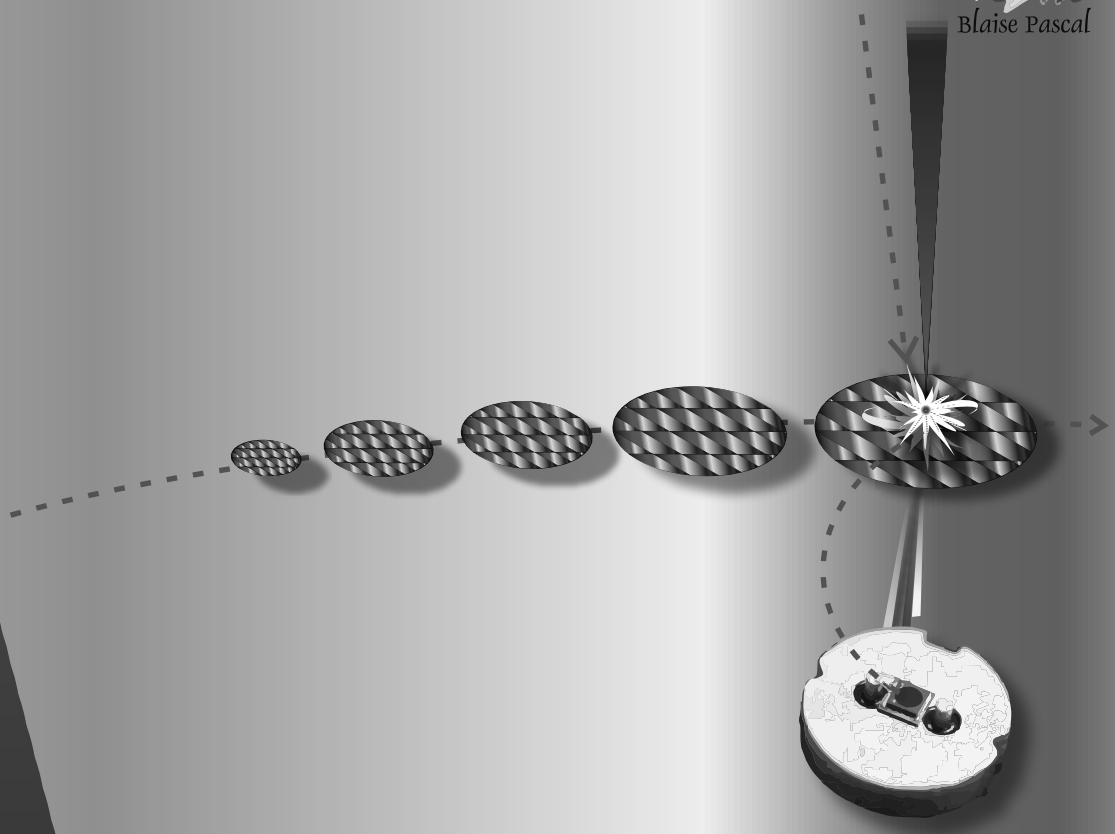
BLAISE PASCAL MAGAZINE 106

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



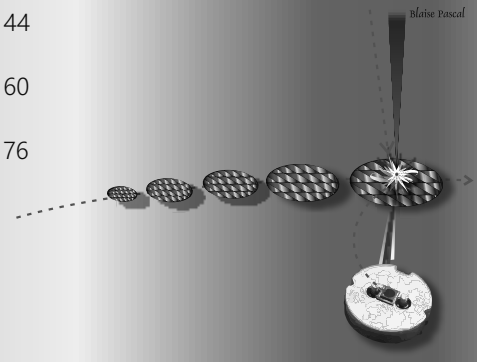
AGSI (Aggregated Gas Storage) - Data Storage
Undo System explained
Laser CPU - the future is getting close
Untappable Internet- no more cracking
The new Delphi 11.2.1 has arrived
Fastreport QR Code makes it very eassy
Translating your Website with PAS2JS
FPC UP DeLuxe the fantastic Lazarus Installer
kbmMW The new version number 5.20.01 has arrived

THE LAZARUS TEAM ADVISES

THE BEST & EASIEST WAY TO INSTALL FPC LAZARUS STABLE / TRUNK OR CROSSCOMPILE

CONTENT

ARTICLES



ADVERTISERS


Niklaus Wirth

Pascal is an imperative and procedural programming language, which Niklaus Wirth designed (left below) 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).

