Kylix 1.0: Delphi for Linux

DEVELOPMENT-CAPABLE

SEBASTIAN GÜNTHER

Now at last, after being constantly postponed, the first version of Kylix, Borland's Linux porting of the development environment Delphi has reached the UK. The conversion has only partly succeeded, so there are lots of problems to spoil the fun of working with what is truly a very powerful development tool.

In autumn 1999, when Borland announced a Linux version of Delphi and later of C++-Builder as well, there was great astonishment, even among the developers in their own company. After all, these products were software, which lives very much by its visual nature and should therefore depend heavily on Windows.

The launch date was initially planned for one year after the announcement, but this was exceeded by about six months in view of the high cost of development. The US retail version just completed can now show whether the development period was nevertheless sufficient to be able to follow in the successful footsteps of Delphi.

Windows past

The promise made by Borland was to offer the options of Delphi for Linux. In a later version an equivalent to C++-Builder is intended to follow as part of Kylix. But what does Delphi do now? The aids of the Delphi IDE simplify, in all product variants, the easy creation of graphical user interfaces (GUIs). Here control elements are no longer created by a function command in the program code, but the drafting of a window, dialog or form is simplified in a form designer for selecting the control elements from a component palette and the visual determination of position and size by mouse click.

A properties editor, which can be opened at any time as a free-floating window, displays all the properties of the currently-selected components in the form of a table and enables the direct manipulation of these properties by keyboard and mouse; the changes can be seen immediately in the designer.

The more expensive versions of Delphi also offer, purely for designing forms - where ordinary windows and dialog boxes are also referred to as forms - more besides: with the aid of database support, control elements can for example be linked to a field of a data table; the Web support allows the dynamic creation of Web contents, and the support of COM/ DCOM/ActiveX and CORBA under Windows makes it possible to develop real multi-layer database applications.

Linux future

So how does Kylix convert this to Linux? Firstly, the software package will have to be installed; this can be done using the very well-known installation

program from Loki Entertainment Software, either via Gtk-based graphical user guide or by command line. Overall, it performs well: Among other things, it checks at the start whether all the requirements are met. So a kernel from at least the 2.2 series must be used, Glibc from version 2.1.2 and Libjpeg from 6.2 are absolutely essential.

A full installation takes up about 200MB on the hard disk, and this can only be reduced noticeably by doing without the online documentation, which would roughly halve it.

Old wine in new bottles

The long load times when you start the integrated development environment are an early clue: The IDE is not so much a newly-developed Linux application, it's just that with the aid of the Wine library the old, familiar Delphi IDE has been ported onto Linux.

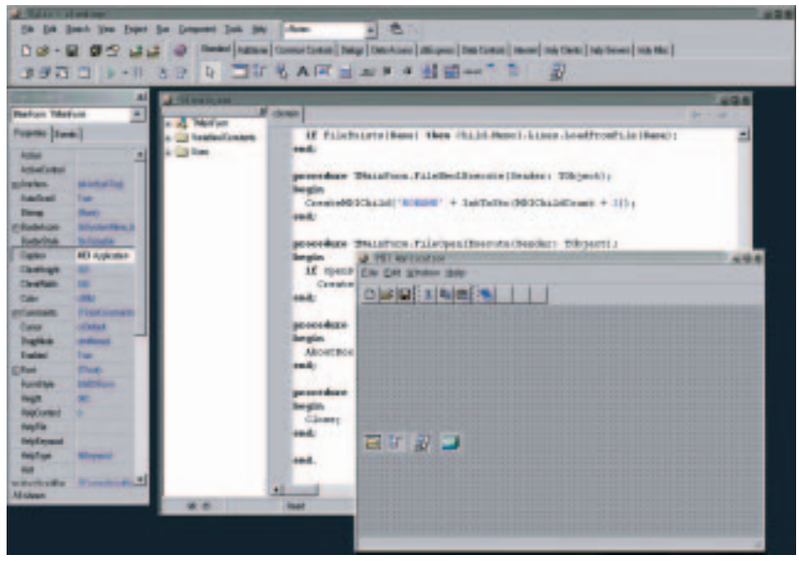
Wine, the imitation Windows programmer interface for Unix-type systems, does allow a very rapid conversion of Windows software onto (among others) Linux, but it does bring a number of considerable disadvantages with it: Long load times, high memory consumption, slow initiation and sluggish reaction by the graphical user interface, and font problems with many X11 installations. If possible, therefore, it is advisable to use TrueType fonts from an original Windows installation.

