BLAISE PASCAL MAGAZINE 80

Object Pascal / Internet / JavaScript / WebAssembly / Pas2Js / Databases
CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Blaise Pascal

No Gui? From Shell to Hell? HellShell!
RAD Studio Roadmap May 2019
The SQLdb Framework under Lazarus
Installing Lazarus on the Mac
REST easy with kbmMW #25 – XML-RPC and JSON-RPC
SmartBinding with kbmMW #1
kbmSQLiteMan v. 1.80 released

BLAISE PASCAL MAGAZINE 80

Object Pascal / Internet / JavaScript / WebAssembly / Pas2Js / Databases
CSS Styles / Progressive Web Apps
Android / IOS / Mac / Windows & Linux

Blaise Pascal

CONTENT

ARTICLES

Publisher: PRO PASCAL FOUNDATION in collaboration © Stichting Ondersteuning Programmeertaal Pascal

Pascal is an imperative and procedural programming language, which Niklaus Wirth designed in 1968–69 and published in 1970, as a small, efficient language intended to encourage good programming practices using structured programming and data structuring. A derivative known as Object Pascal designed for object-oriented programming was developed in 1985. The language name was chosen to honour the Mathematician, Inventor of the first calculator: Blaise Pascal (see top right).

Niklaus Wirth

Contributors

Stephen Ball http://delphiaball.co.uk @DelphiABall		**Peter Bijlsma -Editor** peter @ blaisepascal.eu
Dmitry Boyarintsev dmitry.living @ gmail.com	**Michaël Van Canneyt,** michael @ freepascal.org	**Marco Cantù** www.marcocantu.com marco.cantu @ gmail.com
David Dirkse www.davdata.nl	**Benno Evers** b.evers @ everscustomtechnology.nl	**Bruno Fierens** www.tmssoftware.com bruno.fierens @ tmssoftware.com
Holger Flick holger @ flixments.com		
Primož Gabrijelčič www.primoz @ gabrijelcic.org	**Mattias Gärtner** nc-gaertnma@netcologne.de	**Peter Johnson** http://delphidabbler.com delphidabbler @ gmail.com
Max Kleiner www.softwareschule.ch max @ kleiner.com	**John Kuiper** john_kuiper @ kpnmail.nl	**Wagner R. Landgraf** wagner @ tmssoftware.com
Vsevolod Leonov vsevolod.leonov@mail.ru		**Andrea Magni** www.andreamagni.eu andrea.magni @ gmail.com www.andreamagni.eu/wp
	Paul Nauta PLM Solution Architect CyberNautics paul.nauta @ cybernautics.nl	**Kim Madsen** www.component4developers
Boian Mitov mitov @ mitov.com		**Jeremy North** jeremy.north @ gmail.com
Detlef Overbeek - Editor in Chief www.blaisepascal.eu editor @ blaisepascal.eu	**Howard Page Clark** hdpc @ talktalk.net	**Heiko Rompel** info @ rompelsoft.de
Wim Van Ingen Schenau -Editor wisone @ xs4all.nl	**Peter van der Sman** sman @ prisman.nl	**Rik Smit** rik @ blaisepascal.eu
Bob Swart www.eBob42.com Bob @ eBob42.com	**B.J. Rao** contact @ intricad.com	**Daniele Teti** www.danieleteti.it d.teti @ bittime.it
Anton Vogelaar ajv @ vogelaar-electronics.com	**Robert Welland** support @ objectpascal.org	**Siegfried Zuhr** siegfried @ zuhr.nl

Editor - in - chief
Detlef D. Overbeek, Netherlands Tel.: +31 (0)30 890.66.44 / Mobile: +31 (0)6 21.23.62.68
News and Press Releases email only to **editor@blaisepascal.eu**

Editors
Peter Bijlsma, W. (Wim) van Ingen Schenau, Rik Smit

Correctors
Howard Page-Clark, Peter Bijlsma

Trademarks All trademarks used are acknowledged as the property of their respective owners.
Caveat Whilst we endeavour to ensure that what is published in the magazine is correct, we cannot accept responsibility for any errors or omissions. If you notice something which may be incorrect, please contact the Editor and we will publish a correction where relevant.

Subscriptions (2019 prices)

	Internat. excl. VAT	Internat. incl. 9% VAT	Shipment
Printed Issue ±60 pages	€ 250	€ 261,60	€ 85,00
Electronic Download Issue 60 pages	€ 60	€ 65,40	———
Printed Issue inside Holland (Netherlands) ±60 pages	———	€ 200	€ 40,00

Member and donator of WIKIPEDIA

Subscriptions can be taken out online at **www.blaisepascal.eu** or by written order, or by sending an email to **office@blaisepascal.eu**
Subscriptions can start at any date. All issues published in the calendar year of the subscription will be sent as well.
Subscriptions run 365 days. Subscriptions will not be prolonged without notice. Receipt of payment will be sent by email.
Subscriptions can be paid by sending the payment to:
ABN AMRO Bank Account no. 44 19 60 863 or by credit card or Paypal
Name: Pro Pascal Foundation-Foundation for Supporting the Pascal Programming Language (Stichting Ondersteuning Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Programmeertaal Pascal)
Subscription department
Edelstenenbaan 21 / 3402 XA IJsselstein, The Netherlands
Mobile: + 31 (0) 6 21.23.62.68 **office@blaisepascal.eu**

Its interesting to see how our community evolves and what kind questions are raised.

One of the most intriguing is of course what is the future of Delphi. Having asked this question many times, here it was al of the sudden. Embarcadero, by the name of our well-known Italian Delphi Master Marco Cantu came up with the latest new plans for the Future.
I thought it was so important that we would publish them so all of our readers would be able to take a good look at it.

Conclusions?
Difficult to make yet. Even though there are no real big new items to find, all necessary stuff. Some great news for Firemonkey lovers: support for iOs and Android and Android native controls. Mac iOs 64 bit support and maybe in 2020 (under consideration) support for Raspberry Pi windows ARM.

For the long term: later this year the 10.4 version - probably September or so.
I see nor real improvements that we all had hoped for: something like the IOT support ore internet support at all. The new roadmap of Delphi you can find it at page: 11.

I am glad that there is a company like TMS that is developing the WEB Core. Updated to 1.2 just now and we will publish the first tests in the next issue. It promises great improvements...

Lots of news about Pas2JS and Lazarus.
I finally found the time to do something that has been asked for many times: an article installing Lazarus under Mac.

We will show a number of applications ready made for Lazarus as well Delphi, while we try do things that are also interesting for starter's and experienced developers. The Puzzle suddenly seem to have quite a lot of problems so we need to solve that first. But there is more we will show some spcialties which are embedde in an application: free for use ofcourse.

Lazarus is coming very close towards the abilities of Delphi. We are working on the very last hurdles.
At the moment we are implementing in FPC (the compiler for Lazarus) three main Items that have been sponsored by a number of companies and the community: Anonymous Functions, Attributes and RTTI.

Generics for Pas2JS are also started by Mattias Gaertner and we will try to get them done asap.

We need funding for that!
If you want to sponsor us - please let us know.
We will come up with some special actions for this.
Goals for you: a guaranteed free IDE for Pascal, capable of almost anything you wish for even the internet is included.
And development is fully native.

That should be worth something to you.
Especially the fact that **you can now get professional help for your projects through the Lazarus Factory**. We have the best developers available. Think of a project and we will build it for you, here you can get information:
professionals@lazarusfactory.org

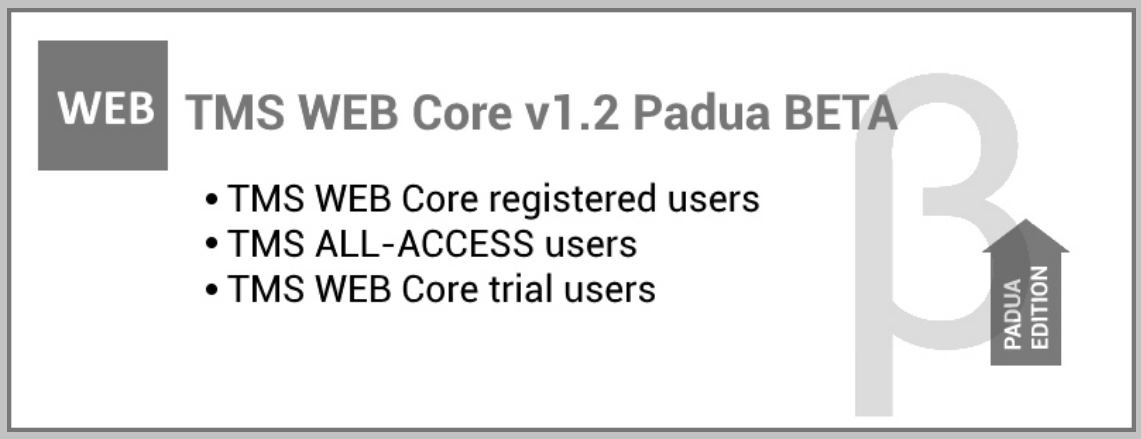
Think of the fact that you will have an IDE that is free! You will not have to pay license costs so you can spend your money on the components you need. So help us please!

We also work on Web Assembly which will become available in the last part of the year.

About the Lazarus handbook
- I will add the new finalized chapters for Lazarus: Installing under Windows / Linux and Mac, and then start on the ide explanations. The book will contain something like 850 pages!

We now are at 650. So the end of this enormous task is in sight...

For details please have a look at our site.



NEW update for TMS WEB Core v1.2 Padua BETA

We are nearing TMS WEB Core v1.2 Padua release and the new beta update available now is likely to be the last beta update before release.
Your help and your feedback is invaluable in ensuring that the quality of the release is as high as possible. We invite you to share all your comments via email and very much appreciate it.

TMS WEB Core v1.2 brings the following new features:

- New support for Electron application development
- New 3D web client controls for scenes, charts, math surface graphs
- New syntax highlighting memo control
- New IndexedDB support with an IndexedDB dataset component
- New HTML treeview component
- New HTML accordion component
- New responsive grid panel component
- New file picker control for easy access to local system files and improved TWebFileUpload component
- New web camera, bar & QR code scanner & media recorder component
- Latest pas2js compiler with anonymous method, class helper, advanced records support
- Extensions to grid and DB grid to allow checkboxes, buttons, non-DB columns in grid, sorting, record indicator
- Extensions for easier relative control rendering
- Extended documentation (PDF developer guide)
- Numerous improvements to IDE integration and framework

This beta version is available for TMS ALL-ACCESS, TMS WEB Core registered users and TMS WEB Core trial users!

For TMS ALL-ACCESS users, simply use the TMS Subscription Manager and from there, you can download and install TMS WEB Core v1.2 Padua beta version.

The v1.2 beta full version for registered users, can be download after login on our website under ?Account / My Products / TMS WEB Core beta?.

For trial users, the v1.2 beta trial can be downloaded from:
https://www.tmssoftware.com/site/tmswebcore.asp

Note: first manually uninstall previous version!

Share your experiences:
We take our time and listen very closely to your feedback as it helps enormously steering the product, documentation and demos updates.

- Is the scope of the demos clear?
- Is there any something you want to see covered with a demo that is not there yet?
- Did you also test the FNC web-enabled components?
- Is there some functionality or feature you?re missing?
- Do you have any other comment, feedback, issues to report?

Chapter 70

How to redirect a form to a shell

This tutor explains a solution to attach a console to your app. Basically we want an app to have two modes, a GUI mode and a non-GUI mode for any humans and robots.
A NoGUI app provides a mechanism for storage and retrieval of data and functions in means other than the normal GUI used in operating systems.
From everything you've read this is supposed to work if we use **AttachConsole().**

Figure1: Two modes-command prompt and Graphic

First we generate the declaration of the 2 DLL's:
It attaches the calling process to the console of the

```
function AttachConsole(dwProcessID: Integer): Boolean;
      external 'AttachConsole@kernel32.dll stdcall';

function FreeConsole(): Boolean;
      external 'FreeConsole@kernel32.dll stdcall';
```

specified process and if the function succeeds, the return value is nonzero.
A process or app can use the **FreeConsole()** function to detach itself from its console. If other processes share the console, the console is not destroyed, but the same process that called **FreeConsole()** cannot refer to it.
Next we have a function to get the parent process name:

```
function GetParentProcessName(): String;
```

This function needs another DLL from the lib PsAPI:

```
function GetModuleFileNameEx(Handle: THandle; pid: THandle;
                ppath: Pchar; buffsize: DWORD): DWORD;
      external 'GetModuleFileNameExA@psapi.dll stdcall';
```

This allows me
to run my GUI app from a command prompt and display output to the same console where my app was launched. Otherwise, it will run the full GUI part of the app and shows the window as a win- or webform.
The GetParentProcessName() asks the command prompt (powershell or cmd):

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

C:\Windows\SysWOW64\cmd.exe

There's no reliable way for a GUI subsystem application to attach to the console of its parent process. If you try to do so you'll end up with two active processes sharing the same console and only one is running:

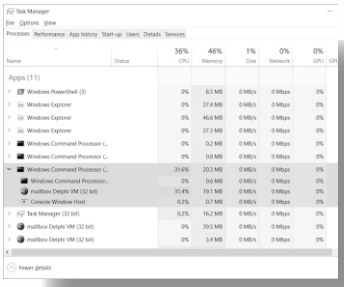


Figure2: See readable size next page

Be careful with this **GetModuleFileNameEx().**
In Win 7, it shows up in the lib kernel32.dll, so you might want to code to check for this and load dynamically as I do with GetModuleFileNameExA().
The main part is as follow:

```
ParentName:= strlower(GetParentProcessName());
ParentName:= PathExtractName(ParentName);
Set_ReportMemoryLeaksOnShutdown(false)

if (ParentName='cmd.exe') or (ParentName='powershell.exe')
 then begin
   AttachConsole(-1);
   NativeWriteln('Start with maXbox4 Console Output--->');
   for it:= 1 to 50 do if IsPrime(it) then
              NativeWriteln(IntToStr(it)+' is prime');
   NativeWriteln('-----end-----');

FreeConsole();
```

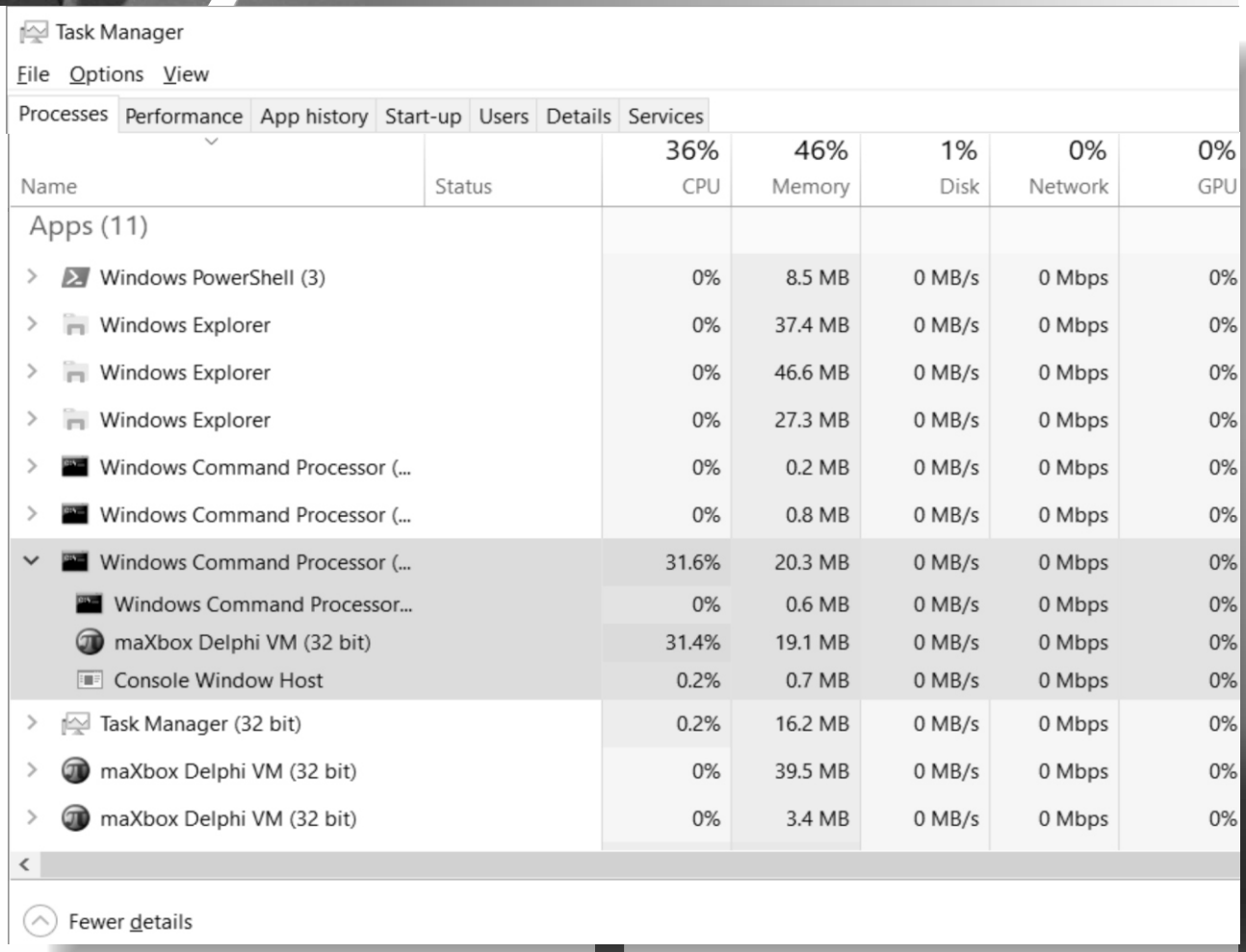


Figure2: Task Manager

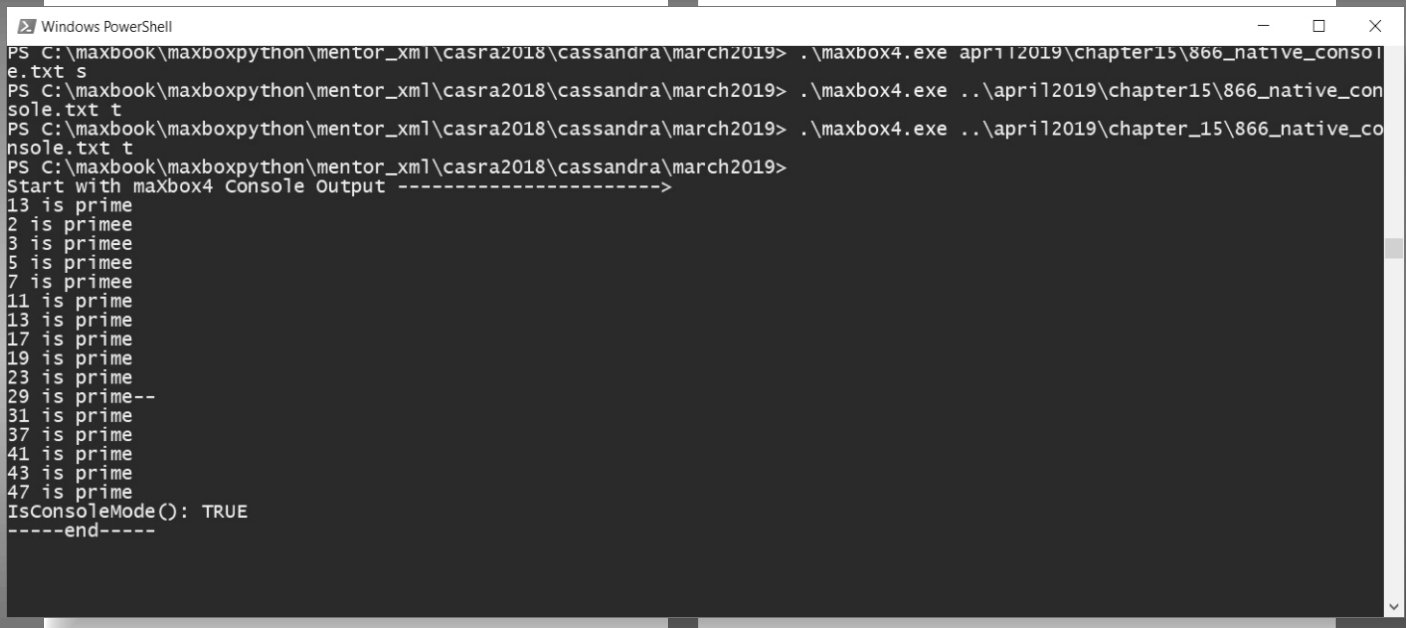


Figure3: Console (CommandPrompt)

This is is just fine if you are just wanting to display output into the command line. But operations like redirecting output into a file for example are not working e.g.: **`start /wait Checker.exe > out.txt`** would still output into console and not into **`fileout.txt.`** Different solution exists for the PowerShell:

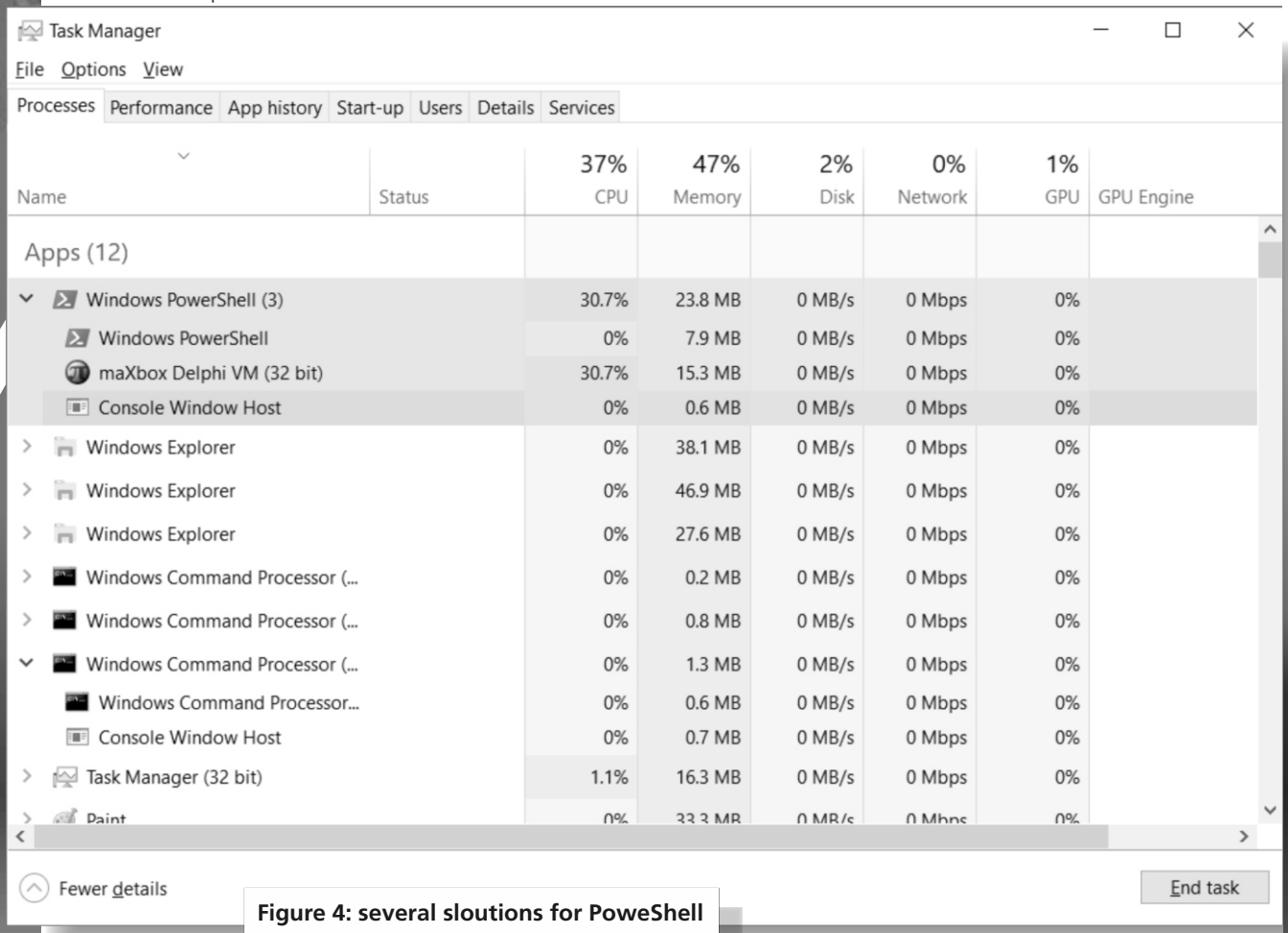


Figure 4: several sloutions for PoweShell

If you are lost into the source code then you could easily add parameters to your app to write output to a file instead of the console: -o out.txt, since it's your tool doing the writing, you can write wherever you want for example to start out of the shell and get output to the shell and in the end plot an image to

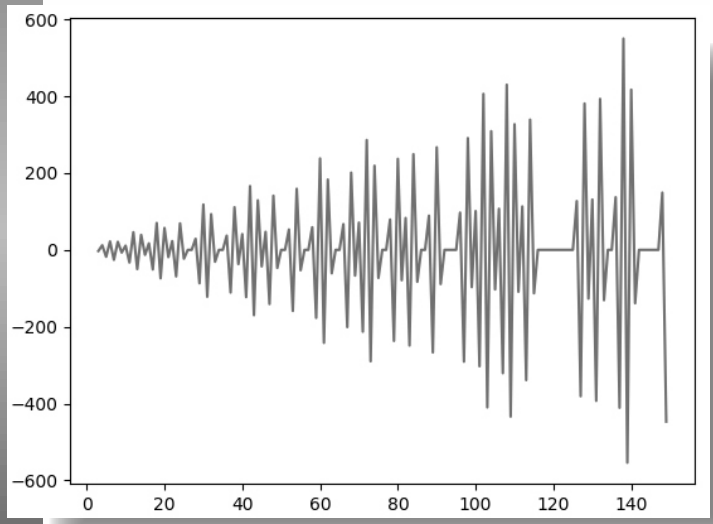


Figure 5: several sloutions for PoweShell

another file output as a png-graphic like below:
We believe the best option is to create two separate executables or scripts. One for the GUI subsystem, and one for the shell subsystem. This is the approach also taken by:

Java: java.exe, javaw.exe.
Python: python.exe, pythonw.exe.
Visual Studio: devenv.com, devenv.exe.

A console is closed when the last process attached to it terminates or calls FreeConsole. After a process calls FreeConsole, it can call the AllocConsole function to create a new console or AttachConsole to attach to another console. Call the script from the shell with

maXbox

The script can be found:

http://www.softwareschule.ch/examples/866_native_console.txt

Author: Max Kleiner

```
function GetParentProcessName(): String;
var
 HandleSnapShot: THandle;
 EntryParentProc: TProcessEntry32;
 CurrentProcessId: THandle;
 HParentProc: THandle;
 ParentProcessId: THandle;
 ParentProcessFound: Boolean;
 ParentProcPath: String;
begin
 ParentProcessFound:= False;
 HandleSnapShot:= CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
 if HandleSnapShot<>INVALID_HANDLE_VALUE then
 begin
  EntryParentProc.dwSize:= SizeOf(EntryParentProc);
  if Process32First(HandleSnapShot,EntryParentProc) then
  begin
   CurrentProcessId:= GetCurrentProcessId();
   repeat
    if EntryParentProc.th32ProcessID=CurrentProcessId then begin
     ParentProcessId:= EntryParentProc.th32ParentProcessID;
     HParentProc:= OpenProcess(PROCESS_QUERY_INFORMATION or
        PROCESS_VM_READ,False,ParentProcessId);
     if HparentProc<> 0 then begin
      ParentProcessFound:= True;
      SetLength(ParentProcPath,BufferSize);
      GetModuleFileNameEx(HParentProc,0,PChar(ParentProcPath),BufferSize);
      ParentProcPath:= PChar(ParentProcPath);
      CloseHandle(HParentProc);
     end;
     Break;
    end;
   until not Process32Next(HandleSnapShot,EntryParentProc);
  end;
  CloseHandle(HandleSnapShot);
 end;
 if ParentProcessFound then Result:= ParentProcPath
 else Result:= '';
end;

>>> from geopy.geocoders import Nominatim
>>> geolocator = Nominatim('maxbox-app')
>>> place,(lat,lng) = geolocator.geocode("Breitenrainplatz 2 Bern")
>>> print ("%s: %.5f, %.5f" % (place, lat, lng))
```

RAD STUDIO ROADMAP MAY 2019

Marco Cantu 23 May2019

With this article, the RAD Studio PM team is introducing an updated roadmap for Delphi, C ++Builder and RAD Studio. This May 2019 roadmap includes plans for the next 12 months. Alongside, we have also published a May 2019 Roadmap PM Commentary blog post, with more details and information.

SAFE HARBOR STATEMENT

These plans and roadmap represent our intentions as of this date, but our development plans and priorities are subject to change.
Accordingly, we can't offer any commitments or other forms of assurance that we'll ultimately release any or all of the described products on the schedule or in the order described, or at all. These general indications of development schedules or "product roadmaps" should not be interpreted or construed as any form of a commitment, and our customers' rights to upgrades, updates, enhancements and other maintenance releases will be set forth only in the applicable software license agreement.

RAD Studio Today

Key features delivered in recent months:

- IDE Productivity Tools including Bookmark and Navigator
- Expanded support for iOS 12 and iPhone X series devices
- Custom VCL Windows and FMX Styles
- HTTP and SOAP Client Library Enhancements on Windows
- RAD Server Console UI redesign and migration to the Ext JS framework
- Enhanced FireDAC support for Firebird 3.0.4 and Firebird embedded
- C++ 17 Win32 Support
- Delphi language enhancements Improved IDE UX

10.3.1 was released in February 2019

RAD Studio Personas *Features are not committed until completed and GA released*

Focus Areas for CY 2019 / 2020

Windows Desktop Devs

- IDE and VCL Enhancements
- Delphi Language Features
- Continued Windows 10 platform support Enhanced IDE Code Tooling
- RTL optimization
- Clang/LLVM upgrade (C++17) for Win 64

.

Multi-Device Developers

- Support for latest versions of iOS and Android
- Android 64-bit Android Native Controls
- Re-architected Android Firebase Push support
- macOS 64-bit
- New multi-device styles

Enterprise Developers

- Support for the latest database drivers in FireDAC
- RAD Server configuration, deployment, and management tools
- IDE code tooling improvements for large codebases

RAD Studio Roadmap *RAD Studio CY 2019 / CY 2020 Timeline***

RAD Studio 10.3.2
mid 2019

Platform Enhancements
Enhanced RAD Server Tooling

Delphi
macOS 64-bit platfonn support

User Experience
FireMonkey UI Templates
Further IDE UI/UX improvements

C++
Windows 64-bit C++17 support
Enhanced C++ LSP support

Quality Focus Areas
VCLHigh DPI
FireMonkey Platform
RTL Perfonnance
C++17 Win32 Compiler
IDE Code Tooling

RAD Studio 10.4
late 2019

Platform Enhancements
Metal 2 GPU driver support (macOS/iOS)
Enhanced Windows Theming
Additional RAD Server Tooling Enhancements
Android Push Notifications and Google Play Services re-architecture
LiveBindings Performance Enhancements

Delphi
Android 64-bit platform support
Language Server Protocol for Delphi Language Enhancements
Unified memory management across all platforms

User Experience
Further IDE UI/UX Improvements
GetIt Package Manager Enhancements
VCL High DPI Styles Support
VCL Per Control Styling
Unified Installer for Online & Offllne Installations

Additional Quality Focus Areas

C++
Expand C ++ libraries support
Toolchain perfonnance and quality improvements
Unified memory management across all platforms

RAD Studio 10.4.x
2020

Platform Enhancements
Consolidate muttiple debuggers, focusing on LLDB tech
FireMonkey quality improvements
MSIX app packaging format support

Delphi
iOS 13 Simulator support

User Experience
High OPI IOE Support

Additional Quality Focus Areas

C++
IOE Language Tooling with Visual Assist support for C++
CMake extended support with IOE integration

RAD Studio 10.3.2 *Features are not committed until completed and GA released*

Native 64-bit macOS platform toolchain for Delphi, C++ language modernization for 32-bit and 64-bit Windows

RAD Studio {all IDEs)

- Further IDE UI/UX improvements refine and improve the user experience introduced in 10.3
- Develop cross-platform apps faster with new FireMonkey UI Templates
- Simplify RAD Server development and deployment with enhanced RAD Server Wizards and Installers
- Support the latest Google standard for Android Push Notification with Firebase support
- New Downloads/Licensing Portal makes discovering your products and license keys much simpler and easier

Delphi

- Address the Apple App Store and macOS platform requirement with 64-bit macOS platform support, including the entire development, deployment, and debugging experience
  - New macOS 64-bit compiler and macOS runtime library updated for 64-bit support
  - Database support and FireDAC drivers
  - HTTP client libraries, including cloud and RAD Server clients
  - Full support for the FireMonkey UI library and platform integration
  - App Store deployment support
  - macOS app notarization support

RAD Studio 10.4 *Features are not committed until completed and GA released*

Platform Enhancements

- Address new Apple platform requirements for GPU drivers by adding Delphi Metal 2 support
- Enhanced RAD Server Development Experience with additional Enterprise capabilities
- Android Firebase Push Notifications and Google Play Services re-architecture for Android and iOS to simplify development of multi-device apps
- Redesign of the LiveBindings architecture to improve performance and capabilities

User Experience

- Additional Getlt Package Manager capabilities for providing unique content to Subscription customers
- VCL High DPI Styles Support, including updated styles and VCL per-control styling, improves the look and feel of your applications and their behaviour on modern displays
- A Unified Online/Offline Installer using Getlt gives you the same swift, efficient installer when you are not connected online

RAD Studio 10.4 *Features are not committed until completed and GA released*

Delphi Android 64-bit support, improved code tooling, language enhancements, and more

Delphi

- Address the Google Play Store requirement for Android 64-bit platform support (compilers, IDE tooling, RTL, database support, FireMonkey)
- Add Managed Records support to the Delphi language to offer additional memory management options and help modernize existing codebases
- Improved in-lDE language tooling using a Delphi LSP server giving performance enhancements through asynchronous, out-of-process processing, and more accurate results for code completion, error insight, and related tools
- Unified memory management across all platforms (disabling ARC on mobile)

C++

- Expand out-of-box C++Builder support for popular C++ libraries in Getlt, to leverage existing libraries & improve development speed
- Toolchain performance and quality improvements assist you in more reliable development behaviour and upgrading from the classic compiler
- Unified memory management, ie removal of ARC, simplifies code behaviour and allows you to rely on canonical C++ memory management mechanisms, like smart pointers
- Improved DLL and package symbol import/export behaviour to assist with common issues using multiple libraries, plus linker quality for larger projects

RAD Studio 10.4.x *Features are not committed until completed and GA released*

New C++ code tooling, 64-bit iOS Simulator support, High DPI IDE

Platform Enhancements

- Consolidate multiple debuggers, focusing on LLDB technology, for unified full-featured debugging on all platforms
- Improve FireMonkey quality on all platforms for better multi-device development experience
- Support the MSIX App Packaging format for Desktop Bridge and Windows Store
- Metal GPU driver support for the iOS platform to improve apps performance

Delphi

- iOS 13 Simulator support (64-bit) for targeting the latest iOS Simulator devices

C++

- Integrated Whole Tomato Visual Assist support for C++ provides best-of-class code tooling, beyond that provided in other C++ IDEs

User Experience

- High DPI IDE Support allows you to use RAD Studio on a high-res display with crisp text and images

RAD server *Features are not committed until completed and GA released*

REST API for microservice architectures

Development

- Add new IDE Wizard to streamline development of database driven RAD Server applications
- Extensions to the Swagger API for better REST API documentation support
- Code simplification for better development and maintenance of RAD Server extensions
- Additional Enterprise-level features

Research

- Further work on web capabilities and Ext JS library integration

Deployment

- Ready-to-use installers to deploy the production engine on Windows and Linux servers
- Tools to help with configuration and management of RAD Server installations
- Cloud VMs and container based (Docker) deployment support, to simplify use of RAD Server REST APIs on modern architectures

Research Areas *Features are not committed until completed and GA released*

Getting ready for future platforms and technologies

- **Platforms**
  - BitCode support and other LLVM-related improvements
  - Target ARM loT platforms (Raspberry Pl, Windows ARM)
  - Deeper Win RT UI integration with VCL and FireMonkey
- Under Consideration
  - FireMonkey Linux GUI support
  - Cloud based RAD Studio projects continuous integration
  - Further FMX native controls (additional controls; macOS)
- Previous Research Areas, now on Roadmap or Completed *
  - Unified installation experience with combined Web and offline installer {10.4)
  - Integrating Whole Tomato's Visual Assist for C++Builder (10.4.1)
  - Language Server Protocol support for Delphi {10.4)
  - FMX Native Controls for Android (completed in 10.3)

starter expert

INTRODUCTION

Many Pascal programmers install the **ZEOS** database framework, thinking it is needed to connect their database to an application.
While that is a well-tried route for many Delphi programmers, for Lazarus and FPC users ZEOS is not necessary (though it is certainly a good alternative to the framework that is the subject of this article).
For many years Lazarus has included the SQLdb framework as a package installed by default in the IDE. The following two links give you lots of information about the SQLdb framework:

http://wiki.freepascal.org/SQLdb_Programming_Reference#Documentation

https://www.freepascal.org/docs-html/fcl/sqldb/usingsqldb.html

The SQLdb unit defines four principal data-handling classes applicable to all popular SQL-based databases. In what follows you gain experience in using the first three:
TSQLConnection

- **TSQLConnection**
  represents the connection to the database. You must set the properties of your connection instance appropriately according to the particular connection you want to establish: the database, the server, the user name and password. **TSQLConnection** is an abstract class, which is never used directly. Rather you use a descendant according to database type (**MS SQL, SYBASE, POSTGRES, ORACLE, MYSQL, SQLITE3, INTERBASE (FIREBIRD) OR ODBC**) such as **TOracleConnection** or TSQLite3Connection.
  Note that non-SQL-enabled legacy databases such as dBase and Foxpro are not supported by SQLdb.

- **TSQLTransaction**
  represents the transaction in which an SQL command is running. SQLdb supports multiple simultaneous transactions in a database connection. For databases that do not support this functionality natively, SQLdb simulates this by maintaining multiple connections to the database.
  A transaction is a hidden object interposed between the local data and the database. When the user inserts a new record, or modifies an existing record, the **TSQLTransaction** instance holds the modified data. This data change is propagated to the database itself by calling **TSQLTransaction.Commit,** or the data change can be thrown away by calling **TSQLTransaction.Rollback.**
  Transaction components have a very important `Action` property which specifies how query execution will affect the database.
  `Action` can take one of 5 possible values:

  - **caNone:**
    no change is made to the database, and the developer has to set the transaction manually.
  - **caCommit:**
    after the query is executed, changes are sent to the database automatically and the transaction is closed.
  - **caCommitRetaining:**
    after the query is executed changes are sent to the database automatically and the transaction remains open.
  - **caRollback:**
    after the query is executed, no changes are sent to the database, and the transaction is closed.
  - **caRollbackRetaining:**
    after the query is executed, no changes are sent to the database and the transaction remains open.

- **TSQLQuery**
  is a component descending from the FCL's **TDataSet** which can be used to view and manipulate the result of an SQL select query.
  It can also be used to execute all kinds of other SQL statements – it is not restricted to **SQL SELECT.**

- **TSQLScript**
  can be used when you need to execute numerous SQL commands against a database - for example when creating a database.

THE RAD PRINCIPLE

SQLdb has been designed to facilitate making database connections with only a few clicks. In the Lazarus IDE the SQLdb components are found on the Component Palette page named SQLdb (see Figure 1).

SQLdb works with third-party client libraries in order to establish the best connection to the databases it supports.
Most well-known databases are supported:

Figure 1: the Component Palette SQLdb

Firebird, MS SQL, Postgres, MySQL, SQLite, Sybase and Oracle.
No other databases are supported directly, but provided the database has an **ODBC driver** then SQLdb can connect using its **ODBC connection.** For instance, you can connect to **Access databases** using SQLdb via **Microsoft's Access library.**

STARTING TO USE SQLDB

To begin using SQLdb you need a connection to a database which can be either a client-server database or an embedded database (such as **Firebird embedded**, **MySQL** or **SQLite).**
This article develops an example application using SQLite since it is such a widely available back-end and ideal for an initial single-user example.
The example project has a main form containing the principal GUI, built using the LCL's data-aware components located on the Data Controls page.
The example then adds a second form with data-aware controls to demonstrate how **SQL JOIN** statements work, and how to edit and update joined tables.
The example concludes by getting you to develop a single consolidated editing form that has similar functionality, introducing a lookup data-aware control to make implicit **JOINs** as needed.

A FEW NECESSARY PRELIMINARIES

While you can just unpack the source code provided and try it out (see the closing paragraph of the article for instructions), you will learn more by following along and developing the SQLdb example application for yourself. You will also benefit by learning to use a database browser to create and edit databases quickly.

A SQLite browser such as **SQLiteStudio** (an open source tool with versions for Windows, Linux and Mac OSX) is fairly easy to use. You can find this tool at **`https://sqlitestudio.pl/index.rvt`** if you don't already have it installed.
First, create a new directory where you have **full permissions,** which you should name **`sqldb.`**
With the help of the SQLiteStudio tool create a database named **`authors.db3`** in this new **`sqldb directory.`**

This is a beginner's convenience, since it is also possible, of course, to create a SQLite database at runtime in code. We don't do that here since it merely complicates learning SQL for a beginner.
To begin with, it is simpler to start with a ready-built database. The first table in our new **`authors.db3`** database will be named **`authors`** and should have columns defined by the following **SQL DDL** statement:

```
CREATE TABLE authors (
id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL,
lastname VARCHAR (30) NOT NULL,
firstname VARCHAR (30),
countrycode VARCHAR (3));
```

A countrycode (rather than a **`'country'`**) column has been introduced here because this article will shortly explain how to write **`JOIN`** queries. This field and its data, unused to begin with, is designed for use in a future **`JOIN`** query.
You must also have the correct **SQLite3 library** on your development system (**sqlite3.dll** on Windows, sqlite3.so on Linux which is often a symlink to libsqlite3.so.0).
This must be installed in the system path or in the same directory as the executable (Windows) or installed in the library path (Linux, Mac OSX).
This applies both to your development machine, and any computer where the released application will run. Windows database developers often include the installation of sqlite3.dll alongside their application .exe file when deploying the released application. The required library is installed by default on most popular Linux distributions, but this is not guaranteed.
If the required client library is missing, attempting

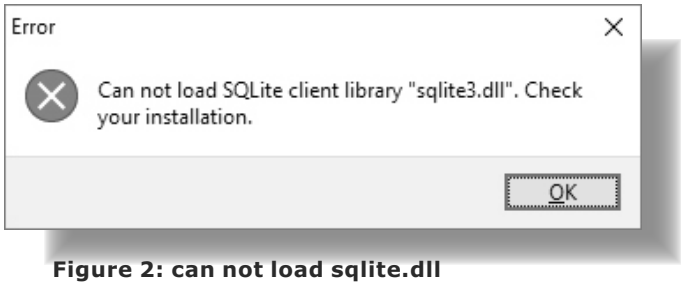


Figure 2: can not load sqlite.dll

to run your database executable will give an immediate error (see Figure 2).

