 2.0. KDevelop 1.2 supports it already. To make getting started easier, the tutorial supplied contains a KDE 2.0 application which you can try out straight away, so allowing you to keep your finger on the pulse. We wish you great success, and hope to see your program soon on the list on the KDevelop website where all the programs created using the tool are listed. ■

Listing 1: main.cpp

```
#include "linuxmagazine.h"

int main(int argc, char *argv[])
{
  KApplication a(argc, argv, "linuxmagazine");

  LinuxMagazine *linuxmagazine =
    new LinuxMagazine();
  a.setMainWidget(linuxmagazine);
  linuxmagazine->show();

  return a.exec();
}
```

Listing 2: linuxmagazine.h

```
#ifndef LINUXMAGAZINE_H
#define LINUXMAGAZINE_H

#include <kapp.h>
#include <htmlview.h>

class LinuxMagazine : public KHTMLView
{
  Q_OBJECT
  public:
    /* construtor */
    LinuxMagazine(QWidget* parent=0,
        const char *name=0);
    /** destructor */
    ~LinuxMagazine();
  public slots: // Public slots
    /* opens the url with KFM and displays
       it. */
    void showURL(KHTMLView* widget,
        const char* url,int,const char*);

};

#endif
```

Listing 3: linuxmagazine.cpp

```
#include "linuxmagazine.h"
#include <kfm.h>
#include <qfile.h>

LinuxMagazine::LinuxMagazine(
QWidget *parent, const char *name) :
KHTMLView(parent, name)
{
  showURL(this,"http://www.linux-magazine.co?
.uk/",1,"test");

  connect(this, SIGNAL(URLSelected(
    KHTMLView*,const char*,int,const char*)),
    SLOT(showURL(KHTMLView*, const char*,
    int,const char*)));
}

LinuxMagazine::~LinuxMagazine()
{
}

/* opens the url with KFM and displays it. */
void LinuxMagazine::showURL(KHTMLView* ,
const char* url,int,const char*)
{

    QString str,text;
    KFM::download(url,str);

    QFile file(str) ;
    if(file.exists()){

    file.open(IO_ReadOnly);
    QTextStream t(&file);
    while ( !t.eof() ) {
    QString s = t.readLine();
    text.append( s );
    }

    begin( str);
    parse();
    write(text);

    end();
    show();
    KFM::removeTempFile(str);
  }
}
```