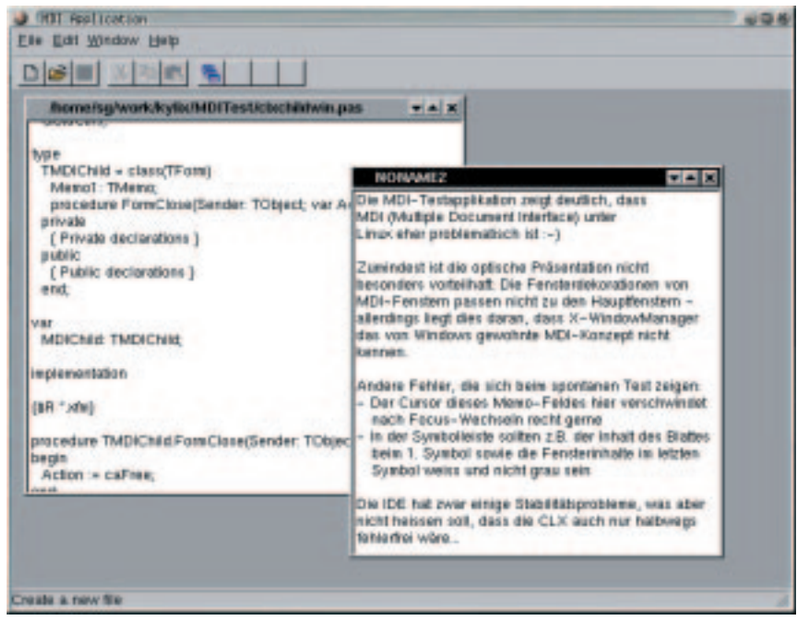
But whether Wine is also responsible for the over-frequent crashes of the IDE, is a question nobody can answer.

After the start, in any case, four windows appear for the user, which float on the desktop: The command centre is in the form of a long narrow window at the top edge of the screen. It contains the menu bar, symbol bars and the component bar. Also opened: a form in the design mode, a source text editor and the Object Inspector. All registered classes of components, spread over several pages, are shown in the component bar in symbol form.

This bar is important in connection with a designer, a sort of form designer: A component, for example a simple button, is selected from the corresponding category by a mouse click. Another mouse click in the form view inserts a new component of the type selected at the site of the click. Each component can be moved later by mouse; it is also easy to change the size directly.

By clicking on a component in the designer, this component is selected; the Object Inspector always represents the corresponding properties and event handling routines of the currently-selected component. The Inspector window always presents them in two columns: The first contains the names of the properties, the second the corresponding property values. A click on a value makes it editable. Unfortunately Kylix can only show properties as pure text, but a graphical representation of certain types of property such as colour values would surely be more user friendly.





[top]
The IDE after creating a simple MDI application: As well as the command centre and an editor window, a form editor and the property editor can also be seen

[above]
The automatically-created basic framework of the MDI application is in fact ready to run. But even here the first errors crop up

Four sections of the component bar are devoted to the purely visual components. Behind this is hidden, basically, all the types of control element already familiar from Windows: buttons, menus, symbol bars, but also complete dialog boxes for things such as file selection. Three additional sections serve as database support: Special database-capable variants of the normal control elements can be connected to a data source component.

Flexible database support with dbExpress

This data source forms the link between control elements and data set components: for example, during development of the application, should it ever become necessary to change from an SQL table to an SQL stored procedure, to do this it is only

necessary to specify the data source of a new data set component - all control elements connected with the data source then automatically access the new mechanism.