Next, create a new Lazarus project **(Project ➔ New Project... Application, OK)** in the sqldb directory, and save it with the name **`authors.lpi`**. Save the project's main form in a unit named main.pas (or main.pp), and leave the name of the form variable as Form1. Although **SQLdb** components can be dropped on a form, it is preferable to have a non-visual container for them in the form of a data module. So create a data module via **File ➔ New... Data Module, OK.** The default name DataModule1 for the module variable is fine. Save the data module unit with the name **`DataModule.pas.`**
After saving, the DataModule unit needs to be added to the uses section at the top of the main form's unit in the interface section. Or you could create a new uses section just under the keyword implementation and add DataModule there:

```
implementation
   uses DataModule;
```

Now you can drop three SQLdb components on **`DataModule1:`**

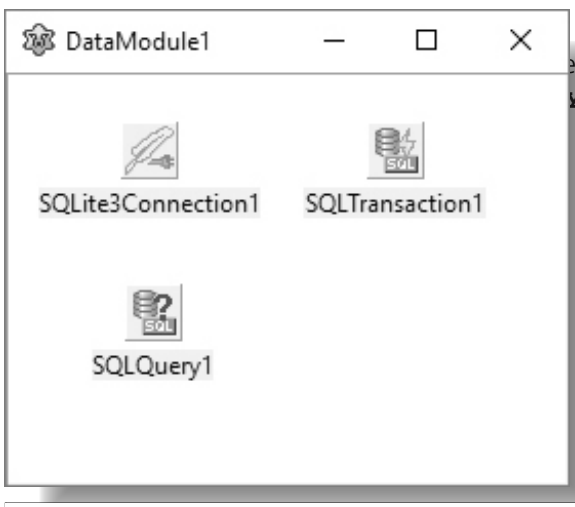


Figure 3: the three components on the Datamodule

component. You can leave their default names as **`SQLite3Connection1`**, **`SQLTransaction1`** and **`SQLQuery1.`**

Your data module should now look something like Figure 3.

CONNECTING TO A DATABASE
The properties of the SQLdb components must now be set correctly.

❶ **`SQLite3Connection1:`**
DatabaseName - Click the ellipsis button of the **DatabaseName** property editor to locate the **`authors.db3`** database file you created earlier. (Remote databases usually require a username and password, for which the **`UserName`** and **`Password`** properties are provided.
A further option usually required for security is a login prompt, so that when any user attempts to connect to the database, a dialog will first be shown requiring entry of a valid username and password.
You enable this feature by setting the **`LoginPrompt`** property to True. These property values control the login phase, which is not needed for this example).
`Transaction`
- set this property to **`SQLTransaction1`** (select this from the dropdown).
`HostName`
- this property is not required for file-based embedded databases such as SQLite. Whereas if the database is on a server, the host name or an IP-address is required to find the database.

❷ **`SQLTransaction:`**
`Action` - set this to **`caCommitRetaining.`**
`DataBase` - this will already be set to **`SQLite3Connection1`** from your earlier setting of **`SQLite3Connection1.Transaction.`**

❸ **`SQLQuery1:`**
`Database` - set this to

SQLiteConnection1 (select from the dropdown).
SQL - click the ellipsis button to open an editor where you should type
SELECT * FROM authors
Options - for fully automated execution check the following to include their values in the **Options** set:
sqoKeepOpenOnCommit, sqoAutoApplyUpdates, and **sqoAutoCommit`.**

Once the initial connection properties are set correctly, the user can connect to the database by setting the `SQLiteConnection1.Connected` property to True. Now when the `SQLQuery1.Active` property is set to True, this causes fresh communication between the database and client, and the query result is put into a dataset. Rather than get the user to set these properties explicitly, our example will open and close the database connection, and activate and deactivate the query in the main form's **OnShow** and **OnClose** events. Select the main form in the object inspector, and on its **Events** page, double-click on the `OnShow` event to generate a new handler in the code editor where you need to type the following lines of code:

```
DataModule1.SQLite3Connection1.Open;
DataModule1.SQLQuery1.Active := True;
```

To close the database connection at the program's end, double-click the `OnClose ` event, and type these lines in the `OnClose` event handler:

```
DataModule1.SQLQuery1.Active := False;
DataModule1.SQLite3Connection1.Close;
```

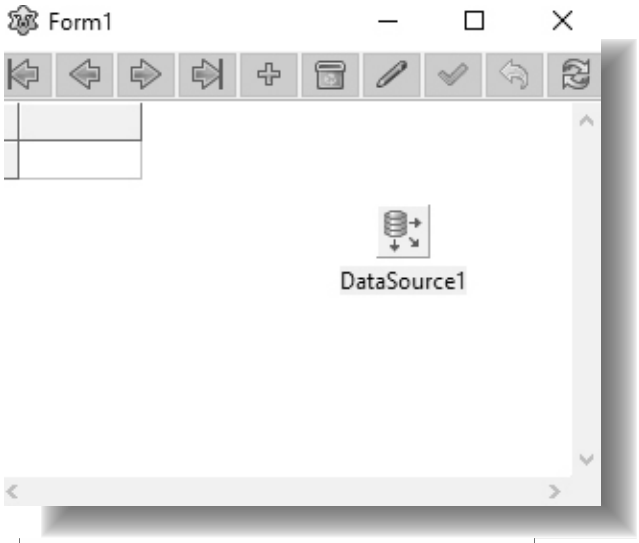


Figure 4: three data aware components

These three components offer the following functionalities:

- **DataSource1:** provides a data link between the SQLdb components and the data-aware GUI components. Set its `DataSet` property to **DataModule1.SQLiteConnection1** (use the dropdown editor provided to do this).
- **DBGrid1:** shows data from the query laid out in rows and columns. Set its DataSource property to **DataSource1.**
- **`DBNavigator1:** offers buttons to navigate through the dataset, and to insert new data and edit existing data. Sets its **DataSource** property to **DataSource1.**

DISPLAYING AND EDITING DATABASE DATA

To provide visual access to the data in the database you must add three data-aware components to the main form in order to view the table data (see Figure 4).

Drop a TDataSource component from the Da
Access palette page, and a TDBGrid and TDBNavigator from the Data Controls palette page on to the main form.
You can leave their default names unchanged. The main form should now look like Figure 4.

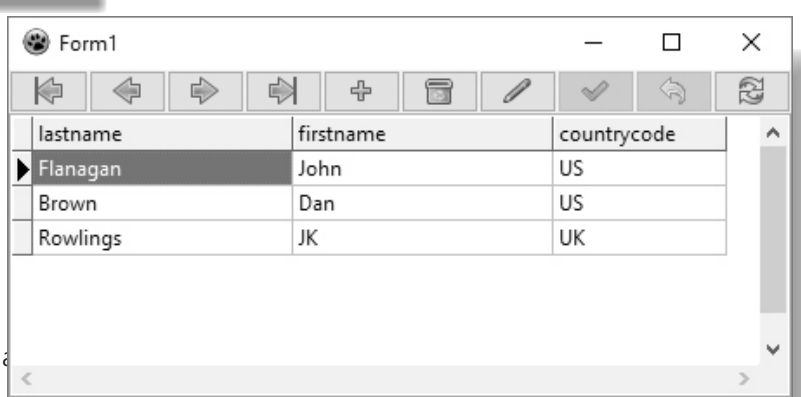


Figure 5: Countrycode

Now when you compile and run the application the main form displays an empty DBGrid. To insert a new record, click on the **+ button** in the **navigator.** Don't fill the first column **id** since this is an **auto-increment field** which will be completed by the database when the new record is posted. However, complete the cells in the other columns (*the column header indicates the corresponding field in the authors table*). Then post the record by clicking the check button on the DBNavigator to save the record. In this way complete a few entries for further author names. If you now close and restart the application, you should see the records you have recently added.The SQLdb framework performs all the data communication, and **SQLQuery1 creates an insert / update query** to send new or altered data to the database.

IMPROVING THE USER'S EDITING EXPERIENCE.
When the record is saved, a new record id is auto-created by the database, which is visible by default in the DBGrid. However, the user should not be able to see this housekeeping field (it's irrelevant to them), and indeed it should be impossible for a user to edit its value).

To ensure that three fields (**lastname, firstname** and **countrycode**) are displayable and editable in the DBGrid, and that the id is not shown or editable you have at least two possibilities:

❶ On **DataModule1** ensure that **SQLite3Connection1.Connected** is **True,** and that **SQLQuery1.Active** is **True.**
Right-click the DBGrid on the main form and select Edit Columns...
Click the **+ Add Fields button** to add all the database fields.
Select only the id field, and click the **- Delete button** (it has a blue minus icon). The grid display in the Designer will immediately lose the id column and its data.
When you run the application there is no display of the **id** *column.*
When you add a new record, a new id value gets ***auto-generated*** *when the record is posted,*
but the user is unaware of this, since it cannot be seen.

❷ Right-click the DBGrid on the main form and choose Edit Columns... from the context menu.
If any fields are shown, click the **- Delete All button.** Then for each of the three fields you want displayed click the **+ Add button**.
In the object inspector find the FieldName property for each new field, and choose the appropriate field name from the dropdown.
The grid display will be updated immediately at design time. Close the application to save the changes.

By default Lazarus sets each DBGrid column's Width property to 64.
Lazarus recalculates each column Width when the application begins to run, and usually the auto-width setting is too wide. Provided your data is reasonably consistent in length it is better to set the column **Width** to a suitable value that differs from the default. When you do this, the application has a better appearance

SQL UPDATES

There are three ways to use SQL statements to make updates to the database, corresponding to the three possible values of **SQLQuery1.UpdateMode:**

① **upWhereAll**
This option uses all the query field's old values to update data in the database. In addition to the key fields, the **WHERE** statement contains all old field values for fields that have pfInWhere set in their Provider **Flags** property. This ensures that if another user has simultaneously changed any part of the record, the update statement will fail.

```
UPDATE authors
SET lastname = 'lastname'
WHERE id = old_value_id
AND lastname = old_value_lastname
AND firstname = old_value_firstname
AND countrycode = old_value_countrycode;
```

② **upWhereChanged**
In addition to the key fields, the **WHERE** statement contains only the old values of all changed fields which have **pfInWhere** set in their **ProviderFlags** property.
This ensures that if another user changed any of these changed fields, the update will fail, and the other user's changes are safe.

```
UPDATE authors
SET lastname = 'lastname'
WHERE lastname = old_value_lastname;
```

③ **upWhereKeyOnly**
Only the key field(s) are used in the **WHERE** clause to determine which record(s) to update.

```
UPDATE authors
SET lastname = :lastname
WHERE id = old_value_id;
```

The third option **(upWhereKeyOnly)** is selected by default and is the logical way to save data to the database in this single-user case.
If the table is configured correctly, the id will be found very quickly in the database so that data changes are applied to the correct record.

ADDING A FURTHER DATABASE TABLE
Small, coded fields such as countrycode are beneficial in databases since they reduce the application's memory footprint, enabling well-normalized data to be added, and greatly reducing the possibility of storing duplicate, redundant data.

You can either use an i**nteger field** or a character field as a code such as countrycode. We used fairly self-explanatory codes in the countrycode values seen in Figure 5 above, but more often the code is an integer which has no self-explanatory meaning. Users often want to see a data column that has an immediate meaning, rather than a code (*which is a programming convenience adopted to minimise database size, and provide a means of joining related fields together for display*). To translate the countrycode data into a more meaningful country name we need to create a new table using SQLiteStudio:

```
CREATE TABLE countries (
   id INTEGER PRIMARY KEYAUTOINCREMENT,
   countrycode VARCHAR (4) UNIQUE NOT NULL,
   countryname VARCHAR (30));
```

Select **DataModule1** in the designer, and drop on it a further **TSQLQuery** component.
Name this **SQLCountries.**

```
Set SQLCountries.Database to SQLite3Connec
Set SQLCountries.Transaction to SQLTransac
Set SQLCountries.UpdateMode to upWhereKeyO
Set SQLCountries.UsePrimaryKeyAsKey to Tru
Set SQLCountries.Options to sqoKeepOpenOnC
      sqoAutoApplyUpdates, sqoAutoCommit
Set SQLCountries.SQL to
SELECT * FROM countries ORDER BY countryn
```

We now need a GUI to aid data entry into this new table. Create a new form in the example application by choosing **File ➔ ÚNew Form.**

Name the new form unit **uCountries,** and in the **Object Inspector** give the form the Name **FrmCountries.**
Drop a **TDBNavigator,** a **TDBGrid** and a **TDataSource** on it as you did for the main form, leaving the components with their default names.

❶ Set **DBNavigator1.DataSource** to **DataSource1.**
❷ Set **DBGrid1.DataSource** to **DataSource1.**
❸ Next add **DataModule** to the uses clause of **uCountries**
(either in the interface, or in the implementation section).
❹ Set **DataSource1.DataSet** to **DataModule1.SQLCountries.**
❺ On **DataModule1** ensure that **SQLCountries.Active** is set to **True.**
❻ Right-click the **DBGrid** on **FrmCountries** and choose **Edit Columns...** from the context menu.
❼ Click the **+ Add Fields** toolbutton in the column editor to add the three columns defined in the countries table.
❽ Select the first line **(0 - id)** and click the - Delete toolbutton to remove it from the grid display, so it cannot be seen or edited by the user.
❾ In the object inspector adjust the Width property of the remaining two fields to give a good display.
❿ On **DataModule1** return to **SQLCountries** and set **SQLCountries.Active** to False, since we will add code shortly to edit this property at runtime.

In order to fill the countries table we need access to the new form just added.
On the main **form (Form1)** add **uCountries** to the **uses** clause.
Drop a **`TMainMenu`** on the main form and use the **menu editor** to add a main menu item captioned **Forms** and below it add a submenu item captioned **Countries .**

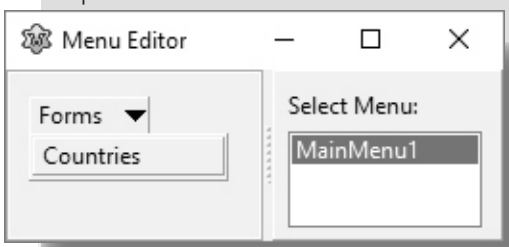


Figure 6: adding a MainMenu

Double-click the sub-menu to generate an event handler, and complete the generated handler with this line:

```
`FrmCountries.ShowModal;`
```

Then select **FrmCountries** in the Object Inspector and add an **OnShow** and an **OnClose** handler for the countries form, as you did for the main form.The **OnShow** handler should contain the line:

```
DataModule1.SQLCountries.Active := True;
```

The **OnClose** handler should contain the line:

```
DataModule1.SQLCountries.Active := False;
```

Compile the whole project again and run it.
Select the **Countries** menu item, which opens the new form where you can add new country records. Complete new records appropriate to the data in your authors table. If you have used data as in Figure 5 you would add a **UK United Kingdom** record, and a **US United States of America** record (see Figure 7).

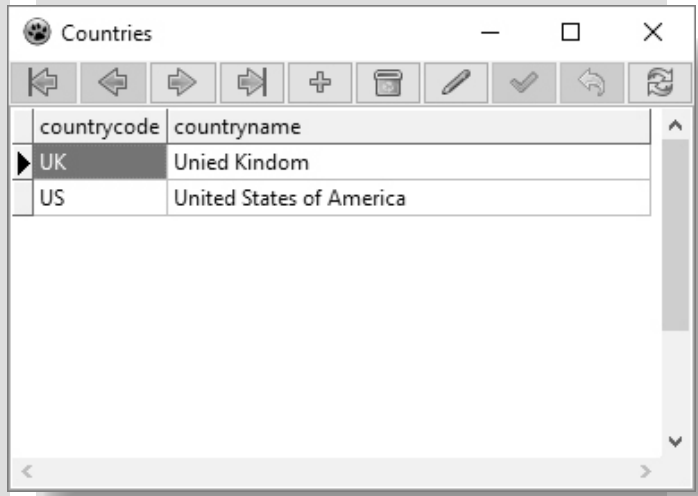


Figure 7: Adding records

JOINING TABLES USING SQL

We now have two tables which can be joined together in a relation based on the **countrycode** field present in each table, a relation which is called a join or a joined query. We do this by adapting the SQL entered in **SQLQuery1.**
Select **DataModule1** and edit the **SQLQuery1.SQL** property to read as follows:

```
SELECT
authors.id, authors.lastname,
authors.firstname, countries.countryname
FROM authors
OUTER LEFT JOIN countries
ON countries.countrycode =
authors.countrycode;
```

If you recompile and run the application you should see country names rather than the crude country code abbreviations in the last column, and the name of the last DBGrid column has changed from **countrycode** to **countryname**.

A DISCUSSION OF SQL SYNTAX POSSIBILITIES

Let's explain the SQL query syntax given above a little more.
Combining the table name and field name using dotted notation (e.g. **authors.id**) specifies the origin of the data unambiguously by identifying a specific field in a particular table.
If the SQL query does **NOT** identify the required field unambiguously, you will get an error (such as when an identically named field is present in several tables, which are not distinguished).
Optionally, you can use table aliases when working with similarly named fields found in several tables. An alias is simply a short form substituting for the full table name, perhaps the initial character of the table name. If you use aliases you have to show which alias marks which table.
For instance the above SQL query statement could be rewritten using aliases like this:

```
SELECT a.id, a.lastname, a.firstname,
c.countryname
FROM authors a
OUTER LEFT JOIN countries c
ON c.countrycode = a.countrycode;
```

The query is shorter and perhaps to some eyes also less easily readable.
If the query has many joins you have to search to discover which fields are a part of the table.
I prefer the alias syntax, because it gives a shorter query statement, and, if a table name ever changes, aliases mean that editing the SQL statement can be done with less typing.

OUTER LEFT JOIN tells the database to search for the referenced data in the specified table, and the search begins.
When a match is found, the requested field from table **c (c.countryname)** is then shown along with the specified table **a** fields.

Another way to search for the referenced data is using an **INNER JOIN**. This has the disadvantage that should the table **c** data not be found, then the entire record will not be shown (*the existing table* **a** *fields are not shown*).

LEFT OUTER JOIN queries always show all records, returning a NULL value if the referenced data is missing from the joined table being searched.
The user usually sees this as a blank field, rather than the display registering an error.

When you run the application the DBGrid shows the query with full country names.
But the GUI has changed, and the **Insert / Edit / Delete buttons** on the **DBNavigator** are now disabled. The dataset appears to be read only. This is because the **SQLQuery1** doesn't yet understand which table(s) must be **INSERTED / UPDATED** in this changed **JOIN** situation. To remedy this lack we need to create an **INSERT** and an **UPDATE** query to instruct **SQLdb** about which table to use. Make sure that **SQLite3Connection.Connected** is False, and then select **SQLQuery1** in the Object Inspector and edit the **SQLQuery1.UpdateSQL** property (which has been blank so far) to read as follows:

```
UPDATE authors
SET
lastname = :lastname,
firstname = :firstname,
countrycode = :countrycode
WHERE id = :id
```

Edit the **SQLQuery1.InsertSQL** property to read as follows:

```
INSERT INTO authors (id,lastname,firstname,countrycode)
VALUES (:id,:lastname,:firstname,:countrycode)
```

As you can see these queries work with **parameters. Parameters** are constructed by prefixing a field name with the ':' colon character. SQLdb parses the parameter to know which field each parameter specifies.With those changes, if you now run the application you should see that the Insert and Post buttons in the DBNavigator are now enabled, letting the user both insert and update table data (*See Figure 9*).

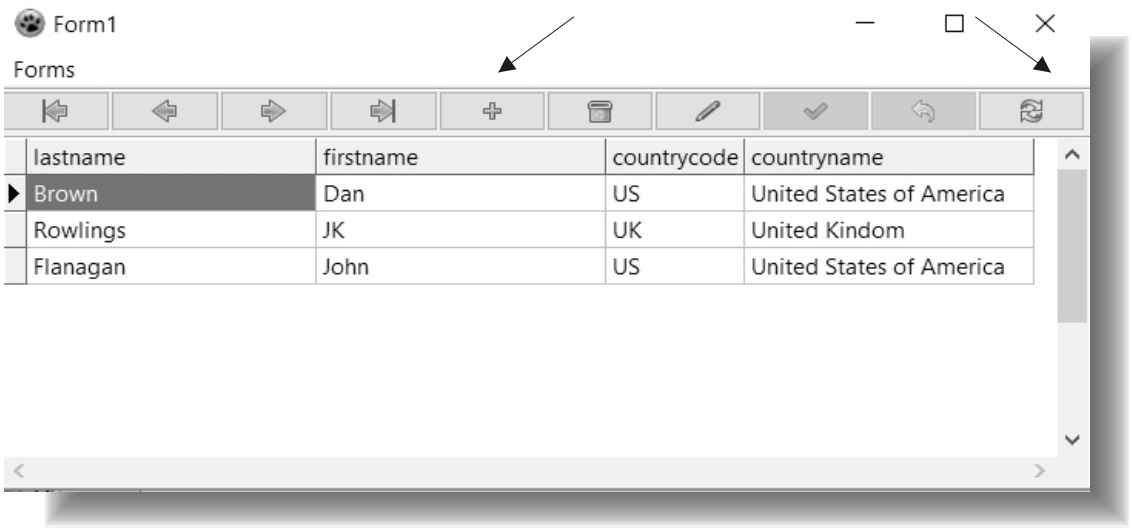


Figure 9: Insert and Post available

A DBGrid is an inconvenient way to edit and manage your data.
It would be better to let the grid be a display-only viewer.
You can then remove the **countrycode** column from the grid, which improves the display.
Also, this is a good moment to adjust the `DBGrid1.Options` property.
Set **`dgRowSelect`** to True, and **`dgTabs`** to False.
Additionally, let's remove the DBNavigator and create a custom toolbar of our own with only the buttons you want.