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

Stephen Ball
http://delphiaball.co.uk
@DelphiABall

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
E-mail: David @ davdata.nl

Benno Evers
b.evers @ everscustomtechnology.nl

Bruno Fierens
www.tmssoftware.com
bruno.fierens @ tmssoftware.com

Holger Flick
holger @ flixments.com

Mattias Gärtner
nc-gaertnma@netcologne.de

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.com

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

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

Danny Wind
dwind @ delphicompany.nl

Jos Wegman / Corrector / Analyst

Siegfried Zuhr
siegfried @ zuhr.nl

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

Subscriptions (2022 prices)

	Internat. excl. VAT	Internat. incl. 9% VAT	+Shipment	TOTAL
Printed Issue ±60 pages	€ 200	€ 218	€ 130,00	€ 348
Electronic Download Issue 60 pages	€ 64,20	€ 70		€ 70

Member and donator of
Member of the Royal Dutch Library
KONINKLIJKE BIBLIOTHEEK

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 (Stichting Programeertaal Pascal)
IBAN: NL82 ABNA 0441960863 BIC ABNANL2A VAT no.: 81 42 54 147 (Stichting Programmeertaal Pascal)
Subscription department Edelstenenbaan 21 / 3402 XA IJsselstein, Netherland Mobile: + 31 (0) 6 21.23.62.68 office@blaisepascal.eu

From your editor

To our great pleasure, we have also produced a German-language version of this issue. This is our first time doing this and we still have a lot to learn. Therefore, we also ask for support through language and writing corrections.
We would like to invite you to help us. By adding more languages, we are making the Pascal language more and more familiar and easier to understand.

Just in this issue, we have added an article on installing Lazarus in the easiest way possible: **FPC de Luxe.**
It has never been easier (*thanks to* **LongDirtyAnimAlf** *alias Alfred Glänzer*)
to install it and this programme is already able to offer a large number of languages for operation. You can install it very easily with almost a mouse click, but it is also excellent at serving separate requirements.
As of now, creating a trunk version is a piece of cake…

Since we are talking about Lazarus now, I can announce in advance that in the December issue we will take a look into the future, and you will be surprised what will be possible then…

Because **PAS2JS** is becoming a very important internet tool - so that you can very easily create and also manage your own website with **Pascal** - we have planned to set up a course for this, more on that soon.
This will then be supported by a book that will highlight all aspects of **PAS2Js** so that it fits well with the course.
Which, by the way, will be given for a maximum of 12 registrations at a time.
If more course participants are signed up, we will reserve several dates.

Speaking of booking, the last meeting here in the Netherlands was very successful and we will stay at this location.
It will take place again in March 2023, now that we hope to have no more problems with Diseases.

Delphi has got a new way of adding new components locally and making them available to third parties. **Danny Wind** writes a something about that.

I have described a few more things that concern the future: using Laser techniques not just for making Chips but replacing them completely with chips that have Laser as the basic element and thus work completely with light-controlled CPUs. That alone would mean we could reduce the use of energy worldwide by more than 10%.
That's another piece of good news.

Of course, you would expect it in this day and age that we should focus on climate-friendly or improving processes. Give us examples of how you can use programming to reduce or eliminate climate impact.
Send your idea to:

editor@blaisepascalmagazine.eu

Detlef

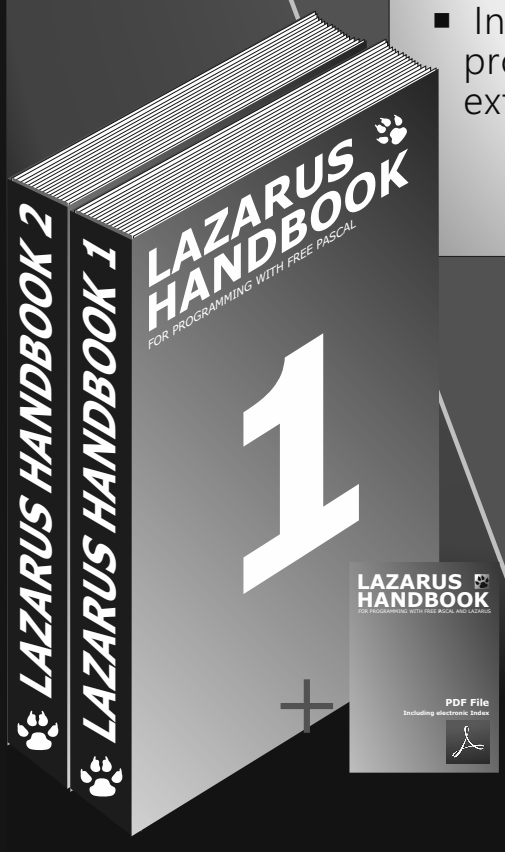
From our Technical advisor: Cartoons from Jerry King

"My phone got wet, so I'm pouring a dry martini over it to dry it out."

Sale
Lazarus Handbook Pocket

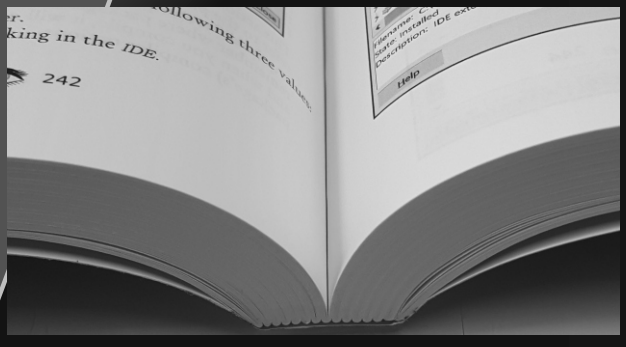
Price: € 26,50 Excluding VAT and Shipping

- English
- Printed black & white
- 2 Volumes
- PDF included
- 934 Pages
- Weight: 2kg
- Shipment included
- Extra protected
- Including 40 Example projects and extra programs



- **BlaisePascalMagazine PDF viewer included**

printed on FSC paper
https://fsc.org/en/forest-management-certification



https://www.blaisepascalmagazine.eu/product-category/books/

maXbox Starter 99 – Data representation of gas in storage as a timeline AGSI dataset

runs under Python3, Delphi, Jupyter-Notebook, Lazarus and maXbox4.

INTRODUCTION

This data science tutorial explains the so called **AGSI** data storage and his visualisation of the time line. **AGSI** is the **Aggregated Gas Storage Inventory** and offers you the possibility to be kept up to date whenever a new service announcement or update from one of our data providers is posted on the website. The **Gas Infrastructure Europe** (**GIE**) is also providing related data such as the **Storage Map** and **Storage Investment Database** at **https://www.gie.eu/publications/maps/**