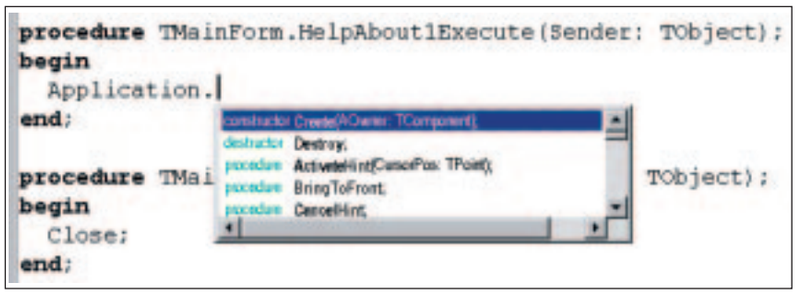
Kylix offers several alternatives as data set components, but there have been some major changes in comparison with Delphi under Windows in this area: In place of the old BDE (Borland Database Engine) and MIDAS there are now a good half dozen new components named dbExpress. They use their internal, special Kylix database drivers to execute the new commands. Drivers for the freely available databases MySQL and Borland InterBase are delivered from the factory as well as for the well-known commercial products IBM DB2 and Oracle 8i.

Friends of older database systems seem at first to have been left out in the cold, because unlike Delphi, so far under Kylix there is no support for a general interface standard like ODBC or ADO, although this would be perfectly possible with UnixODBC, as for example StarOffice demonstrates. This gap will surely be closed very quickly by interested third-party suppliers.

As data set sources, dbExpress offers the usual database objects: Direct read and write access to tables, the result data set volume of a stored procedure or the result of a manually coded SQL query. But a data source cannot be linked to a table to create a master-detail relationship: The table displays all the data sets which correspond in a field to the current value of a selected field of the master data source.

A classic example of this: A master table contains customer data, while a calculation table acts as a detail table. The customer number of a calculation is linked with the customer number of the customer data table. The master-detail relationship ensures that the detail data set always represents exactly the calculations of the currently selected customer.

Also of interest are the so-called Client Data Sets. These enable the use of a simple database in the memory, swapping into a file on the user computer, but also complex mobile solutions, in which a client does not always have access to the database server in a network.



When editing, built-in programming aids such as code-completion are extremely useful. Here you can see what happens if, after entering the point, you hesitate: Kylix displays which elements the global application object possesses. You can now select a method, such as MessageBox, from the list

Rapid development for network and Internet

The last big area of the component bar concerns the development of network or Internet applications: As well as components which encapsulate the TCP/IP or UDP/IP sockets, the Web-dispatcher distributes HTTP queries to various data-producing components called producers. These HTTP queries can be differentiated according to the type of command (such as *get*, *head*, *post*, *put*) and the address (URI).

Since there are also components which exist as producers that can create HTML pages or tables automatically from a database, whole Web servers can be created using Kylix. But equally, it is also possible to create just one CGI application or an expansion module for the Apache server.

For further reaching Internet and network support Borland supplies the new Linux version of the well-known open source component library Indy (formerly known by the name of Winshoes) along with Kylix. This allows access to practically all relevant Internet protocols: TCP/IP, UDP/IP, daytime and time servers, DNS, Echo, finger, FTP and TFTP, Gopher, HTTP, ICMP, POP3, NNTP, QOTD, SMTP, SNTP, Telnet and WhoIs. Raw Sockets for communication under TCP or UDP are also supported by Indy.

Servers for corresponding protocols can also be realised easily, those supported being TCP/IP, UDP/IP, Chargen, daytime and time servers, DICT, Discard Protocol, Echo, finger, Gopher, Hostname, HTTP, IMAP4, IRC, Portmapping, NNTP, QOTD, Telnet, TFTP, IP Tunneling and the WhoIs service. 21 additional components also provide help functions, such as encoders or decoders for important codings like Base64 or UUEncode.

Development means more than just clicking

A relatively large part of applications development does consist of clicking together existing components into data modules or forms and placing the corresponding properties in the object inspector. A great many assistants and special component editors continue to support the creation of complex applications. But at some point, glue code will have to be written, to bond, hold together or expand the structure.