BETTER DATA MANAGEMENT

We will create a new form to handle inserting and updating the data, and leave the DBGrid on the main form as a data viewer, and use a lookup control to handle the table JOIN for us. Drop a further **`TSQLQuery`** on **`DataModule1`** and name it **`SQLAuthorsEdit`,** giving it the same properties as **`SQLQuery1`,** except for the **`SQL`** property in which you should type this value:

```
SELECT * FROM authors
WHERE id = :id`
```

- Create a new form **(File → New Form**) and save the unit as **`uAuthorEdit.pas.`**
- Name the new form variable **FrmAuthorEdit,** and drop on it a **`TLabel` (LblLastName)** and **`TDBEdit` (DBEditLastName).**
- Below them drop a **`TLabel` (LblFirstName)** and a **`TDBEdit` (DBEditFirstName).**
- Below them drop a **`TLabel` (LblCountryName)** and a **`TDBLookupComboBox`** (named **DBLookupCountry** - it is approximately the middle icon on the **Data Controls** palette page).
  This will enable the user to look up the country name in full, yet insert only a country code in the database. TDBLookupComboBox offers much the same functionality as the `JOIN` query given above.
- Below the **DBComboBox** drop a **`TButton`** (BtnPost) with the **`Caption` "Post"**. Drop a **`TDataSource` (DataSource1)** on the form.

Add `DataModule` to the uses clause of the new editing form.

- Add **DataModule** to the uses clause of the new editing form.
- Set **`DataSource1.DataSet`** to **DataModule1.SQLAuthorEdit.**
- Set **`DBEditLastName.DataSource`** to **DataSource1.**
- Set **`DBEditFirstName.DataSource`** to **DataSource1.**
- Set **`DBLookupCountry.DataSource`** to **DataSource1.**
- Set **`DBEditLastName.DataField`** to **lastname.**
- Set **`DBEditFirstName.DataField`** to **firstname.**
- Set **`DBLookupCountry.DataField`** to **countrycode.**

Other properties of **`DBLookupCountry`** can be filled automatically if we introduce a further **TSQLQuery.**

Drop a further **TSQLQuery** on DataModule1 and give it the name **L_Countries.** For its `SQL` property type the following:

```
SELECT countrycode,
countryname
FROM countries
ORDER BY countryname;`
```

Only the two fields **countrycode** and **countryname** are needed for our purposes.To use **L_Countries** we have to drop another **TDataSource** on **FrmAuthorEdit.** Name this datasource **listSource,** and set its **DataSet** property to point to **L_Countries.**
Your form should look something like Figure 10.

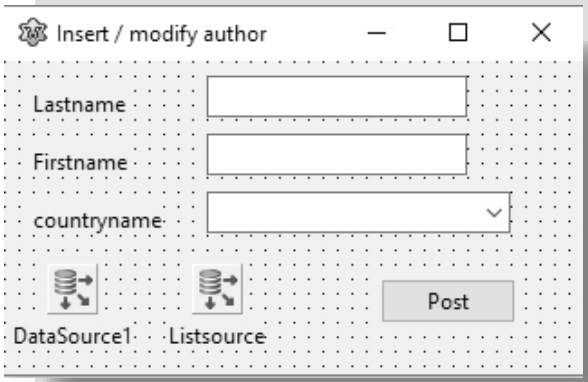


Figure 10: three data aware

Set **`DBLookupCountry.ListSource`** to **listSource.**
Set **`DBLookupCountry.KeyField`** to **countrycode.**
Set **`DBLookupCountry.ListField`** to **countryname.**

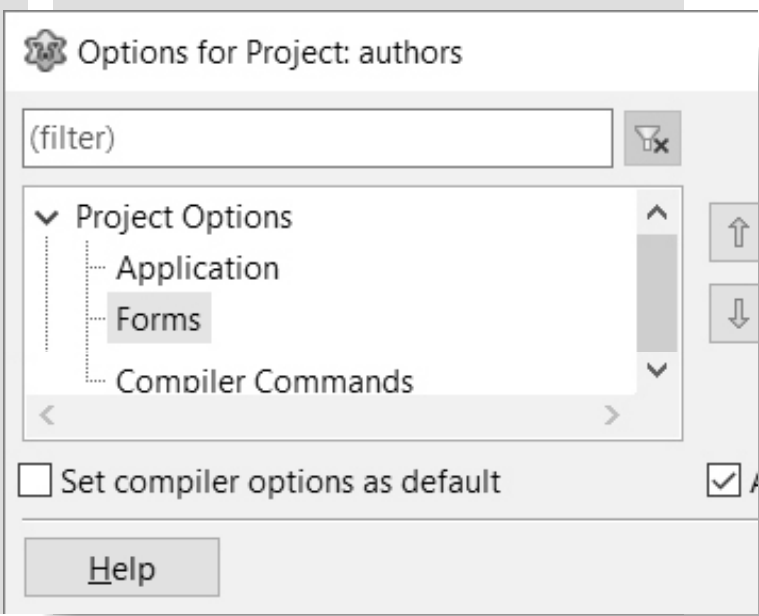
This property setting is very important, informing the application that the name shown in the **countryname** column is directly related to the value in the **countrycode** field for that record.
Thus when edited, the corresponding **countrycode** value will be saved in the database.
Double-click **BtnPost** to generate an **OnClick** handler, and complete it thus:

```
DataModule1.SQLAuthorsEdit.Post;
```

MANAGING THE DISPLAY OF THE EDITING FORM

When should **FrmAuthorEdit** be shown?
By default Lazarus assumes that every form you add to your application should be created automatically.
This is unhelpful for most forms with database connections, because this leads to activation of all the dataset(s) associated with all the forms (**in the absence of any other approach**).
However, you don't necessarily want every table in your databases opened at the outset, at form creation. Some databases like **Firebird** use a kind of locking which means that a record cannot be used to make a change to another record when the record is locked.
We need to manage the creation of our application forms more carefully.

This is easy to do via the Project Options. Open the Project Options dialog **(Project ➔ Project Option**s) and select the **Forms** node which opens a page with two listboxes on the right (*see Figure 1*1).



The left listbox lists all auto-created forms.
Select **FrmAuthorEdit** (*which contains the* **TDBLookupComboBox**) and click the right-arrow toolbutton to move the form to the listbox on the right which means it will **NOT** now be auto-created at application start-up, but is an **"available"** form, i.e. it is declared, but not created.
Click the **OK** button to accept this setting and close the Project Options dialog. Now we need to adapt the main form so we can show **FrmAuthorEdit** via a toolbutton click, or via a double-click on the DBGrid viewer.
A simple way to achieve this is to write a **`StartScreen method`** that takes a single integer parameter indicating whether an **insert-new-record** or an **edit-existing-recor**d scenario applies. Add **uAuthorEdit** to the uses clause of main.
Add a public **`FID: Integer;`** field variable to the **`TFrmAuthorEdit`** class in **uAuthorsEdit.**
Add a private **`StartScreen(const aID: Integer)`** procedure to the **`TForm1`** class in the **main** unit.In the implementation of the unit complete the **StartScreen** method as follows:

```
procedure TForm1.StartScreen(const aID: Integer);
var
 frm: TFrmAuthorEdit;
begin
 frm := TFrmAuthorEdit.Create(Nil);
 try
  frm.FID := aID;
  frm.ShowModal;
 finally
  frm.Free;
  DataModule1.SQLQuery1.Refresh;
 end;
end;
```

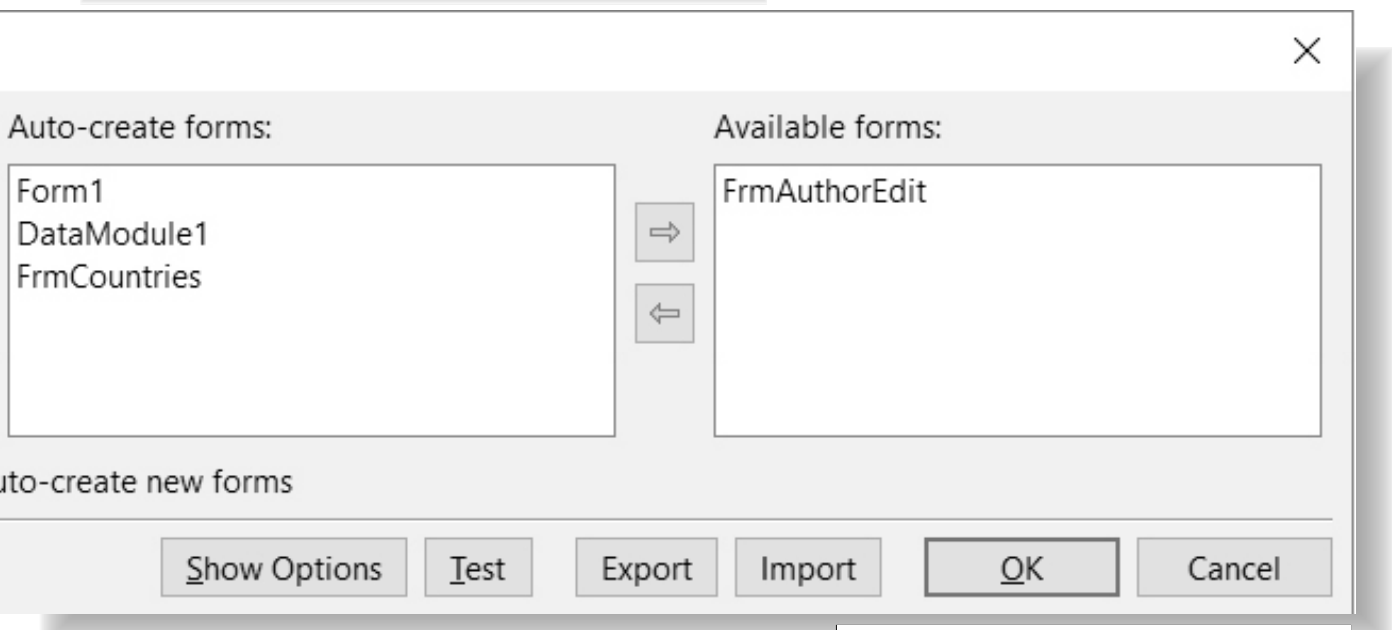


Figure 11:

The `aID` parameter indicates whether a record is located in the table. Giving the parameter a value of **-1** indicates that the fields in the edit form are empty and can be filled as a new record (*see Figure 12*). **Inserting** a **NEW** record:
`StartScreen(-1);`

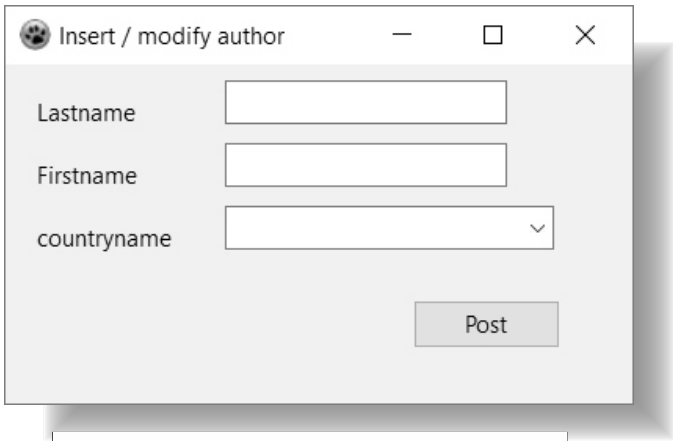


Figure 12:

Editing an **existing** record:
`StartScreen(Datamodule1.SQLQuery1.FieldByName('id').AsInteger);`
Here the editing form should be filled with the existing record values *(see Figure 13).*

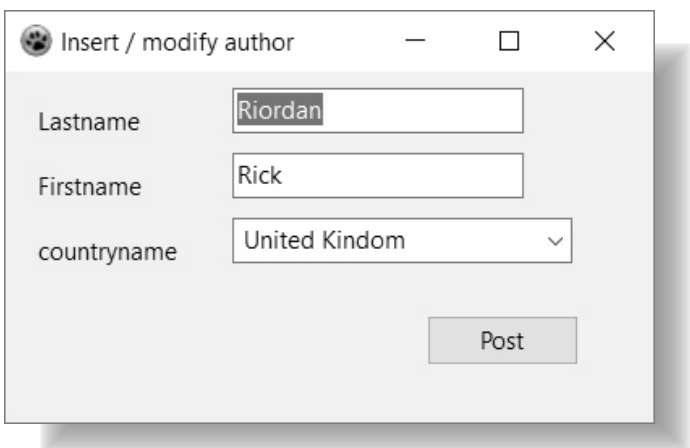


Figure 13:

Two toolbuttons must be provided on the main form for **inserting or updating** records.
We also want to provide the functionality that **double-clicking the DBGrid** should open the editing form too. Drop a **TToolbar** on **Form** and add two tool buttons to the toolbar, **ButtonNew** and **ButtonEdit.Set ToolBar1.ShowCaptions** to True.
Set **ButtonNew.Caption** to **New.**
Double-click **ButtonNew** to generate an **OnClick** handler for it which should contain this line of code:
StartScreen(-1);

Set **ButtonEdit.Caption** to **Edit.**
Double-click `ButtonEdit` to generate an **OnClick** handler for it which should contain this line of code:
StartScreen(DataModule1.SQLQuery1.FieldByName('id').AsInteger);

Generate an **OnDblClick** handler for **DBGrid1** and insert the same code given above for **ButtonEditClick.StartScreen** enables both the toolbutton **OnClick** events and a double-click on the DBGrid to show the edit form.
Now we have a small demo where we can run a **SQLite** database with **SQLdb.**
It's easy to change the database to **Firebird, MySQL** or another supported database in Lazarus.

A future article will extend the present example with detail tables, and offer other tools to make **SQLdb** work well for you.

Figure 14:

TSQLDBLIBRARYLOADER`: OPTIONAL DATABASE LIBRARY SELECTION

Some time ago a helper component for **SQLdb** was introduced to assist in finding the required client library. This is helpful when deploying your application away from the development set-up, since database applications depend on the presence of a specific library.
The helper is called **`TSQLDBLibraryLoader.`** When dropped on a form or data module it allows you to connect the appropriate client library to the database.
It is a component that searches for the chosen library **firstly in the Lazarus installation path**.
If the library is not found there, the system library path is searched *(on Windows this is the system directory)*.
If the required library remains unfound, no connection is established, then an error message is shown.In order to use **TSQLDBLibraryLoader** with **SQLite** as in the above example, **sqlite3.dll** *(or the equivalent Linux library)* must be present.
Provided this client library is installed in the correct directory, the database can be connected. **SQLdb** has several connection components which make connecting to specific databases possible:

- **`TMSSQLConnection`** (Microsoft SQL Server)
- **`TSybaseConnection`** (Sybase)
- **`TPQConnection (Postgres)`**
- **`TOracleConnection`** (Oracle)
- **`TODBConnection`** (Various databases with ODBC support)
- **`TMySQL`**<version>Connection (MySQL database. <version> = releases from 4.0 to 5.7)
- **`TSQLite3Connection`** (SQLite)

These components all descend from **`TSQLConnection,`** but the **ConnectorType** property is not available. In the example above if you used **`TSQLDBLibraryLoader`** you would set the library loader's **ConnectionType** property to **SQLite3,** and you would need to set its **LibraryName** property to the full path and filename of the required library.
Setting its **`Enabled`** property to **`True,`** sends a message to the **`TSQLite3Connection`** instance to ensure it uses this specific library, enabling a connection to be established to the database.
A word of warning: before the Lazarus 1.8 release there was a bug which caused the library to be loaded too soon. This resulted in an error and could cause problems like a restart of the IDE when your application files are loaded. If you still use such an old version of Lazarus you'll need this workaround: **`(http://wiki.freepascal.org/TSQLDBLibraryLoader)`**

RUNNING THIS ARTICLE'S SOURCE CODE

In order to compile and run the source code supplied you need to unpack it in a new directory. Then open the **authors** project in the Lazarus IDE. Open the **DataModule** unit, and select **SQLite3Connection1.**
In the **Object Inspector** click the ellipsis button to open the **DatabaseName** property editor, and locate the local SQLite database file distributed with the source.
It is called **authors.db3.** Once the absolute path and filename of this database is correctly entered for your local set-up, you should be able to compile the project without errors.

INTRODUCTION

This article is divided in to two main sections. The first section is about Apple and its future. The second part is about installing.

GENERAL ABOUT APPLE

POLICY FOR THE FUTURE:

The future of the Mac might be [Apple-designed CPUs]**(https://www.macworld.com/article/3315097/macs/macbook-a-series-chip-2020.html)**, but the **present** of the Mac is Intel. Apple is increasingly in charge of its own destiny, adding features like the T2 chip to Macs to handle as much of the system's security, encryption, and other miscellaneous tasks as it can. Still, we expect Apple to ship most or all of its laptops and desktops this year with Intel inside.

The chip giant hasn't yet announced the specific processors it will ship this year, but it **has** delivered a few sneak peaks at its roadmap. Here's what we know about what Intel is cooking up for 2019, and how it might impact the Mac. Apple has finally sent out invitations for its semi-annual fall event, that expects to be loaded with new products: an all-new iPad Pro with USB-C, slim bezels, and Face ID. We may also see a new Mac mini allegedly geared at pro users, refreshed iMacs, and a new low-priced MacBook.

I'm most intrigued by the MacBook. Not since the days of the polycarbonate iBook G4 and original MacBook has Apple sold a true budget notebook. The current MacBook Air costs $999, but there are a lot of compromises there: a non-retina screen, outdated processor, and aging ports. And above that is the plain MacBook, which is hardly a bargain with its $1,300 starting price.

Macs

Apple finally updated the iMac in March 2019, and added some new upgrade options to the iMac Pro at the same time. But there are still some Macs that are due for some love.

MAC PRO

Apple announced a new Mac Pro on stage at WWDC 2019. It returns the look of Apple's most powerful computer to the look of the 'cheese grater' first generation Mac Pro after the second generation 'trash can' version that has just been retired.

Available with 8-28 cores and specs that elevate it above the iMac Pro, the Mac Pro will go on sale from Autumn.

NEW APPLE DISPLAY

Alongside the new Mac Pro, we expect that Apple will launch a new display. We don't know much about what to expect other than it will support Thunderbolt 3/USB Type-C, and is said to be 6K and measure 31.6in.

MACBOOK

Apple left out any upgrades for its MacBook laptop when the covers were pulled off the new MacBook Air in October 2018. This has led to questions whether the slimline device will be discontinued due to the arrival of the Air, which is more powerful, cheaper, and features a larger display, while the MacBook is merely a lighter, low-powered machine.

The most obvious areas of improvement for a MacBook upgrade would be a move to a newer generation of processors, most likely Intel Amber Lake Y-series, the introduction of Thunderbolt 3, and a price drop to bring it down to the €1.000 that would differentiate it from the MacBook Air.

Unfortunately it seems that moving to the third generation of Apple's controversial Butterfly keyboard wasn't enough to stop the keyboard problems many users experienced with the first and second versions, Apple has since extended the program whereby they will fix Macs with faulty keyboards. Maybe Apple will update the MacBook with a new keyboard, but it seems unlikely given that the MacBook Pro update in May 2019 didn't do much to fix the problem.

NEW MACOS CATALINA: RELEASE DATE AND FEATURES

Apple's update to macOS for 2019 will be known as **CATALINA** and it will include a way for developers to port their iOS apps to the Mac, as well as lots of other exciting features. Apple has announced the details of the next version of MacOS including its name: Catalina, and when it will be released: the autumn.

Catalina will bring the demise of iTunes, in its place three apps that iPhone users will be familiar with from iOS: Music, the TV app, and Podcasts. As for how you will manage syncing if you wish to sync with your Mac, this will be done via the Finder. Apple also hopes that developers will be bringing their iOS apps to Catalina.
Other apps getting a look in this time round as Photos, Safari, Mail, Reminders, and ScreenTime. Apple had confirmed at WWDC 2018 that it would be making it easier to port apps from one OS to the other in 2019.

INSTALLING LAZARUS ON A MAC

installing LAZARUS on a MAC is not particularly difficult but it is critical that you do the install in the correct order. Skipping steps will almost certainly lead to disappointment. In brief, here is what you can do. The detailed instructions assume a recent version of macOS on you MAC, a recent version of XCode from Apple and recent version of Lazarus.

In general, this is about using both the **CARBON** and **COCOA** Widget Set. While Carbon may still be seen as a little more stable, as of release 2.0.0 the 64bit Cocoa is very close and certainly should be considered. Carbon is intentionally (by Apple) limited to 32 bits and we know the next release of MacOSX will probably not support it.

Figure1: The Application overview Find the Xcode.app

1. DOWNLOAD AND INSTALL XCODE.

http://wiki.lazarus.freepascal.org/Installing_Lazarus_on_MacOS_X

You need the Apple Developer tools, which are a part of the Xcode development environment. They can be installed from the original macOS installation disks, in which case you will have to update it. Or a copy downloaded from the APPLE DEVELOPER CONNECTION (ADC), which requires free registration:
http://developer.apple.com
Download the XCode file, it will end up in your Downloads directory as a zip file. Right click it. It is unarchived into your Downloads directory. You may be happy with it there but maybe not. Other users will see the path to it but be unable to use it. And it is untidy there, so move it and then tell xcode-select where it is moved to (in a terminal).

```
mv Downloads/Xcode-beta.app
/Developer/
sudo xcode-select -s/Developer/
Xcode-beta.app/Contents/Developer
```

But if you are new to this maybe you best leave the original way of placing it in to downloads.Then you do not make the last step.

1.1 INSTALL THE GLOBAL COMMAND LINE TOOLS FOR XCODE.

This is shown here as a separate step because it is a different separate step in addition to Step 1. **Don't confuse this with the internal XCode command line tools that the XCode GUI will tell you are already installed.** Lazarus cannot use those XCode internal command line tools, so do the following. Open up a terminal:
Open **Finder --> Applications--> Utilities**
There you will find the terminal: type
xcode-select --install
you need to install the SDK headers as well:
open
/Library/Developer/CommandLineTools/Packages/macOS_SDK_headers_for_macOS_10.14.pkg
Once you have installed all this you can start with the next step:

2 INSTALL FREE PASCAL COMPILER, FPC SOURCE AND LAZARUS

Get and install FPC and FPC Source. A compatible fpc (and source) **must be installed before you install Lazarus**. Download Freepascal. This will come in an precompiled version. So wont have to that.
You need to download and install all three packages fpc, fpc-src and lazarus.