This is where the powerful source text editor and the object Pascal compiler come into play. For every object that can be created in the large designers (such as for forms or data modules), a unit is created automatically. In Pascal, larger applications are not simply distributed over several source text files, but a clear distinction is made between the main program and the add-on modules – the units. Each unit can be independently compiled and integrated into various applications at

the speed of light. The split into main module and units is, by the way, the main reason for the generally very short compile time of these sorts of Pascal compilers, as here it really is only the parts of an application which have actually changed that have to be recompiled.

Editing at a higher level

The IDE creates the basic framework for units themselves, where a new class of the respective basic class is derived for each form or data module. The components used here now all reappear as fields within the new class. The programmer can even save a bit more typing work: The code developed most often will be one that is intended to respond to specific events. The object inspector, though, as already hinted at, lists not only the properties of a component, but also all possible events. A simple double-click on one such event entry causes the IDE automatically to insert an event-handling routine in the source text of the corresponding unit. This now only needs to be filled with code.

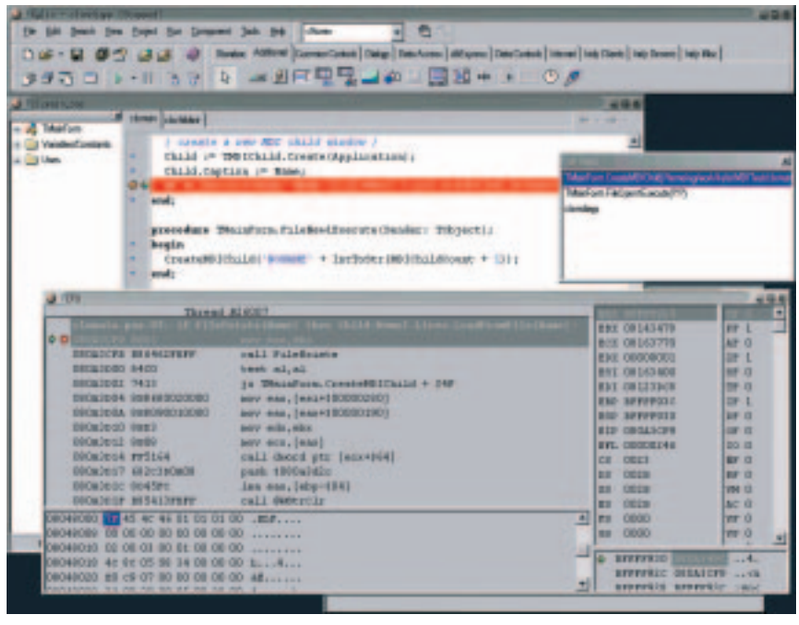
And this is a real delight with the easy-to-use editor: Because, as befits a development environment from the superior class, it offers more than just syntax-highlighting and adaptable keyboard layouts. If, for example, a certain keyword is typed in and then Ctrl+J is pressed, the editor recognises this as a copy command and replaces the keyword with a more complex expression. Thus an *forb* plus Ctrl+J turns into a complete block *for ...:=... to ... do begin ... end*.

Borland combines other programming aids under the name Code Insight, all based on an evaluation of the source texts during editing. Code-completion becomes active after a short pause after entering a point or pressing Ctrl+Enter. It shows a selection list of all the appropriate continuations at the current cursor position – for example after the name of an object variable and a following point the list shows all properties and methods for this object. The editor recognises the type of variable from its previous declaration.

If a procedure or a function is now called up and if the parameter list is to be entered, here again the IDE helps: A brief hesitation during input leads to a display of the declared parameter list. There is now no need to guess or look it up in the documentation to find the correct parameter. And even during actual programming the IDE provides a built-in symbol browser, the Code Explorer, which can display the structure of a module in real time.

Easy debugging

During troubleshooting via the integrated debugger, the ToolTip support is useful as it is familiar from other development environments under Windows. If the mouse pointer in the editor stops over a symbol name, the value of this symbol is calculated and displayed in a ToolTip. So in many cases it is no longer necessary to work laboriously over the additionally available expression evaluation or the watch list.