[fpc-3.0.4.intel-macosx.dmg
[fpc-src-3.0.4-macosx.dmg
[lazarus-2.0.2-i686-macosx.dmg

fpc
- the Compiler, some command line tools, base units and non visual components like database access

fpcsrc
- the sources of fpc and its packages, needed for code browsing

lazarus
- the IDE, visual components and help files. After that you will have to do the followings steps.

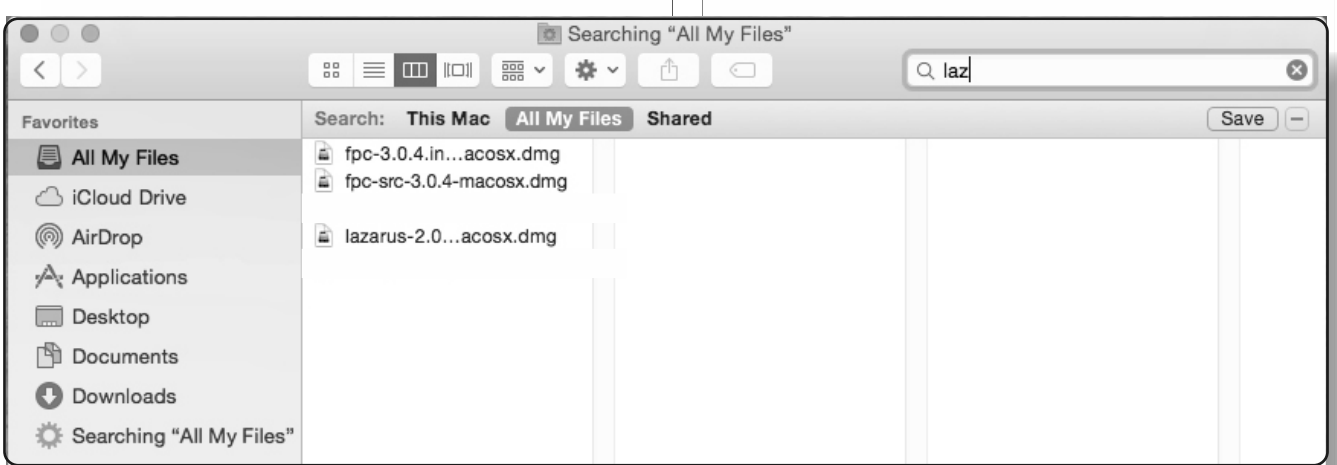


Figure 2: The files that are downloaded overview

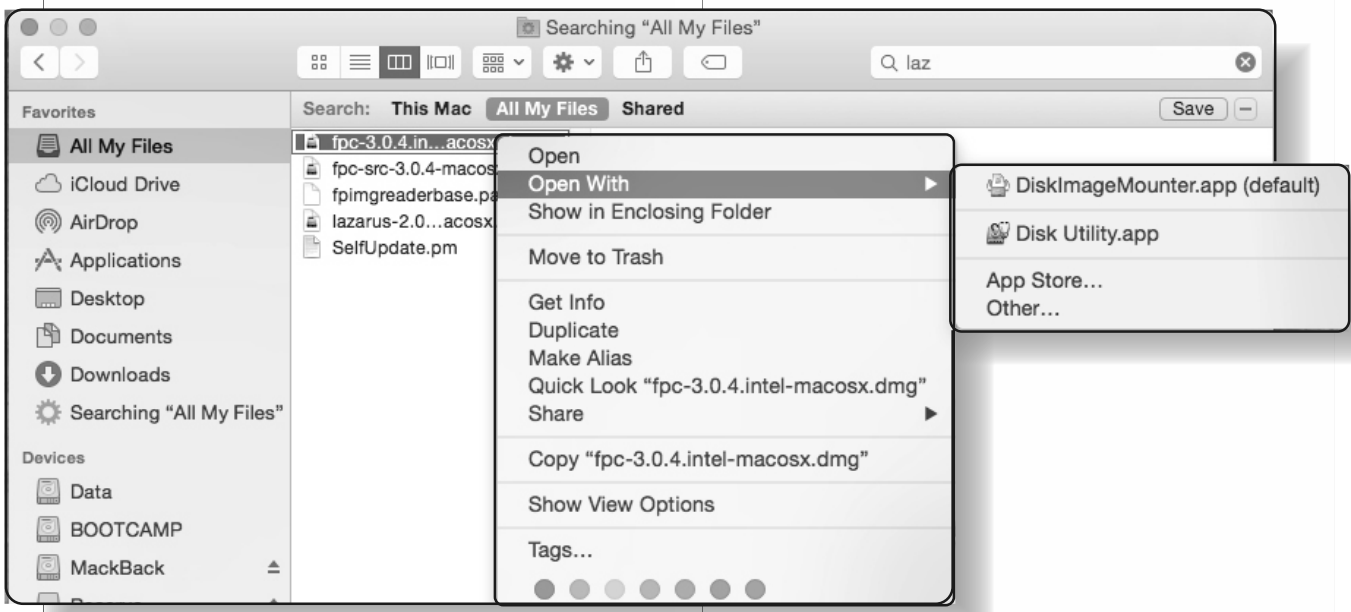


Figure 3: The files can be mouted to disk so by the default Disk Image Mounter

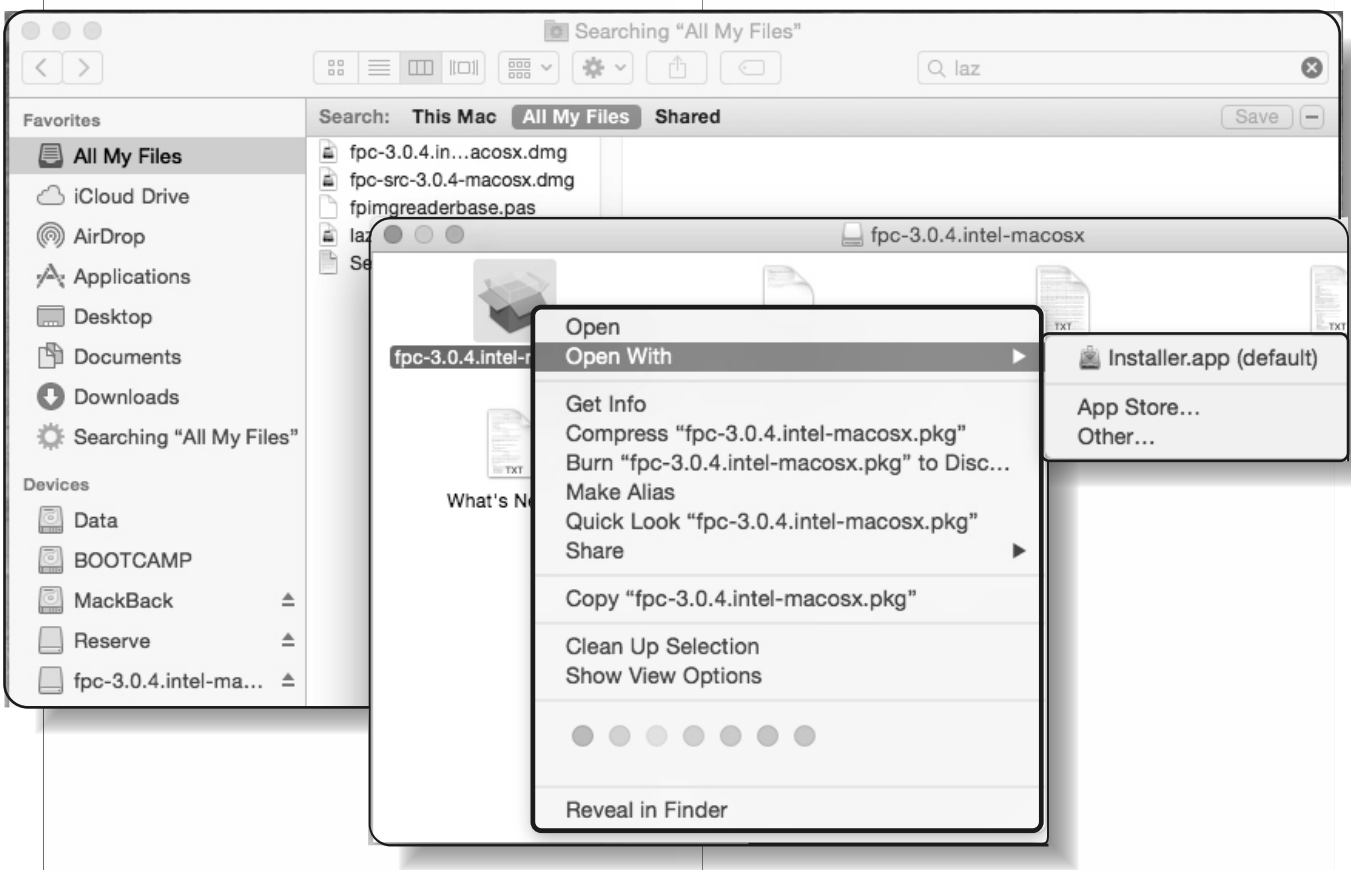


Figure 4: After that they can be installed

Figure 5: Accept and open

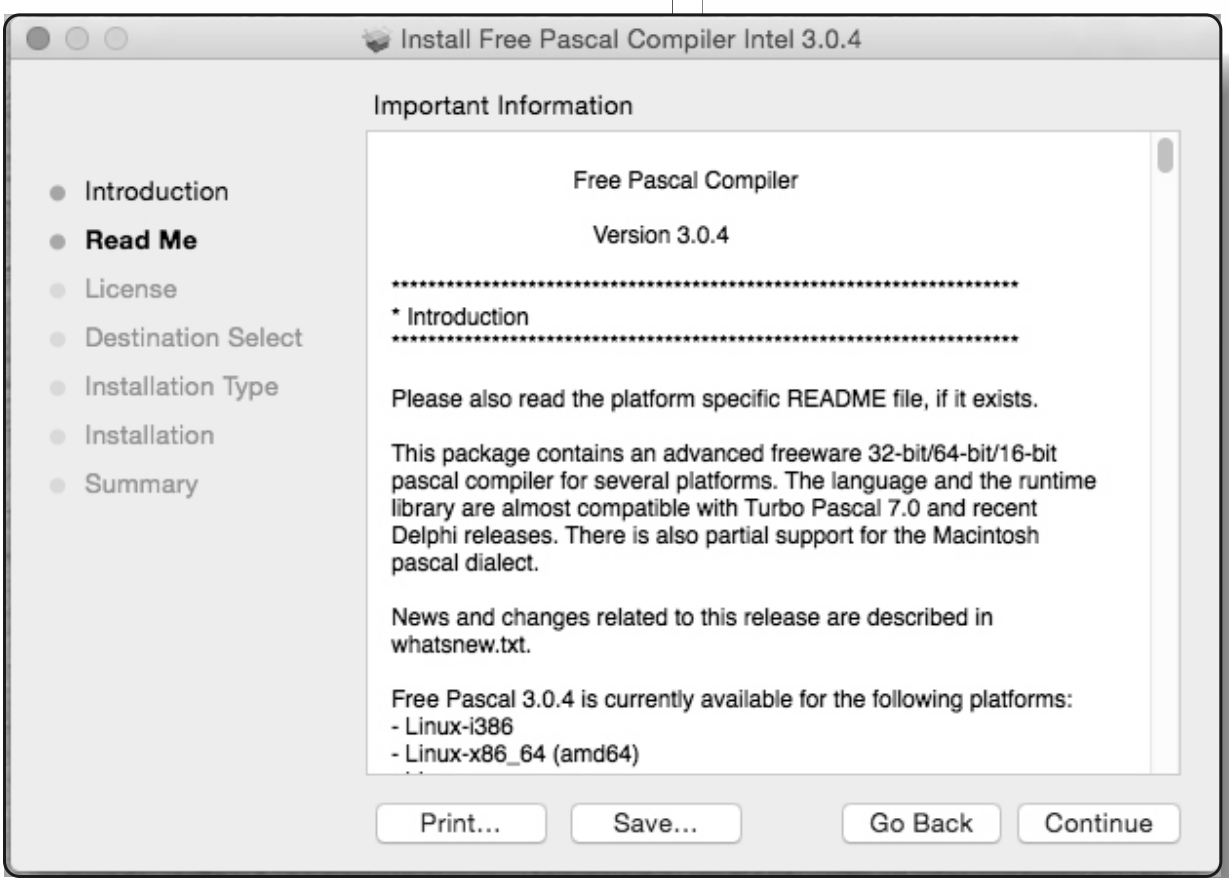


Figure 6: Read carefully

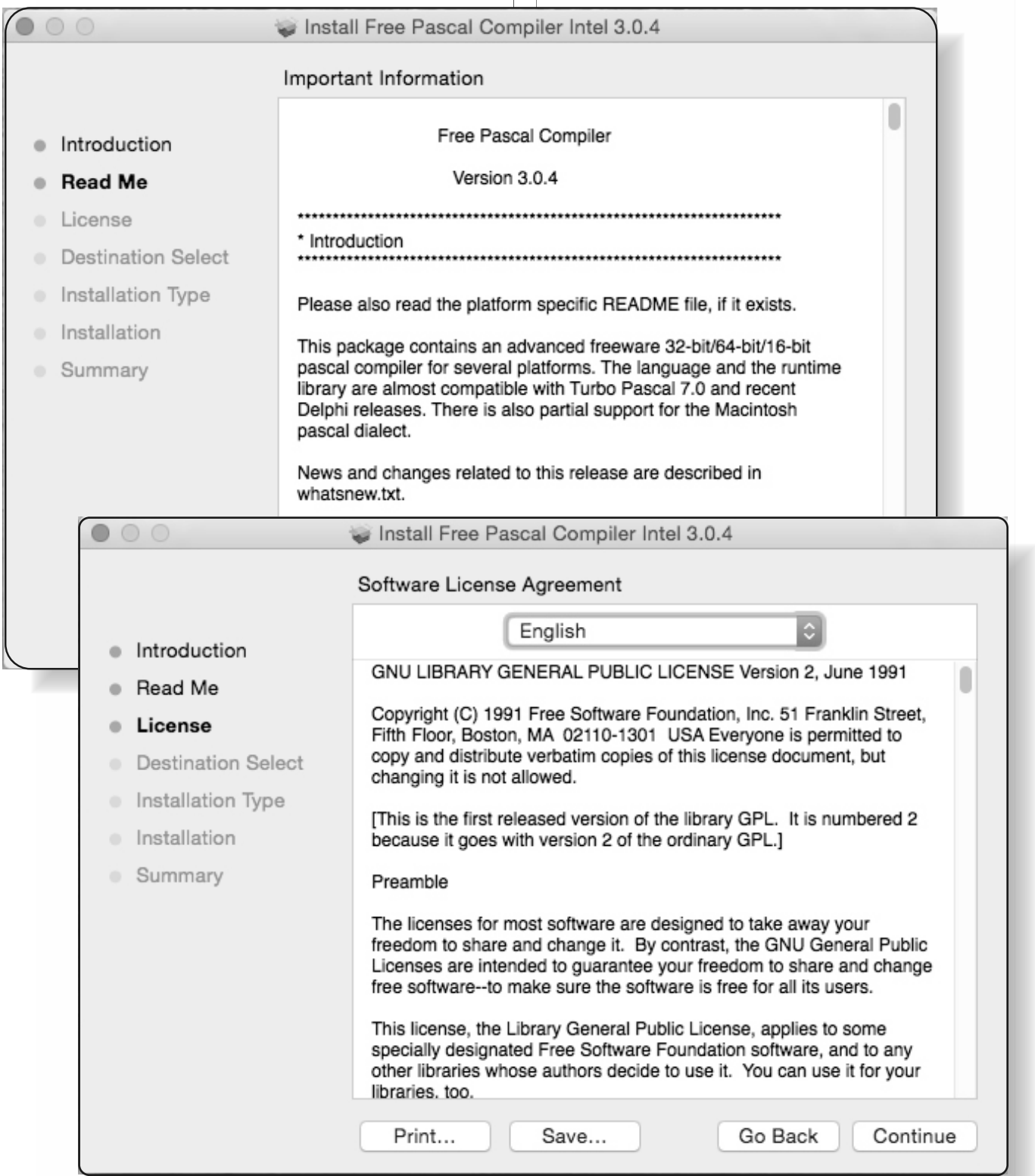


Figure 7,8: Just continue

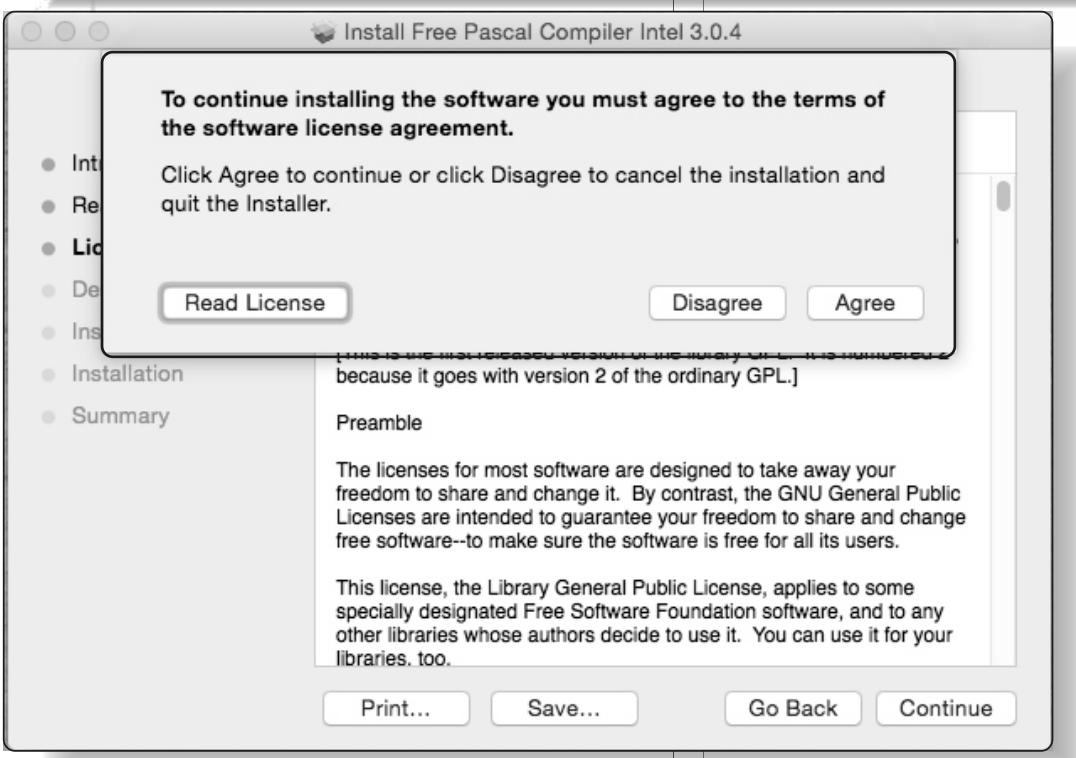


Figure 9: Agree

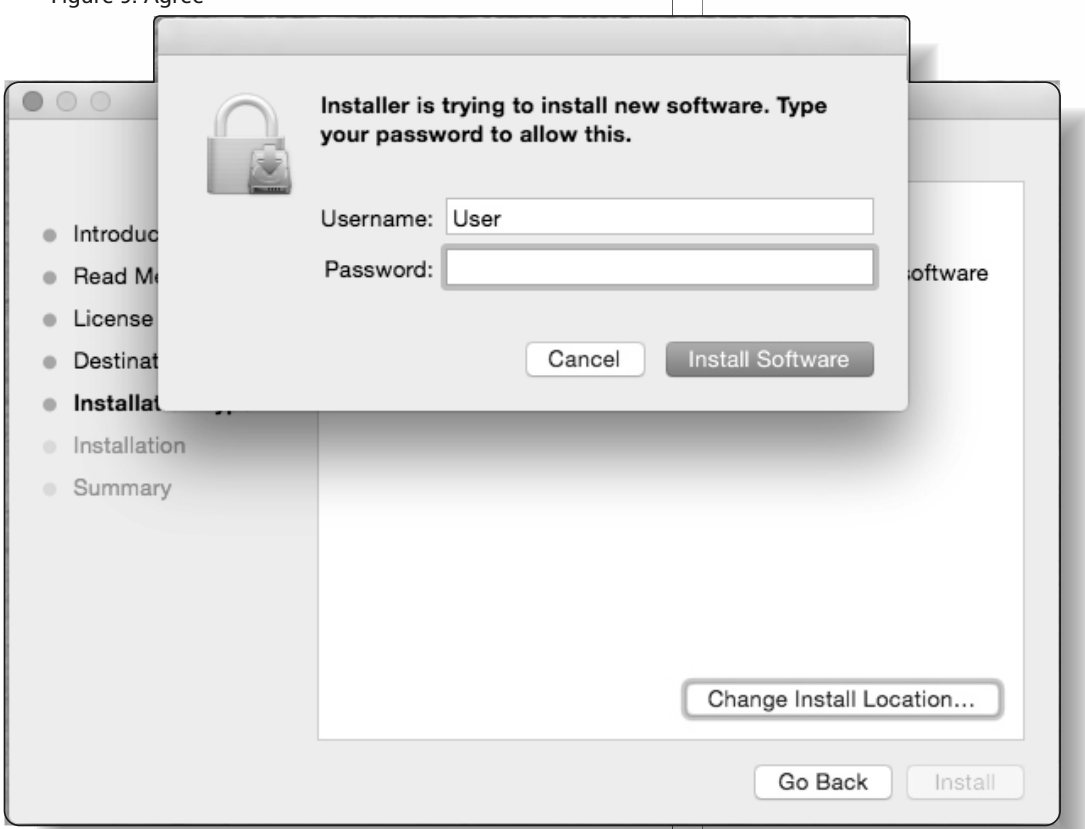


Figure 10: Enter your password

Figure 11:
Lazarus becomes visible in the Applications Group

You have to repeat this with each installation of the mentioned files. Then its done: You will see Lazarus as installed App.