The IDE provides a powerful debugger. Since Kylix uses a real compiler, there is also a display of the CPU register and a disassembler

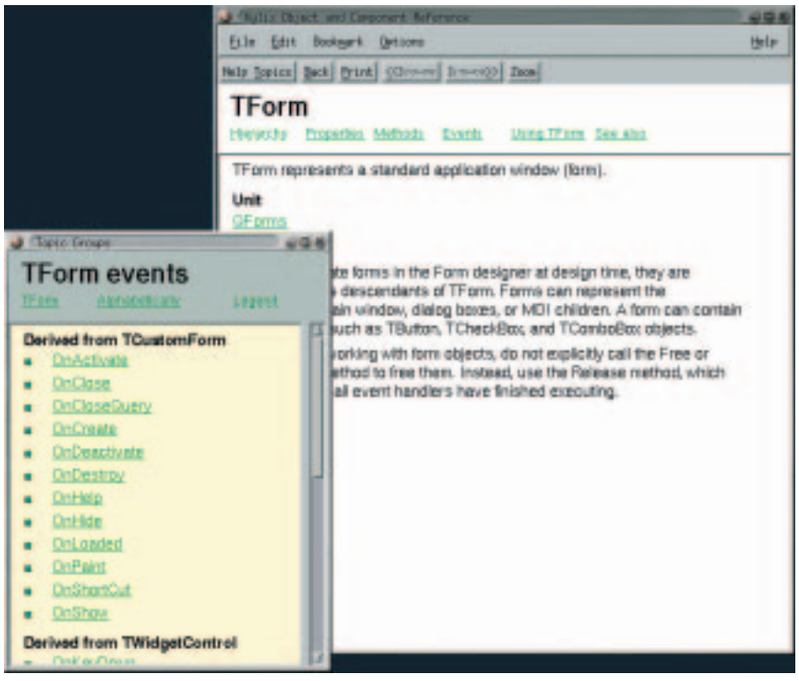
The debugger turns out to be an indispensable tool during application development, it supports practically everything that could be expected of a modern debugger, including an in-built disassembler.

For handling larger projects, which are spread over several applications or modules in the shared object format (*.so*), the IDE has a project manager. It combines all binary modules (files with executable code) into a project group, for which an individual makefile is created. On the other hand, in order to combine a group of components as smaller units into a SO-module, packages can be produced. A package combines several units with components and there is also a comment as to which other packages this package depends on.

This technology makes it easy to use larger components from several applications in combination via a SO-file. The components supplied with Kylix are even installed in such packages.

An application created with Kylix finds and loads the necessary package SOs at run time by itself – no registration in the system is necessary. New packages and components though, do have to be registered in the IDE, so that they can be implemented in applications.

The online documentation leaves a mixed impression. Borland has licensed a tool here that makes it possible to show Windows help files under Linux. The documentation thus corresponds, in terms of structure, to that of Delphi. But the descriptions are not error-free or complete and generally the help texts could easily be a bit more comprehensive in many places. Also, many things



The online help is based on the help files familiar from Windows, which have been extended according to the scheme which you may know from Delphi

VisualCLX outside, Qt inside

Borland has developed a component library for Kylix, for use on several operating systems, called CLX (pronounced clicks). It is derived from Delphi's VCL, the Visual Component Library, and also works with Delphi 6 on Windows.

CLX is split into several parts: The BaseCLX contains general classes and routines (for file accesses or loading and storing components for example). This part is relatively independent of the operating system, as it largely relies on the underlying Run-Time Library, or RTL, which itself abstracts most of the functionality.

The same holds true for NetCLX - the network components - and for DataCLX, as this is merely a link between dbExpress and VisualCLX. This ultimately contains all visible components, thus mainly control elements. It rapidly turned out to be a new wrapper for an old acquaintance: TrollTech's Qt-Library, which is also the basis for the popular KDE-Desktop environment.

The sense and nonsense of this decision may be disputable, because Qt is still far more than just a GUI library. When all's said and done, the entire functionality of BaseCLX is also reproduced here one way or another, but, mainly for reasons of compatibility with Delphi, is not used by VisualCLX.

are described only from a very high level of abstraction. Anyone interested in the internal method of working will not find much information.