Uninstalling Lazarus and FPC from the Mac

To uninstall Lazarus there is no macro or uninstall file that does it. It can best be done by hand. You do not need a special app that will do it for you.

If you want to uninstall the following **10 easy steps** are needed:

A

Use this only if you want to delete everything of your installation for Lazarus. If you have several installations you need to do that differently.

1. **Go to finder open new Finder window.**
   See Figure 12

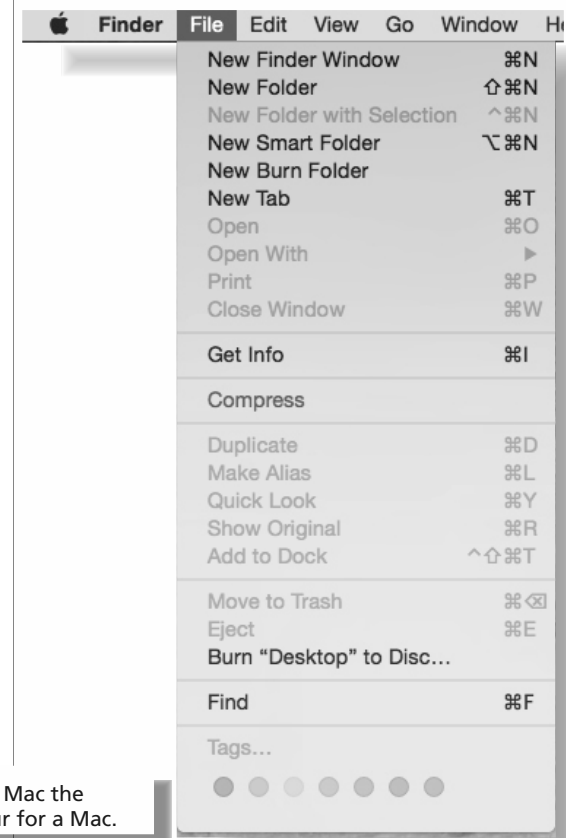


Figure 12: If you click somewhere in your desktop of the Mac the information file tab will change. This is normal behaviour for a Mac.

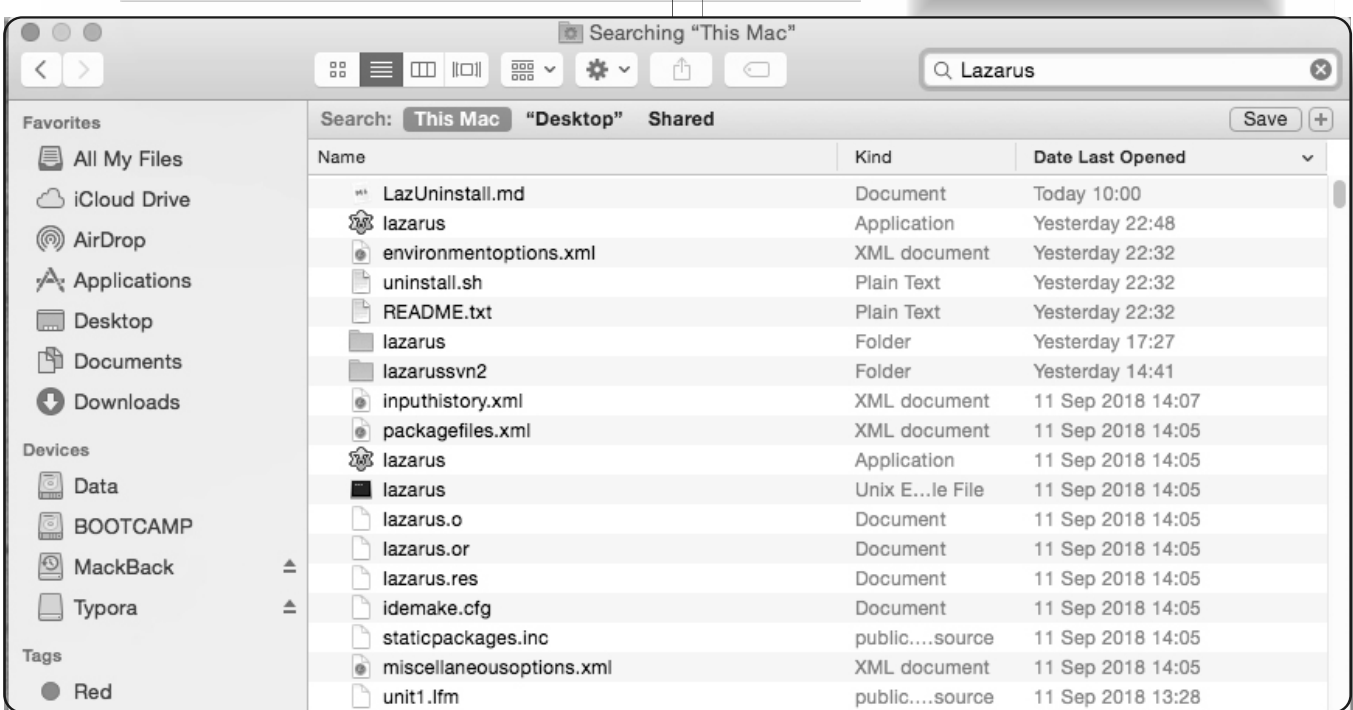


Figure 13: Type in Lazarus and This mac becomes an option.

2. **Select: list view at the top to have a better overview.**
   The window gives an enhanced overview of the way you can show your files.
   Try and use what ever you like, here we used a list view.

3. **Make sure you select This Mac at the left top Colum.**
   This Mac is only the name of the Machine.

4. **After adding the name**
   in the search box you will see that for searching special selections come available. Since we want to remove all the files belonging to the installation of Lazarus, type **Lazarus.**

5. **After that select the second item:**
   „This Mac" where you want to be searched. It could also be an other name like My Mac. The list appears is a drop down list: choose **System Files**.

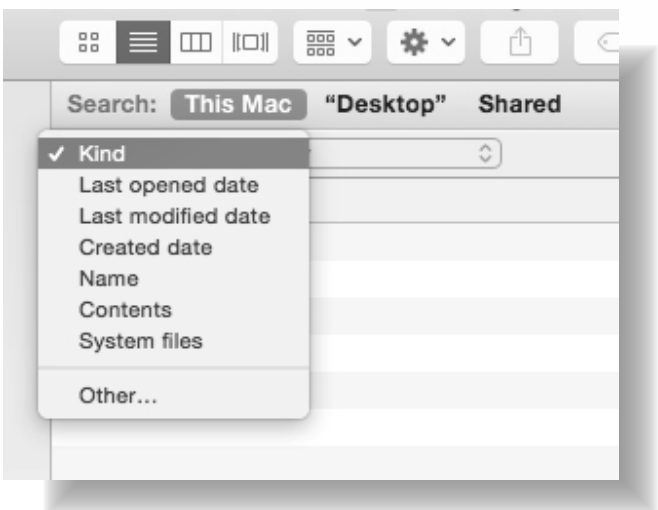


Figure 14: Listbox with the selection Kind becomes available.

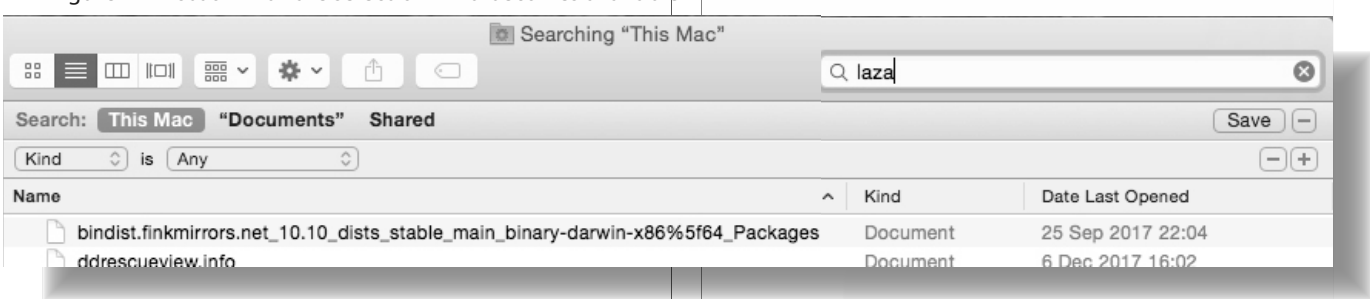


Figure 15: The Kind has been set.

6. **At the right edge there are two small buttons,** (figure 15).
   a button with **Save** as caption and an other button having a small **plus (+) sign**. If you press the **+sign** it will reveal some extra options that are important. A new line will appear that has the option button **Kind.**

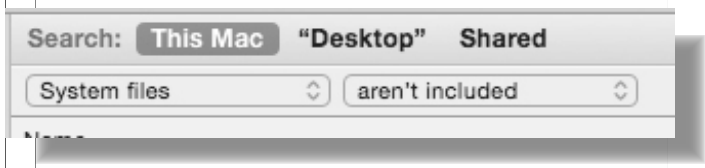


Figure 16: See the changes, Kind becomes System files and those aren't included

7. **If you click on Kind it you will get the options list to select System Files.**
   Then it changes to that setting.

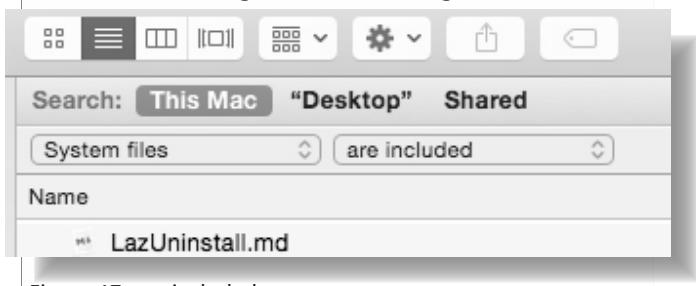


Figure 17: are included

8. The window now will change and tell you: aren't included.

9. **Change that position to:**
   are included by clicking that option.
   You have now all the files gathered that are used with Lazarus in one big overview.
   (If you select all Files it normally does not include system files). Be careful with this, because removing items can damage files on your machine. That is why you will have to give by deleting your password.
   (Move to Trash)

10. **Select them all and delete them by right clicking.**
    A dropdown list becomes available.
    By pressing **command + 4** (⌘+4) you can select them all and then delete them.
    (Move to Trash) Using this command means you will have to give your Password.

Use this only if you want to delete everything of your installation for Lazarus. If you have several installations you need to do that differently.

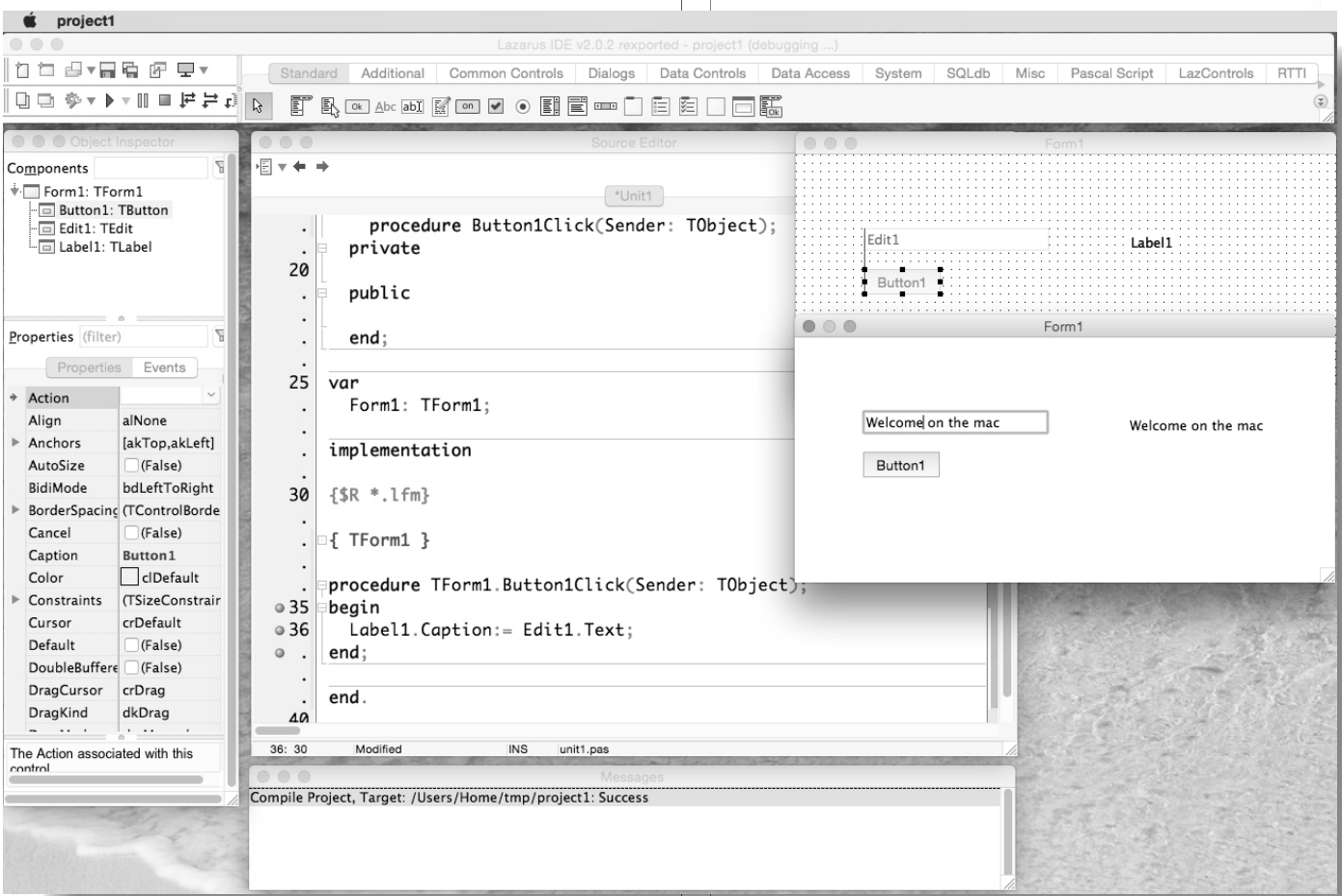


Figure 18: Lazarus working in a project on a Mac

B

This is to uninstall only the version you had last installed if you have it installed under the name Lazarus.

Open a console like you did before and then type:
`sudo rm /Developer/lazarus.`

Figure 19: Finding the Terminal

Figure 20: The Terminal is once opened White, not like the icon

If you go to
Applications ➔ Utilities ➔ Terminal App
you will find the Icon for the console.

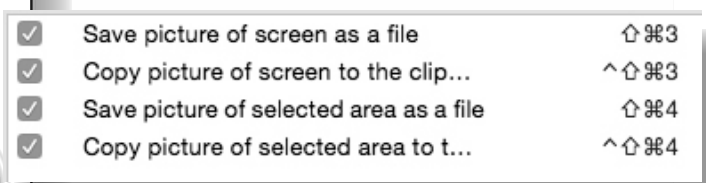
`rm stands for remove.`

Figure 21: The Terminal and code to remove Lazarus

After this your Lazarus has been uninstalled.
At this end we show you the Mac Keyboard Symbols:

- ⌘ is the **Command** key.
- ^ is the **Control** key.
- ⌥ is the **Option** (alt) key.
- ⇧ is the **Shift** key.
- ⇪ is the **Caps Lock** key.
- fn is the **Function** key.

There is something else very interesting for you: Commands that will allow you to create pictures without installing a special App. See figure 7



I'm always happy to get input from users of kbmMW about features they would like kbmMW to support.

A short while ago, a user sent an email, stating that he was being asked to provide XML-RPC and possibly JSON-RPC support for some of his clients.

XML-RPC is a fairly old standard made by an employee at Microsoft back in late 80's. I have not spent much time actually considering supporting it until now, because I have assumed it to be practically out of use, beaten mostly by REST these days.

However digging into it a bit, I find many scenarios where XML-RPC is being used, so of course kbmMW should support it

XML-RPC is fairly easy to implement as it supports only a very few well defined datatypes and that it is using positional arguments, which makes it integrate nicely with kbmMW's own RPC style.

One datatype XML-RPC supports which, at least in the old days before kbmMW v5, did not really play well with kbmMW, is a key/value style datatype.
However v5 comes to the rescue, since it is way more clever regarding what can be used as arguments and results.
Now not only simple types can be handled but just about any type, including the nifty Object Notation ones, including **`TkbmMWONArray`**, **`TkbmMWONObject`** and **`TkbmMWONNative`**. In this case XML-RPC will forward the key/value data as a **`TkbmMWONObject`** instance.

Similarly one can return **`TkbmMWONObject`** instances from the methods called and it will be understood correctly as a key/value response datatype.

Another Remote Procedure Call method is called JSON-RPC. As the name implies, JSON-RPC use JSON instead of XML as the container format. That also means that anything that can be expressed as JSON can be used as arguments or results in JSON-RPC. Fortunately again the kbmMW Object Notation framework fits in nicely.

Let us define an XML-RPC server.

It is as easy as adding a TCP based server request/response transport like **`TkbmMWTCPIPIndyServerTransport`**, and setting its StreamFormat property to XMLRPC and include **`kbmMWXMLRPCTransStream`** in the main units uses clause.Now the server will accept requests like this:

```
POST http://127.0.0.1/smartdemo

<methodCall>
  <methodName>addnumbers</methodName>
    <params>
      <param>
        <value><i4>11</i4></value>
      </param>
      <param>
        <value><i4>2</i4></value>
      </param>
    </params>
</methodCall>
```

which will result in responses like this:

```
<methodResponse>
  <params>
    <param>
      <value>
        <double>13</double>
      </value>
    </param>
  </params>
</methodResponse>
```

Remember that the request must be given with the mimetype/Content-type text/xml to be valid. kbmMW supports both UTF-16 and UTF-8 charset setting.

The last part of the URL (after the last /) must be the name of the service optionally with a dot (.) and the version of the service.

Hence SmartService.1.0 is valid as a way to call the service named SmartService version 1.0
If the dot is omitted, the default version will be used, as is normal for kbmMW

If you instead set StreamFormat property to JSONRPC and include **`kbmMWJSONRPCTransStream`** in the main unit uses clause, you will have asked kbmMW to understand JSON-RPC v2.0. kbmMW is almost 100% compatible with JSON-RPC except that it currently will not accept batch statements (multiple calls grouped in a JSON array).
Let us look at the same request in JSON-RPC format.

```
POST http://127.0.0.1/smartdemo

{"jsonrpc": "2.0", "method": "addnumbers",
"params": [42, 23], "id": 1}
```

Which will result in this response

```
{"jsonrpc":"2.0","id":1,"result":65}
```

JSON-RPC supports named arguments where the arguments would be provided as a JSON object (key/value list) instead as an array of values. kbmMW will forward this as a TkbmMWONObject to the method being called. So in this case the method must be defined to only receive one argument.

Eg.

```
POST http://127.0.0.1/smartdemo

{"jsonrpc": "2.0", "method": "addnumbers", "params": {"val1":42, "val2":23}, "id": 1}
```

The method will have to work like this to get access to the named arguments:

```
function TMyService.AddNumbers(const Arg:variant):integer;
var
  co:TkbmMWONCustomObject;
  o:TkbmMWONObject;
begin
 if TkbmMWRTTI.GetkbmMWONCustomObject(Arg,co) then
 begin
  if co.IsObject then
  begin
    o:=TkbmMWONObject(co);
    i1:=o.AsInt32['val1'];
    i2:=o.AsInt32['val2'];
...
  end;
 end;
end;
```

The class TkbmMWRTTI contains a couple of other nice methods that makes it easy to extract other complex values from the variant arguments:

```
function TkbmMWRTTI.GetkbmMWDateTime(AVariant:variant; var ADateTime:TkbmMWDateTime):boolean
```

and

```
function TkbmMWRTTI.GetkbmMWNullable(AVariant:variant; var AIsNull:boolean;
var AVariant:variant):boolean
```

The later will return the value of a kbmMWNullable contained in a variant, and if the value is actually null.
You can return a complex value like this:

```
Result:=Use.AsResult(akbmMWOnObject);
```

kbmMW will take care of freeing the object when it is no longer needed.

All this will be available in the upcoming kbmMW release.

You can read more about XML-RPC here:
https://www.tutorialspoint.com/xml-rpc/index.htm

and JSON-RPC here:
https://www.jsonrpc.org/specification

If you like what you see, please share the word.

barnsten

- development tools
- consultancy
- components
- training
- hands-on workshops
- support

Be the First to Get 10.3.2 and Save 15%! Also, get Enterprise connectors when you buy Enterprise or Architect (€ 1.057,-Value). Expand FireDAC & SQL to connect to Enterprise data sources spanning Accounting, CRM, Marketing, NoSQL, eCommerce, Social Networking and more.

Get Ready for 10.3.2! We're offering you a pre-release discount on RAD Studio, Delphi and C++Builder. Buy 10.3.1 now at 15% off and get the 10.3.2 upgrade upon release. Use this early bird special to get all upcoming product releases (including 10.3.2).

We cannot tell you everything about 10.3.2, but we are confident you will love what's coming. Lock in your savings now and enjoy 12 months of update subscription (That means you get all new releases that come out in the first year)!

Purchase Enterprise or Architect and maximize your productivity with Enterprise Connectors. Seamlessly connect your application to over 70 enterprise apps (€ 1.057,- value). Enterprise and Architect editions offer significant value over the Professional edition and your 15% discount can be combined with the free Enterprise Connectors offer.

https://www.barnsten.com/default/development-tools/rad-studio/rad-studio-professional?___from_store=nl

PREFACE

One of Delphi's forces (and weaknesses) has been the ability to do data binding between a TDataset and specially designed data aware controls. It made it very easy, in VCL, to quickly develop a datadriven GUI, although from purist views, the end result often ended up with a mix of view and model parts intermixed with control parts in the same unit, in one big spaghetti mashup.

Firemonkey is really cool

When Firemonkey came along, Embarcadero decided to no longer provide the **`TDataset/TDatasource`** based data binding, but instead provided a much cooler looking framework called live binding, which have additional flexibility but at the expense of a more complicated binding process. Specially in the first several versions of it, binding stuff were quite complicated, because various classes had to be instantiated, somewhat confusing properties set and all the stuff linked together by hand.

It led to lenghty explanations and posts attempting to explain the inner workings of live binding.

Later versions introduced a very nice (at least on the surface) live binding designtime editor, which made it fairly easy to link objects and properties together. Behind the scenes it however continued to rely on complex class instantiations, property settings and object hierarchies to work. But it was much nicer and easier to use…. at least for fairly simple forms with not too many controls in play.

I prefer making new apps using Firemonkey, simply because it has some design abilities that makes for a more modern looking application, and it also opens up executing those apps on various platforms, which in my belief has a life extending potential for the apps. In practice Firemonkey turned out fairly well, and gave nice sideeffects, where some platforms availability suddenly broadened the potential use cases for my applications. One application could sometimes work very well on both mobile platforms and as kiosk mode like Windows applications.

Although my workdays often exceeds 15 hours, I'm actually lazy by nature (*just ask my wife!*), and I absolutely despise having to make the same work twice. Hence Firemonkey came as a tool to enable my future laziness by being able to reuse with little extra effort.

Live binding problem #1

What I however discovered, was that although live binding seems beautiful on the surface, it quickly turned out to be quite ugly. The designtime drag/drop designer became exponentially slower and slower in use, making it unbearable to use for more complex forms typical in applications where business style grid like data is to be presented in a non grid way, hence requiring separate controls and such.

The binding is still fairly easy to do at designtime, it is just very very slow reacting and it is with a larger number of controls getting unusable in practice.Having a larger number of controls also impairs the overview in the designer, even though it does support some layer management. But it's just slow to use.

Live binding problem #2

The problems with live binding unfortunately do not stop there.
In a program where one want to separate view from model from control, it is virtually impossible to code in a nice way that enables speedy refreshing of data, while not having live binding affecting performance greatly.

For example, if using a **`TDataset,`** doing `EnableControls` and `DisableControls` have absolutely no effect on live bound controls. If your **`TDataset`** originally contained say 3000 rows, and you want to refresh those 3000 rows from a database or an application server, then due to the way live binding works, it will cause your poor `TDataset's Post` to run 9.000.000 record navigations due to Resync being called by Post, while large buffers has been allocated for **`TDatalink's`** which are the actual glue between the **`TDataset`** and the DB live binding source classes.

It makes an otherwise speedy interface crumble to an almost complete halt by several orders of magnitude.

So how to avoid that? Well one would think DisableControls would do the trick… alas it doesn't.

So you could stop using **TDataset's!** Yes… but **TDataset's** are actually very nice containers for storing data and the history of the data, specially when using kbmMemTable (*used by* kbmMW) as the storage. It makes for a very easy way to figure out what to `insert, update, delete` etc. when the time comes to that in the data's visual life time.
But there must be a way? Yes. I did find one…..
Completely disconnect the `TDataset` from the **TBindSourceDB** before the refresh operation, and reconnect it after. The problem with that, is that you now in most cases will have to make your model/control part that handles your data access to the database or application server, aware about the existence of a **TBindSourceDB,** so it can be disconnected and reconnected at the right moments.

Perhaps one can "handle" this "nicely" with some callback **IoC** techniques, but I really don't want to have to remember to circumvent structural design issues in code, every single time I'm going to write a new application. So it is in my opinion a mitigation of a problem, not a solution, one you have to remember to mitigate each and every time new data is going to be worked on.

What about avoiding the Resync?
Unfortunately that is not an option at all. It's part of **TDataset's** way of working.

Livebinding problem #3
Ok… I have to make the applications work, and the end users don't really care how it is working behind the scenes, as long as it works and looks pretty. So I bite the bullet and mitigate problem #2. But remember there is really no nice mitigation for problem #1 without entering attempting live binding between separate **TFrame** and other such stuff, that even more removes focus on the actual functionality of the application, to be able to handle all the plumbing to make live binding work seamlessly.

But then enters problem #3. What if I want to `refactor` the visual controls? What happens with the live binding? If your controls are deleted/moved to another form, your live binding disappears, and you will have to manually remember all the bindings you made at design time, and redo them later on after the refactoring. It makes refactoring the **G.U.I** a true **P.I.T.A.**
But then don't use the designer… make the binding in code! That would be a typical answer to both problem #1 and #3… and that takes us all the way back to how live binding was used from day 1…. having to remember and instantiate the correct object hierarchies, set the correct somewhat obfuscated properties … and pray. Making accessing and viewing the data much more cumbersome than I like.
I just frigging want to focus on the actual business aspects (*visual, algorithms and data*) of the application, not to have it drown in all sorts of stuff draining my mental capabilities, and in fact hindering my way towards delivering something stable, speedy and business wise feature rich to the customers.

Livebinding problem #4
Live binding is event oriented. That means, each and every time something sneezes in your app, it might trigger live binding code that may result in another part of your apps controls or data getting updated which in turn may trigger yet more events being fired. Most Delphi coders probably have tried the simple circular never ending event loop having one `TEdit.OnChange` enter data into another `TEdit.Text` which in turn also have an `OnChange` event handler, which is supposed to update the first `TEdits` Text property. It is an easy way to spend all your computers CPU power on one core and does not make for a nice GUI for your end users.

The idea about binding data to visual controls should IMO handle this issue, to ensure that it is easy to get your data presented, and to handle user input. Unfortunately using an event based setup, makes it very likely that you will end up in circular event trains which you again have to code your way out of to handle. Again taking time away from actually making business code.

"Well Kim, you have done nothing except complain. Do something about it then if you are so damned negative about live binding, grumpy ol' man!'

SmartBinding comes to the rescue
And so I did. I realised about a year ago, the above issues that affects my productivity in a negative way, and started to brainstorm and prototype a bit to see if I could find some solution. At first I was thinking about improving live binding, because it is already there… but I soon came to the conclusion that the design in itself was wrong, and there would never be a fix that solved all 4 above problems.

It needed a new thinking… a new approach. After some tinkering, I had the basic structure in place for what is now called kbmMW SmartBinding and which will be included as beta code in the upcoming kbmMW release.

The design goals for SmartBinding was:

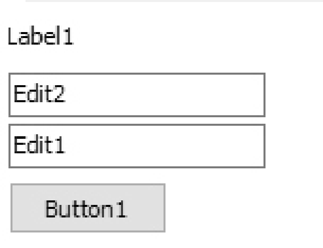
- Must be easy to use
- Must minimise or completely remove boiler plate code.(do you see the trend here?… kbmMW has since v5 been all about making things easier!)
- Must have good performance
- Must have a low CPU, memory and size footprint
- Must not result in endless circular event trains
- Should work with all sorts of data and controls
- Should be flexible and extendable
- Must be near real time
- Must be easily refactor-able
- Should play well with kbmMW's other features
- Should be usable even without using kbmMW's other features

All those goals has now been achieved in the current beta version.

SmartBinding is based on a poll/update strategy rather than a strict event oriented code bound methodology. Done right it fits well with the above requirements.

Let's see some action then!

Example – Simple property binding



We start out gently… Assume we have these controls on a form. We want to be able to type stuff in `Edit1 (TEdit)` and have it reflected in `Edit2 (TEdit)`, `Button1` and `Label1`, and we want to be able to type stuff in Edit2 and have it reflected in Edit1 (*which in turn further reflects to Button1 and Label1 as already mentioned*).
A simple binding case.. but fairly complex in an event driven binding context.
This is the code needed to bind:

```
Binding.Bind(Edit1,'Text',Label1,'Caption');
Binding.Bind(Edit1,'Text',Button1,'Caption');
Binding.Bind(Edit1,'Text',Edit2,'Text',[mwboTwoWay]);
```

I mean… that's ALL that is needed to bind those controls together!
The first 2 arguments are the source instance and property name, and the next 2 are the destination instance and property name.

What about the Binding instance…where does that come from? kbmMW SmartBinding default comes with an instantiated singleton **`Binding:TkbmMWBindings`** which can be used immediately. See later for more information.
The last line also includes an optional flag, that indicates that the binding is two ways, so changing one side will automatically change the other too.

Basically all `string, boolean, floating point, int64` and `integer` properties can be easily bound this way with auto conversion where SmartBinding make sure to automatically convert data between the different types as needed.
Other types of data can be bound too, but the source and destination properties will then have to be of same type (***there are ways around that too… see later***).

What about thread safety?

kbmMW SmartBinding maintain two pools of bindings, one where binding polling and updates can run completely async in a separate thread, and one where the binding polling and updates must run in the main GUI thread.
When you bind, SmartBinding automatically recognise if you are binding to and/or from **`TControl's,`** and assign those bindings to the appropriate binding pool, thus ensuring correct operation.

It is important to know, that each poll and update pass will be fully done before a next one will happen. The async pool will be immeidately started and run alongside the sync pool, but the complete pass will have to finish before being considered done and next update pass will be attempted according to the scheduled interval. See prologue for more information about the interval.

Example – Binding to and from record data

kbmMW SmartBinding can also bind to regular objects or even records of data. Just make sure that the data is constantly available for as long as the binding exists. For that reason kbmMW SmartBinding also contains easy unbind and rebind features.

```
type
 TData = record
  FData1:string;
 end;

var
 data:TData;
...
   Binding.Bind(Edit1,'Text',@data,TypeInfo(TData),'FData1');
   Binding.Bind(@data,TypeInfo(TData),'FData1',EditN,'
```

Now let's bind:

```
var
 bnd:IkbmMWBinding;
begin
   Binding.Clear;
   bnd:=Binding.Bind(lines,'Name',Edit1,'Text');
   Binding.Bind(lines,'Address',Edit2,'Text');
   if bnd.Navigator<>nil then
    bnd.Navigator.First;
end;
```

```
type
 TLine = class
 private
   FName:string;
   FAddress:string;
 public
   constructor Create(const AName:string; const
AAddress:string);
   property Name:string read FName write FName;
   property Address:
            string read FAddress write FAddress;
 end;

 TLines = TObjectList<TLine>;

var
 lines:TLines;
...
   lines:=TLines.Create;
   lines.Add(TLine.Create('Hans','Hansvej 1'));
   lines.Add(TLine.Create('Jens','Jensvej 1'));
   lines.Add(TLine.Create('Frederik','Frederikvej 1'));
```

```
var
 mt:TkbmMemTable;
 csv:TkbmCSVStreamFormat;
begin
   csv:=TkbmCSVStreamFormat.Create(nil);
   try
     mt:=TkbmMemTable.Create(nil);
     mt.LoadFromFileViaFormat('biolife.csv',csv);
   finally
     csv.Free;
   end;
```

And let's make the bindings to the edit controls (*you may be able to guess how it's done by now*)

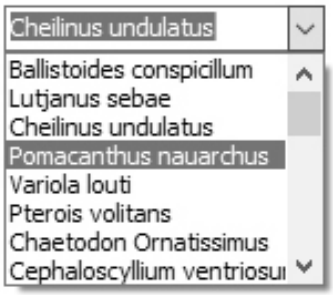
```
  Binding.Clear;

bnd:=Binding.Bind(mt,'Category',Edit1,'Text',[mwboTwoWay]);
  Binding.Bind(mt,'Species Name',Edit2,'Text',[mwboTwoWay]);
  if bnd.Navigator<>nil then
    bnd.Navigator.First;
```

Again exactly same way to bind. In this case we have told the bindings to be two way, essentially making `Edit1` and `Edit2` behave the same way as if it was the old dataware `TDBEdit` controls. Again we have optional access to the navigator and can easily navigate the dataset.

Example – Binding to a list box or combo box

Sometimes one want to use the source list/dataset (*or selected parts of it*) to populate a `TList` or `TCombobox` (or descendant).



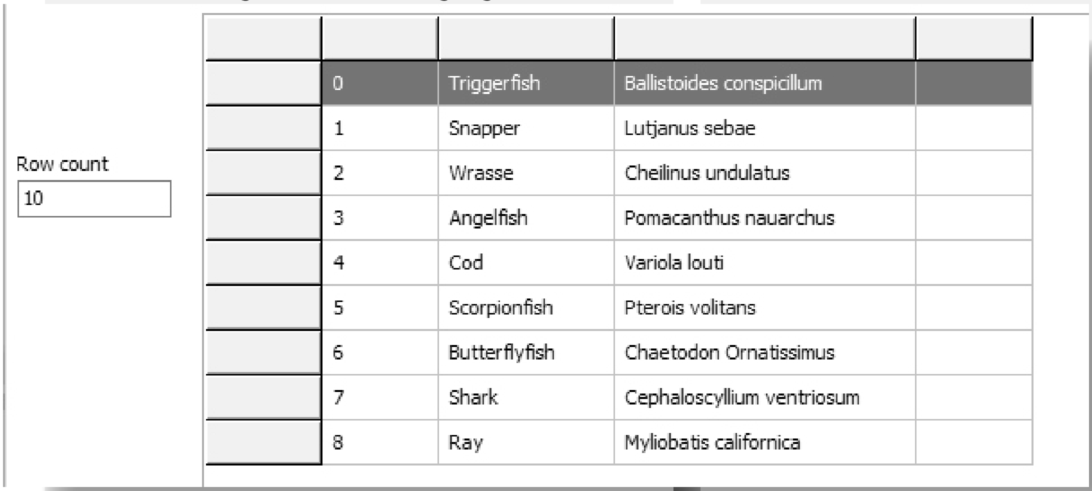
For this example we also want to synchronise the controls so selecting in one of them is automatically reflected in the other.

```
Binding.Clear;
Binding.Bind(mt,'Species Name',ComboBox1,'Items');
Binding.Bind(mt,'@',ComboBox1,'ItemIndex',[mwboTwoWay]);
bnd:=Binding.Bind(mt,'Common_Name',ListBox2,'Items');
Binding.Bind(mt,'@',ListBox2,'ItemIndex',[mwboTwoWay]);
if bnd.Navigator<>nil then bnd.Navigator.First;
```

First we bind one field from the dataset mt, to the items string list of the `TCombobox`, and we bind another field to the items string list of the TList. But what we have also done, is introducing the @ (*at*) indicator. When used as the source property, we ask `kbmMW SmartBinding` to refer to the position of the source list/dataset. So we bind the position of the source to the `ItemIndex` (position) of the `TList` and `TCombobox`. Further we have told **SmartBinding** that we want it to be two ways. Hence not only does changing the position in the source navigator update the position in the `TList` and `TCombobox`, but selecting something in the controls also automatically updates the source list/datasets position essentially in this case ensuring the two controls are in automatic sync with each other and their source.

Example – Binding to a grid

No SmartBinding without binding a grid…



	0	Triggerfish	Ballistoides conspicillum	
	1	Snapper	Lutjanus sebae	
	2	Wrasse	Cheilinus undulatus	
	3	Angelfish	Pomacanthus nauarchus	
	4	Cod	Variola louti	
	5	Scorpionfish	Pterois volitans	
	6	Butterflyfish	Chaetodon Ornatissimus	
	7	Shark	Cephaloscyllium ventriosum	
	8	Ray	Myliobatis californica	

In this case we bind the dataset mt from before, to columns of a grid, and as a little bit of extra spice, we also bind a `TEdit` to the grid's `RowCount` property, so we dynamically, at runtime, can change the number of shown rows.

```
Binding.Clear;
bnd:=Binding.Bind(mt,'Category',StringGrid1,'#1');
Binding.Bind(mt,'Species Name',StringGrid1,'#2');
Binding.Bind(mt,'@',StringGrid1,'@',[mwboTwoWay]);

// Show position number
Binding.Bind(mt,'@',StringGrid1,'#0');

// Bind to rowcount for easy on the fly change at runtime
Binding.Bind(leRowCount,'Text',StringGrid1,'RowCount');

if bnd.Navigator<>nil then
  bnd.Navigator.First;
```

It is still the same way we bind as we are getting used to, but now a **`#n syntax`** is revealed.
It simply refers to the column number (starting with 0) in the grid, that is bound to.

Using the navigator, the grid will now act almost exactly as if it was a `TDBGrid`. You can scroll through the source dataset, and the grid will automatically update.
Changing the selected row within the grid will also automatically update the position of the source dataset, because we twoway bound the @ of the source dataset to the @ of the grid.

Changes to the current record will in the source dataset automatically be reflected in the grid and had we bound two ways, the data entered into the current row in the grid would be reflected back to the source.

Example – functions as binding source and/or destination
Not only objects, data or datasets can act as source or destination for a binding. `Anonymous functions` can also.

```
// Show calling function when Edit1.Text is changed.
Binding.Bind(Edit1,'Text',function(const AProxy:TkbmMWBindingCustomProxy;
var AValue:TValue):boolean
  begin
    Log.Debug('Got change from binding to Edit1: '+AValue.ToString);
    Result:=true;
  end);
```

The above example essentially works as a kind of event handler where the code in the `anonymous function` will be called every time a change has been detected in `Edit1.Text`. It will in this case log the situation using the **kbmMW logging** framework **`TkbmMWLog.`**

Next example is how to use a function as the source for data.

```
Binding.Bind(function(const AProxy:TkbmMWBindingCustomProxy; var AValue:TValue):boolean
begin
  AValue:=Random(100);
   Result:=true;
end,Edit1,'Text');
```

Now the `anonymous function` is repeatedly called. Every time the function returns a new changed value, `Edit1.Text` will be updated. How often the function is polled depends on the setting of the property `UpdateFrequency` of the **Binding** instance which we use to define the bindings with. Default the frequency is 10 times per second, but you can always change the frequency to match your liking.

```
Binding.UpdateFrequency:=1000;
```

The above line will change the update frequency to once per second.

Example – Augmenting bindings
What if you want to define a binding that takes a numerical value from one source, and outputs that to a label but formatted differently?

You will use the **`ToDestinationExpression`** method available on the resulting interface returned from the Binding.Bind function.

```
// Show calling function to populate Edit1.Text and format its look.
Binding.Bind(function(const AProxy:TkbmMWBindingCustomProxy; var AValue:TValue):boolean
  begin
     AValue:=Random(100);
     Result:=true;
  end,Edit1,'Text')
   .ToDestinationExpression('"Hello "+data');
```

In this case we simply add on to the previous function binding example, and asks **kbmMW SmartBinding** to augment the data, on the way to the destination according to the **string expression** given in the `ToDestinationExpression` function. This example results in `Edit1.Text` containing the value **'Hello '** and a random number.
The **string expression** is quite feature rich, as it is based on the same expression handling capabilities that **kbmMW** is taking advantage of elsewhere, that originates from **kbmMemTables** capable SQL parser and evaluator. In this case, we however only support the math like expression part, not the SQL itself. **But you can use all regular operations you would expect to be able to use, including many nice conversion, regular expression, math, conditional evaluation and more functions.** You can read more about supported functions and how to extend the expression parser and evaluator with your own user defined functions, here:
https://components4developers.blog/2017/08/20/user-defined-functions-and-kbmmemsql/

As a binding can be two way, there is a need to also be able to format or perhaps unformat the value when it is going back to the source. Hence a **ToSourceExpression** function also exists.

```
Binding.Bind(Edit1,'Text',Edit2,'Text',[mwboTwoWay])
  .ToDestinationExpression('"Hello "+data')
  .ToSourceExpression('Mid(data,7)');
```

This example grabs what is in `Edit1.Text` and puts it into `Edit2.Text` with the text **'Hello '** prefixed. However it also recognise changes made in `Edit2.Text` and moves that text to `Edit1.Text` after first having removed the first 6 characters of it. This type of two way binding would often make event driven binding go nuts, because of the potential endless event train happening in the `TEdit` control by the changes. However **kbmMW SmartBinding** is not affected by those events, and ensures the updates with minimum effort.

ENABLING, DISABLING, UNBINDING AND REBINDING

Sometimes you may want to prevent a binding to do its job. If the prevention is only supposed to be temporary, then one way is to disable it.

```
var
  bnd:IkbmMWBinding;
begin
  bnd:=Binding.Bind(....);
...
  bnd.Disable:=true;
...
  bnd.Disable:=false;
end;
```

If you want permanently to disable it, you might as well remove it. For that purpose the Unbind methods exists.

```
Binding.UnbindSource(Edit1);
```

The above will unbind `Edit1` from being a source for any bindings.

```
Binding.UnbindDestination(Edit2);
```

And the above will unbind `Edit2` from being a destination for any bindings.

You can also unbind using the `IkbmMWBinding` you received when issuing the Bind method.

```
var
 MyBinding:IkbmMWBinding;
begin
 MyBinding:=Binding.Bind(....);
...
 Binding.Unbind(MyBinding);
```

If you are not binding to anonymous functions, you can also unbind using the exact same arguments as the bind

```
Binding.Bind(Edit1,'Text',Edit2,'Text');
...
Binding.Unbind(Edit1,'Text',Edit2,'Text');
```

Finally you may want to **rebind. Rebind** essentially makes it possible to modify a binding from using one source or destination instance to another. It is specially interesting when binding to transient records or objects

```
Binding.Rebind(@data,@data2);
```

The above changes any bindings which references the record or memory buffer 'data', and update those bindings to instead reference the record or memory buffer 'data2'.

Similarly you can rebind a control

```
Binding.Rebind(Edit1,NewEdit1);
```

All bindings referencing `Edit1` will now instead reference `NewEdit1`.

PROLOGUE

As you may now have noticed, the syntax for runtime binding is consistent and simple, and makes it easy to refactor bindings when user interfaces or controls are refactored.