Visual development environments

In the last few years a new method has been becoming ever more popular: Instead of writing software complete, line by tedious line manually, advanced development packages support or replace this process with a range of visual help programs. These attempt to reduce applications development to combining ready-made components plus a bit of classic code as glue.

For example Microsoft, with Visual Basic, scored a direct commercial hit in this field, whereupon suppliers of countless additional components shot up out of the ground like mushrooms. But other firms too, such as Borland, were developing similar solutions at the same time. Borland was formerly mainly known for two products: the C/C++ compilers (starting with Turbo C) and Borland Pascal, the amalgamation and further development of the classics Turbo Pascal and Turbo Pascal for Windows.

Visual Basic (VB), though, had to combat a number of deficits: For a long time, VB was not really a proper compiler, but the code was interpreted at run-time, which did not exactly have a positive effect on the execution speed. On top of this, the component model used was anything but fast or memory saving. The upshot was that VB applications on the computers of that time turned out to be very large, memory-guzzling and slow. But on the other hand application development was extremely simplified, which in many cases more than compensated for the greater demands on hardware.

Borland read the signs of the times, and so, in a tour de force, Borland Pascal, a representative of the classic method of programming, was expanded into an easy development package for modern, graphical applications.

Two things were necessary for this: Extending the language of Pascal for better support of objects and components - the language variant Object Pascal was created - and a high-powered integrated development environment (IDE) together with special support for component technology for rapid application development (RAD).

The finished package with RAD-IDE finally came out under the name of Delphi and was now available only for Windows, while Borland Pascal also supported DOS. In parallel, a product was created, with C++ Builder, which on the basis of the same component library made it possible to work with the language C++ instead of Pascal.

Object Pascal

The variant of the programming language Pascal, originally created for educational purposes, on which the modern Pascal compilers such as Delphi, Kylix or even Free Pascal are based, was christened by Borland with the name of Object Pascal. In comparison with the classic ANSI-Pascal standard it was mainly expanded by options for object-oriented programming (OOP). Borland was in fact introducing objects in Turbo Pascal 5.5 more than ten years ago, but with Delphi the OO-capabilities were considerably extended.

An example demonstrates some of the new capabilities using a class to generate random numbers:

```
program ClassDemo;

type
 TRandomGenerator = class
 private
   FMaxValue: Integer;
   function GetValue: Integer;
 public
   constructor Create;
   property MaxValue: Integer read ➥
\FMaxValue➥
write FMaxValue;
   property Value: Integer read Get➥
Value;
 end;

constructor TRandomGenerator.Create;
begin
 Randomize;
 MaxValue := 10;
end;

function TRandomGenerator.GetValue➥
: \Integer;
begin
  Result := Random(MaxValue + 1);
end;

var
 RandomGenerator: TRandomGenerator;
 i: Integer;
begin
 // Create random number generator
 RandomGenerator := TRandomGenerato➥
r.\Create;
 try
   RandomGenerator.MaxValue := 99;
WriteLn(`10 Random numbers in the r➥
ange \0..9➥
9:');
   for i := 1 to 10 do
     WriteLn(RandomGenerator.Value);
 finally
   RandomGenerator.Free;
 end;
WriteLn(`Done.');
 end.
```

One of the extensions in Object Pascal is the properties, which exist in addition to the normal methods and object fields (the variables within an object): A property has, like a field, a data type. For a property, though, no code is created, nor is memory space reserved in the object - the property is a virtual construction, which can be applied in the program code almost like a field.

To now give the property a meaning, the programmer states where it receives its value in a read access, or what is to happen to the new value in a write access. In both cases it is possible, separately in each case, to define a field or a method as source or destination. In the example the property MaxValue corresponds precisely to the internal field FMaxValue, while value can only be read - each read access would be identical to calling up the method GetValue.

Properties were mainly created for component-oriented programming, since run-time type information is created for all properties within a published extract (an extended public extract): At run-time the list of all properties in a class can be queried. But even without RTTI the properties have a few advantages: So one could easily extend the sample program so that a write access on value leads to an initialisation of the random number generator at a specified value. In exactly the same way, MaxValue can also be changed later to method access, without the rest of the application having to be changed.