I did earlier on mention that in addition to the existing thread safe **`Binding singleton`**, you can choose to make your own binding manager instances. The reason to do that can include that you want different bindings to be updated at different intervals of some reason, or that you want very easy access to drop or recreate all bindings for for example a frame in one simple go without affecting all other bindings defined in other frames, and without having to explicitly unbind each of them.

```
var
 myBindingMgr:TkbmMWBindings;
begin
 myBindingMgr:=TkbmMWBindings.Create(1000);
...
 myBindingMgr.Free;
```

The above example creates another binding manager which only polls every second.
Remember to free your own created binding managers when you do not need them any longer.

There are many more ideas in my head about making binding easier and add more features to it, but this is what will be included as beta code in next full release of **kbmMW Enterprise Edition.**

If you like our products and posts, please share the posts with everyone you know could benefit from them!

C4D is in it for making coding applications easier, letting you focus on the business features rather than the plumbing. Why? Because I hate doing plumbing when I'm developing end user code. So I'm actually not developing all this stuff for you, but selfishly and egoistically for my self, hoping you will like it too.

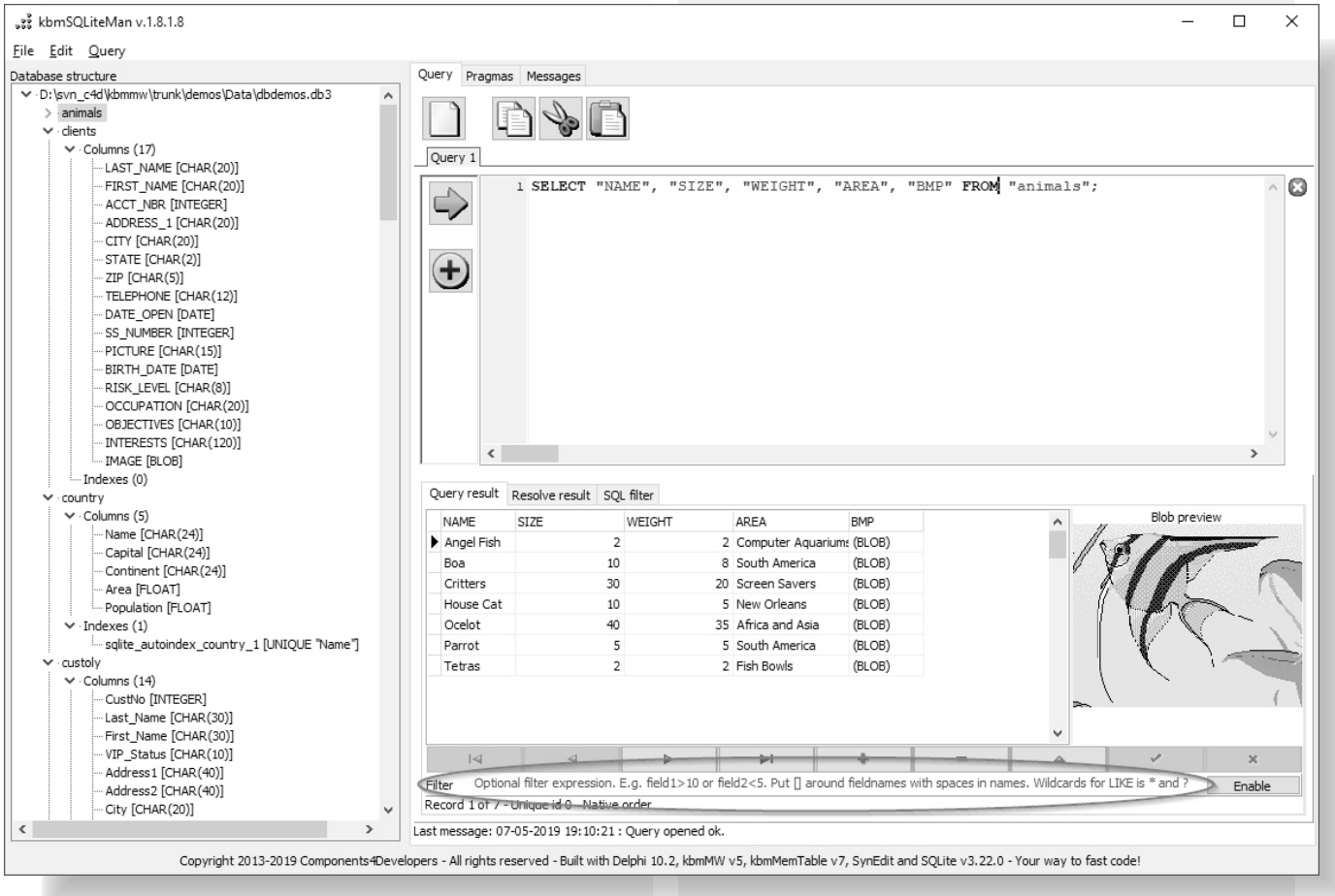
MANAGING SQLITE DATABASES FREE TOOL

OUR FREE COOL TOOL FOR MANAGING SQLITE DATABASES HAS BEEN RELEASED IN V. 1.80.
This includes a new local SQL filter capability in addition to being compiled with latest version of kbmMW and kbmMemTable.

To download it, login at
`https://portal.components4developers.com`
and it is readily available for you to download.

If you do not have a login already, you can easily and for free sign up same place.

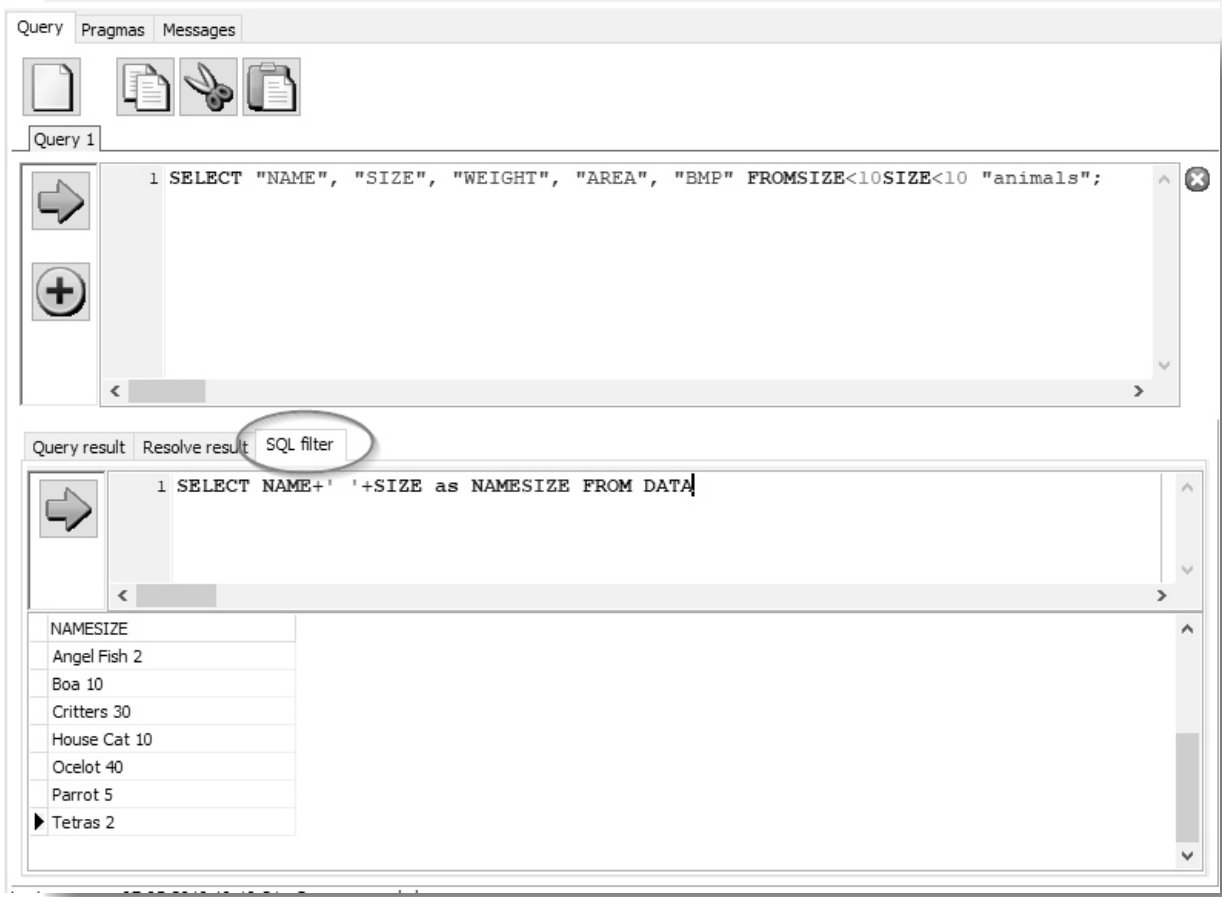
The existing filtering capability **kbmSQLiteMan** has for a long time had the ability to locally add an additional filter on result sets returned from the SQLite database.



In this filter are, a fairly complex expression can be type, ranging from simple **SIZE<10** which filters the result set to only show the 3 records with a SIZE value less than 10. **AND, OR, NOT, (), LIKE "P*"** etc. are all supported.

The new local SQL "filtering" capability

The above filtering is fine for simply ensuring some records in the result set are not displayed. However if you want to do something more complex with the result set, like complex filtering, grouping, calculations or custom ordering etc, you may want to use the new SQL filtering capability. It internally use kbmMemTable's SQL ability which is fairly close to regular SQL.



Here we combine the result set's **NAME** and **SIZE** column to one column called **NAMESIZE.** This simple example could of course just as well have been handled in the original **SQLite SQL** statement, but there may be situations where you want to keep the original data, but do something more with it.

kbmMemTable SQL supports **SELECT** statements with optional sub select, **CASE/WHEN/ELSE, IN, BETWEEN, LIKE, MOD, DIV, ORDER BY, GROUP BY, HAVING, LIMIT, OFFSET, || (concat), +, -, *, /** in addition to a good number of functions:

```
SIN(x), COS(x), TAN(x), LOG(x),
LOG2(x), EXP(x), TRUNC(x), FRAC(x),
MOD(x), DIV(x), SQRT(x), SQR(x),
ROOT(x,y), MIN(x,…), MAX(x,…),
AVG(x,…), SUM(x,…), ABS(x),
POW(x,y)

UPPER(x), LOWER(x), TRIM(x),
TRIMLEFT(x), TRIMRIGHT(x),
MID(x,p,n), LEFT(x,n), RIGHT(x,n),
LENGTH(x), LEFTPAD(x,c,n),
RIGHTPAD(x,c,n), CHR(x), POS(sx,x),
REPLACE(x,so,sr [,o1][,o2]),
SPLIT(x,sx, OUT v), REGEXP(re,x
[,OUT v]),

NOW, DATE(x), TIME(x), YEAR(x),
MONTH(x), DAY(x), HOURS(x),
MINUTES(x), SECONDS(x),
DATESTRING(x), TIMESTRING(x)
CASTTODATETIME(x), CASTTOSTRING(x),
CASTTONUMBER(x), CASTTOFLOAT(x),
CASTTOINT(x)
IF(x,te,fe [,ne]), NULLIF(x1,x2),
ISNULL(x), COALESCE(x,…)
```

x and y = A value (constant, out defined variable, expression or a field). Values are auto converted between numbers and strings as needed
p = An offset. 1 is first character
n = A number indicating a count
c = A character
sx = A sub string value
so = String to search for
sr = String to replace with
o1 = True/false value. If true, replaces all occurrences
o2 = True/false value. If true, ignores case
v = Name of a variable. The name must start with $. $VAR1 is a valid name
re = Regular expression
te = True expression. Returned if x evaluates to true
fe = False expression. Returned if x evaluates to false
ne = NULL expression. Returned if x evaluates to NULL
x1 = A value. If x1=x2, NULL is returned, else x1.
x2 = A value
fn = A fieldname as a string, to return the value from

The FROM clause must always refer to the virtual table named DATA which represents the complete original result set.

Examples of valid SQL filter statements

```
SELECT fld5,MAX(fld1),MAX(fld2) FROM DATA GROUP BY fld5
SELECT DISTINCT * FROM DATA
SELECT FieldValue('fld1') as myfld1,Coalesce(FieldValue('NoField'),"ISNULL")
as NoField  FROM DATA
SELECT Coalesce(10) FROM DATA
SELECT Coalesce(Null) FROM DATA
SELECT Coalesce(fld7,99999) FROM DATA
SELECT Coalesce(fld7,Null,99999) FROM DATA
SELECT fld1, (SELECT Max(fld1) FROM DATA) FROM DATA
SELECT fld1 FROM DATA WHERE fld3 IN (SELECT b.fld2 FROM DATA b WHERE b.fld2<4)
SELECT 1, MAX(fld1) FROM DATA GROUP BY 1
SELECT 1 AS 'ID', MAX(fld1) FROM DATA GROUP BY ID
SELECT NULL as fld1, fld2, "ABC" as fld3, 10+20 as fld4 FROM DATA
SELECT RegExp("1\d",fld2), fld2 FROM DATA
SELECT RegExp("^(\d)(\d*)",fld2,OUT $PAR),$PAR,fld2 FROM DATA
SELECT fld1 AS TIME, fld2 FROM DATA WHERE fld1 LIKE „STR1%"
SELECT fld1, fld2 FROM DATA WHERE fld1 LIKE „STR1%"
SELECT fld1, fld2 FROM DATA WHERE NOT fld1 LIKE „STR1%"
SELECT fld1, fld2 FROM DATA WHERE fld1 NOT LIKE „STR1%"
SELECT fld1, fld2 FROM DATA WHERE fld2 LIKE „8%"
SELECT fld1, fld2, CASE WHEN fld2<100 THEN 'LOW' WHEN fld2>=100 AND fld2<200
THEN 'MEDIUM' ELSE 'HIGH' END FROM DATA
SELECT fld1, fld2, CASE fld2 WHEN 10 THEN 99999 WHEN 20 THEN 22222 ELSE -1 END
FROM DATA
SELECT RecNo,RowID,* FROM DATA
SELECT fld1 FROM DATA WHERE fld2 in (10,20,30)
SELECT 1-2-3 FROM DATA LIMIT 1
SELECT LeftPad(fld3,'A',10),RightPad(fld3,'B',12),fld3||'ABC' FROM DATA
SELECT fld2+1 as fld2a FROM DATA ORDER BY fld2a DESC
SELECT fld2+1 as fld2 FROM DATA
SELECT fld1,fld2,fld3,fld6,fld3 AS SomeField1,fld2 AS SomeField2,fld5 FROM
DATA WHERE fld5 IN (5) ORDER BY fld6,SomeField2
SELECT fld2 as Field2, fld3, sum(fld5) as fld5, Sum(fld2) as SomeField1,
Sum(fld3) as SomeField2 FROM DATA GROUP BY Field2, fld3
SELECT fld2 as Field2, fld3, sum(fld5) as SomeField1, Sum(fld2) as
SomeField2, Sum(fld3) as SomeField3 FROM DATA GROUP BY Field2, fld3
SELECT fld5,sum(fld5) as sumoffld5,count(fld5) as countoffld5 FROM DATA GROUP
BY fld5 HAVING count(fld5)>2
SELECT fld2 as somefield, fld3 FROM DATA
SELECT fld5 as somefield,sum(fld5),count(fld5) FROM DATA GROUP BY somefield
HAVING count(fld5)>2
SELECT count(*)+5 FROM DATA
SELECT * FROM DATA LIMIT 10 OFFSET 50
SELECT * FROM DATA LIMIT 10
SELECT * FROM DATA OFFSET 50
SELECT fld2, IF(fld2>10,True,False) AS IsSomething FROM DATA
SELECT SUM(fld5),SUM(fld6),SUM(fld5)+Sum(fld6) AS TotalField FROM DATA
SELECT count(distinct Left(fld1,4)) from DATA
SELECT length(fld1) from DATA
SELECT fld5,sum(if(fld5>5,1,0)),count(fld5) from DATA group by fld5
SELECT
fld1,Min(20,30,10,40),Max(20,30,10,40),Avg(20,30,10,40),Sum(20,30,10,40)
FROM DATA
SELECT fld2 FROM DATA where fld2 xor (fld2 mod 10)
SELECT if(1 xor 1,0,1),fld2 FROM DATA
SELECT if(0 xor 1,0,1),fld2 FROM DATA
SELECT if(10 IN (10,20,30),1,0),fld2 FROM DATA
SELECT if(11 IN (10,20,30),1,0),fld2 FROM DATA
SELECT MAX((SELECT max(b.fld1) FROM DATA b WHERE a.fld2=b.fld2)) FROM DATA a
```

ANN: kbmMemTable v. 7.82.00 Standard and Professional Edition released

• Added support for SQL syntax **`ALTER TABLE <tbl> PRIMARY [KEY] (fld1,…)`**

UPDATED TO SUPPORT LAZARUS 2.0.2

• Changed to allow only 2 arguments to MID(..) SQL function. If length not given, copies rest.
• Added sfLoadAsUTF16 to CSV stream format.
• Will assume fields are widestring rather than string.
• Improved sub select to operate on derived data based on original data
to ensure seemingly transactional separation
when sub select is running on same table as outer operation.
• Fixed subselect bugs.
• Fixed hints.
• Fixed bug creating index with more than one field using SQL.
• Fixed stack overflow bug when function expression argument cause exception during parse.

kbmMemTable is the premier high performance,
high functionality in memory dataset for Delphi and C++Builder
with kbmMemTable Professional topping the scales as being the worlds fastest!

If you have an up to date Service and Update (SAU) subscription,
then you can immediately visit **`https://portal.components4developers.com`**
to download the latest **kbmMemTable** release.

If not, please visit our shop at **`http://www.components4developers.com`**
and extend your SAU with another 12 months.

COMPONENTS 4 DEVELOPERS

ANN: kbmMW Professional and Enterprise Edition v. 5.09.00 released!

We are happy to announce v5.09.00
of our popular middleware for Delphi and C++Builder.

Notice that kbmMemTable v. 7.82.00 or newer is a prerequisite to this update.

This is a major release containing major new features,
updates to existing features and bugfixes.

The release includes:

NEW! SmartBinding support for kbmMW Enterprise Edition (BETA!)
NEW! XML-RPC support for kbmMW Enterprise Edition
NEW! JSON-RPC support for kbmMW Enterprise Edition
NEW! Automatically analyse JSON, YAML or XML data and have kbmMW generate Delphi marshalling classes
Significant improvements in
SmartServices / RTTI / Scheduler / LINQ / Object Notation
XML and JSON marshalling ORM
Important fixes.
Please check the end of this post for a detailed change list.

CodeGear Edition may be available for free,
but only supports a specific Delphi/Win32 SKU, contains a limited feature set and do not include source.

https://portal.components4developers.com

kbmMW is the premiere n-tier product for Delphi, C++Builder and FPC on .Net, Win32, Win64, Linux, Java, PHP, Android, IOS, embedded devices, websites, mainframes and more.

Components4Developers is a company established in 1999 with the purpose of providing high quality development tools for developers and enterprises. The primary focus is on SOA, EAI and systems integration via our flagship product kbmMW.

COMPONENTS 4 DEVELOPERS

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.09.00 RELEASED!

- RAD Studio XE2 to 10.3 Rio supported
- Win32, Win64, Linux64, Android, IOS 32, IOS 64 and OSX client and server support
- Native high performance 100% developer defined application server
- Full support for centralized and distributed load balancing and failover
- Advanced ORM/OPF support including support of existing databases
- Advanced logging support
- Advanced configuration framework
- Advanced scheduling support for easy access to multithread programming
- Advanced smart service and clients for very easy publication of functionality
- High quality random functions.
- High quality pronouncable password generators.
- High performance LZ4 and Jpeg compression
- Complete object notation framework including full support for YAML, BSON, Messagepack, JSON and XML
- Advanced object and value marshalling to and from YAML, BSON, Messagepack, JSON and XML
- High performance native TCP transport support
- High performance HTTPSys transport for Windows.
- CORS support in REST/HTML services.
- Native PHP, Java, OCX, ANSI C, C#, Apache Flex client support!

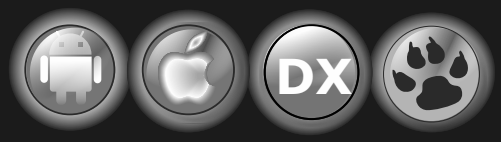
NEW! SmartBinding support for kbmMW Enterprise Edition (BETA!)
NEW! XML-RPC support for kbmMW Enterprise Edition
NEW! JSON-RPC support for kbmMW Enterprise Edition
NEW! Automatically analyse JSON, YAML or XML data and have kbmMW generate Delphi marshalling classes

- High speed, unified database access (35+ supported database APIs) with connection pooling, metadata and data caching on all tiers
- Multi head access to the application server, via REST/AJAX, native binary, Publish/Subscribe, SOAP, XML, RTMP from web browsers, embedded devices, linked application servers, PCs, mobile devices, Java systems and many more clients
- Complete support for hosting FastCGI based applications (PHP/Ruby/Perl/Python typically)
- Native complete AMQP 0.91 support (Advanced Message Queuing Protocol)
- Complete end 2 end secure brandable Remote Desktop with near realtime HD video, 8 monitor support, texture detection, compression and clipboard sharing.
- Bundling kbmMemTable Professional which is the fastest and most feature rich in memory table for Embarcadero products.

kbmMemTable is the fastest and most feature rich in memory table for Embarcadero products.

- **Easily supports large datasets with millions of records**
- **Easy data streaming support**
- **Optional to use native SQL engine**
- **Supports nested transactions and undo**
- **Native and fast build in M/D, aggregation/grouping, range selection features**
- **Advanced indexing features for extreme performance**

COMPONENTS 4 DEVELOPERS



EESB, SOA,MoM, EAI TOOLS FOR INTELLIGENT SOLUTIONS. kbmMW IS THE PREMIERE N-TIER PRODUCT FOR DELPHI / C++BUILDER