Compared to C++, it is noticeable that objects are always stored on the heap, so they cannot be placed on the stack and automatically constructed and deconstructed. But this is only a disadvantage in a few cases, for example when simple data structures (such as a rectangle structure: x1, y1, x2, y2) are to receive a simple OO-wrapper.

By way of compensation, a few of the complicated C++ peculiarities such as default- and copy-constructors in Object Pascal could be dropped. As a rule, though, that is, with storage on the heap, practically identical procedures are used in the various compilers.

As can be seen from the example, Object Pascal also supports exception handling. The functional method is extremely similar to that of C++ or Java: If an error occurs (exceptional condition), an appropriate exception object with additional information is created. These can now be caught via a try/except block, while a try/finally-block allows the code, which is also executed if an error condition arises within the try-section, to be specified:

```
try
 Anyfunction;
except
 on e: Exception do
   WriteLn(`error: `, e.Message);
end;
```

This mechanism allows, for example, the reliable release of previously reserved areas of memory or objects – as is typical of compilers, Object Pascal uses no automatic memory management.

Borland has also added a few more things to the Pascal language: ANSI-strings increase the maximum length of Pascal character strings from 255 characters and manage copies of these same strings very efficiently (Copy-on-Write). Unicode strings can be stored in a WideString variable. The length of the new dynamic arrays can be defined and altered at run-time, if necessary with checking for correct array-indices (by range checks).

Variables of the variant type store almost any other data type; This type was really mainly introduced to support COM/ ActiveX under Windows, but is also available, slightly limited, in Kylix. In Kylix, a variant cannot, understandably, point to a COM-object.

One innovation which may be controversial: Pointers no longer have to be de-referenced with \^{}, if the source text nevertheless remains unequivocal; this is certainly something to do with the development that languages like Java prefer memory management to disappear completely into the background - and pointers just do not fit into this concept.

Interview with Jason Vokes, Director, Rapid Application Development at Borland

Jason Vokes

Linux Magazine: *What made Borland decide to develop Kylix?*
Jason Vokes: I regard the Linux operating system literally as a golden opportunity to reach new developers. We began with Delphi and ended up with a cross-platform system.
Linux Magazine: *Borland wanted to introduce Kylix a year after the announcement. The deadline was passed by about six months. Why?*
Jason Vokes: *There were two main reasons for this. Firstly, there was no rapid application development environment. Under Windows, we were used to the tools and advantages available there. Here we had to start with rudimentary things such as gcc and gdb. Once our own debugger and our IDE were available, productivity increased. The second point was that the various Linux distributions all behave in different ways. It took longer than planned to complete it.*
Linux Magazine: *Was the porting of the IDE with the aid of Libwine, which has now been done, your second choice?*
Jason Vokes: *We originally only planned Delphi for Linux, but before we really started, we noticed that the market needs more, namely a cross-platform environment. That's why we developed the component library CLX.*
Linux Magazine: *Is it your intention that the Kylix-IDE will one day also run with CLX? Is there a specific deadline?*
Jason Vokes: *The use of Libwine was a time-to-market decision. We wanted to be fast. In future there will of course be a complete CLX-IDE. It is only internally that there are precise deadlines.*
Linux Magazine: *Have the stability problems with the IDE anything to do with Libwine?*
Jason Vokes: *These problems lie primarily with the Linux loader. Our developers have suggested numerous fixes to the Linux community and sometimes also even done them themselves. Many were adopted and are available and some have yet to penetrate the Linux distributions. When they do, stability will improve. This has nothing to do with Libwine.*
Linux Magazine: *When will the no-charge version of Kylix for the development of free software be available?*
Jason Vokes: *By the middle of this year. We are not announcing a specific date at this time, though.*
Linux Magazine: *Borland has made parts of CLX open source. Will there also be other components, too?*
Jason Vokes: *No, there are no plans for that.*

Besides, Qt is a C++ library, which cannot be used directly with the Kylix compiler - a C-wrapper is needed, which repackages all the classes, methods and functions of Qt into normal C-functions. These C-functions can then be imported by a Kylix unit, so that ultimately VisualCLX can use Qt.

This solution was certainly the fastest solution for Borland to get Kylix ready for marketing, but it also means that visual Kylix applications need more memory and are dependent on countless libraries: Starting with Kylix's Qt-Interface-Unit *libqtintf* via Qt2 itself and all sorts of X11 libraries to the C++ run-time library.

So it will be especially interesting to observe how far Kylix applications will operate reasonably under older or future Linux installations. To complicate matters, Borland itself provides no support for the creation of installation programs, as is the case with Install Shield Express in Delphi.

Quo vadis Pascal?

Kylix is not completely free from competition under Linux: Firstly, there is the C- and C++-compiler with ever more powerful IDEs such as KDevelop. Secondly, in the server field the importance of compiler languages is certainly going to continue to fall, when more and more special scripting languages like PHP or highly-specialised visual development environments succeed.

And finally, Borland should also keep an eye on the field of classical programming: Kylix is certainly

suitable for the creation of command line-based tools, too, and the good editor and debugger are a great help in this. Nevertheless, there is stiff competition in this field with the two free projects GNU Pascal and Free Pascal; in fact, the latter provides not only compilers for several operating systems, but overall comes with a considerably broader palette of additional units and C-Header conversions. It is only in the IDE field that Borland, despite the said problems, has a clear advantage.

Prices and licences

Borland is demanding truly beefy prices for Kylix, which are scarcely justified in comparison with the markedly more stable and more complete Delphi: Buying Desktop Developer would cost some £800 - even though this version still lacks the full NetCLX for developing network and Internet applications. This is reserved for the Server Developer edition, which costs twice as much at about £1600.

But Delphi offers considerably more in this price class, for example support for ActiveX (Windows-specific) or the not-insignificant CORBA architecture, which could also be used under Linux without any problem.

And yet Borland has announced that from the summer a version which is free of charge (but not free) - probably from the release for Desktop Developer - will be on offer for the development of free software under the GPL licence for download (or on CD for about £80). CLX received a double licence for this: Borland's commercial No-Nonsense Licence and the GPL. This could have far-reaching consequences: Firstly, it is to be expected that a flood of programs under GPL licence will crash in on Linux. But it remains to be seen whether, in view of the problems mentioned with library dependencies, this will be a curse rather than a blessing. And on the other hand many component developers who want to port their products from Delphi onto Kylix will also have to consider the use of such a double licence if they want to build up a substantial following of users.

Conclusion

Kylix is currently definitely the most comprehensive software for rapid development of applications (RAD) under Linux. It should give the operating system a bit of impetus, because developing applications has never been so simple. Ultimately however, Kylix has to be described as a very hasty porting of the old, familiar Delphi. The commercial version is anything but refined, and a few extra months for error corrections really would have made all the difference.

The main aspects of Delphi can also be found in Kylix, but behind the scenes the first version comes across like a botch job, which becomes apparent through the instabilities of the development environment and some of the errors in the CLX run-time library.

The frequent crashes of the IDE and the CLX bugs are something Borland will eventually get to grips with. It would certainly be very helpful if the IDE was converted from Wine to CLX itself, but VisualCLX still lacks some urgently required capabilities to do this, such as support for dockable windows.

But Borland should not take too long to come up with these improvements, because the free IDEs for C++ are getting better all the time and the teams of GNU Pascal and Free Pascal are not sitting idle. One major problem for Borland could be that the free developer community will not restrict itself to developing components and utilities for the commercial product Kylix.

It will - because of the many Kylix bugs, but also on principle - appreciate a free compiler and a free IDE more. And Borland cannot simply release both, as they are the essential foundation of the company's business. ■

The author

Sebastian G¸nther is technical director of Areca Systems GmbH in Munich, a service provider involved with networking, the Internet and of course, Linux. For aesthetic reasons he is a great fan of the language Pascal for its own sake and especially the modernised variants. His first contact with the Pascal compilers of Borland was with Turbo Pascal 5.5.