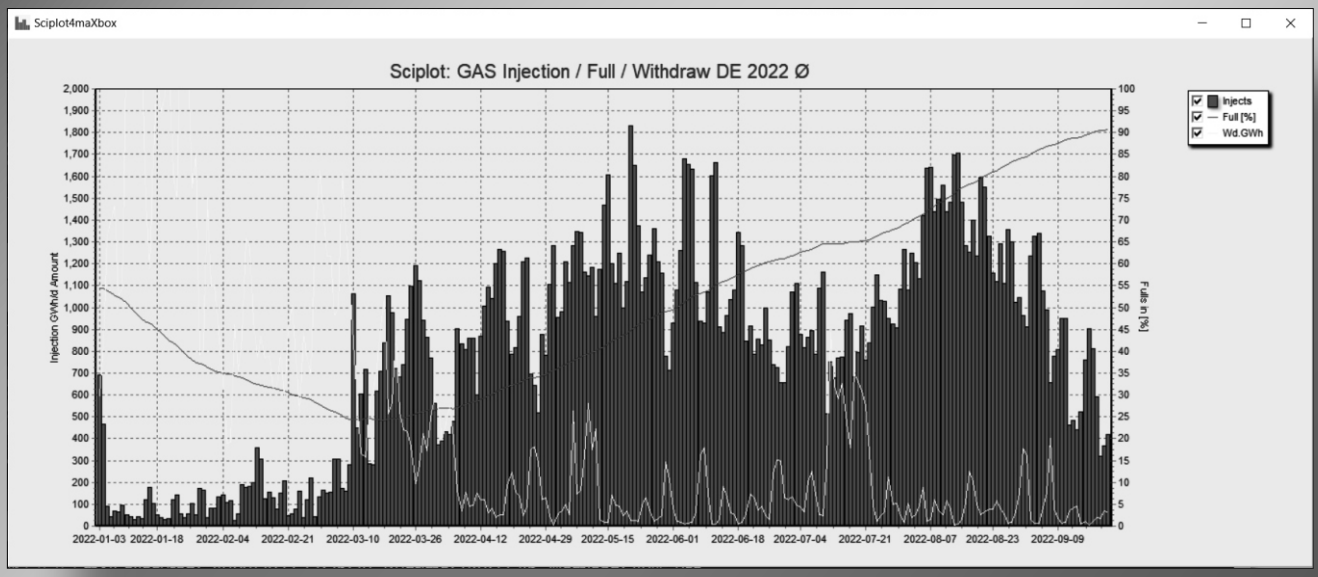


Figure 1: The result of the data is the chart.

We use **WinHttp.WinHttpRequest, JSONObjects** and **TEECharts** library with loading and testing the plot. Also an **API**-key is needed, get the key first at: **https://agsi.gie.eu/account**

The data represents gas in storage at the end of the previous gas day. Data is updated every day at 19:30 **CET** and a second time at 23:00. Before we dive into code this is the main part of the script:

```
plotform:= getForm2(1400, 600, clsilver, 'Sciplot4maXbox');
plotform.icon.loadfromresourcename(hinstance,'ZHISTOGRAM');

HttpResponse:=
    getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
JSON2Plot(plotform, letGenerateJSON2(HttpResponse));
```

The main part generates a form, invokes the **API** and plots the data.
GIE is providing an **API** (Application Programming Interface) service on its **AGSI** and **ALSI** transparency publication platforms.
Using **API** access, consumer can bypass the **AGSI** and **ALSI** website and get hold of the data directly and continuously. It enables to extract, filter, aggregate the data and create any subset as required, without having to download each dataset separately from the website. The **API** export format is **JSON.** For example a subset of 150 days:

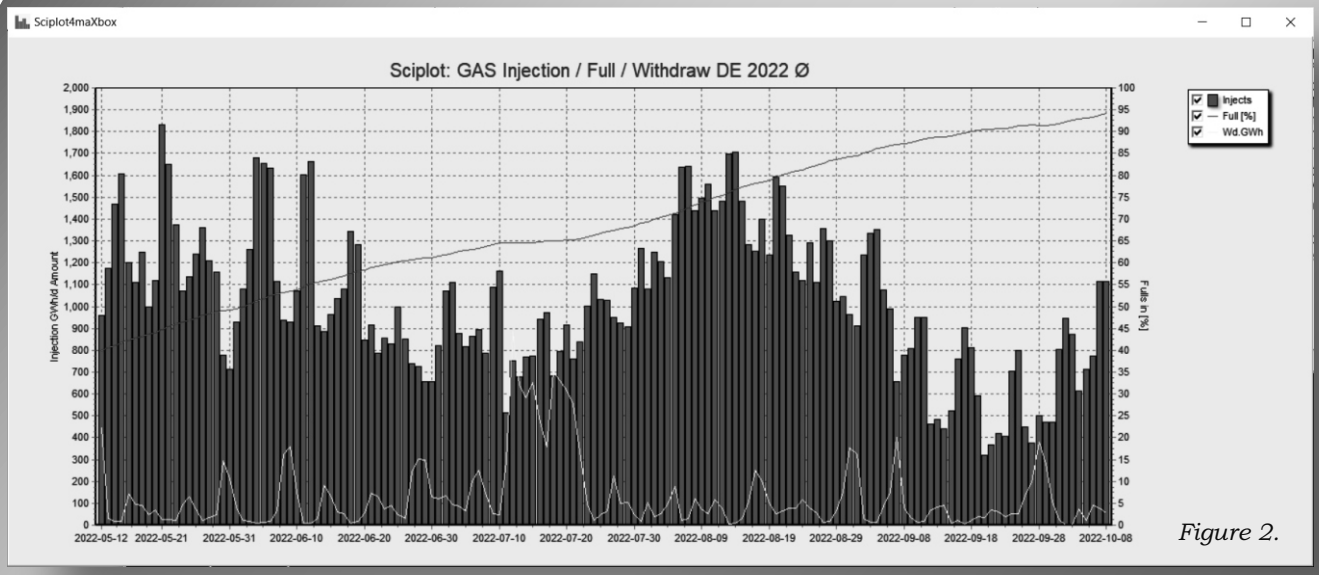


Figure 2.

The published datasets are based on the **EIC** code mapping table provided to **ACER**. Storage data is aggregated by company, and country.
With the call I pass country, start-date and amount of days:

```
_getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

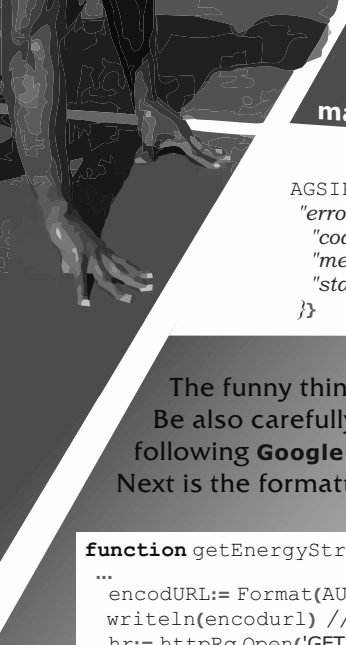
All available datasets can be downloaded also in **Excel, CSV** and **JSON** format. The data in this report shows an aggregated view – individual datasets by company and storage facility are also accessible.

So lets start with the **API** call:

```
_HttpResponse := getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

This command and script runs **`WinHttp.WinHttpRequest`**. When you fail you get a bunch of exceptions like the following:

Exception: **`WinHttp.WinHttpRequest`**: The data necessary to complete this operation is not yet available; or you missed a valid key:



```
AGSIPost: Failed at getting response: 403{
 "error": {
   "code": 403,
   "message": "The request is missing a valid API key.",
   "status": "PERMISSION_DENIED"
 }}
```

The funny thing is the **JSON** formatted exception.
Be also carefully to expose your key as I get from **Git**: **GitGuardian** has detected the following **Google API Key** exposed within your **GitHub** account.
Next is the formatting of the get-call with a valid **API**-key request in the function **energyStream()**

```
function getEnergyStreamJSON2(AURL, feedstream, aApikey:string): string;
 ...
  encodURL:= Format(AURL,[HTTPEncode(asp[0]),(asp[1]),asp[2]]);
  writeln(encodurl) //debug
  hr:= httpRq.Open('GET', encodURL, false);
  httpRq.setRequestheader('user-agent',USERAGENTE);
  httpRq.setRequestheader('x-key',aAPIkey);
 ...
```

And where is the fabulous content-type? As far as I understood there are only two places in a web-request where to set a content-type:

❶ The client sets a content type for the body he is sending to the server (e.g. for get and post).
❷ The server sets a content type for the response.

A sender that generates a message containing a payload body has to generate a Content-Type header field in that message unless the intended media type of the enclosed representation is unknown to the sender; otherwise we failed at getting response: 503503 - Service Unavailable.

```
('headers={"Content-Type":"application/json"}')
 httpRq.setRequestheader('Content-Type',application/json);
```

It means that the content-type **HTTP** header should be set only for **PUT** and **POST** requests.
GET requests can have "Accept" headers, which say which types of content the client understands.
The server can then use that to decide which content type to send back.
As an option you can also use **TALWinInetHttpClient**.
It is a easy to use **WinInet-based** protocol and supports **HTTPs.**
HTTP client component which allows to post and get any data from the Web via **HTTP** protocol.

```
function TALHTTPClient_Get(aUrl: AnsiString; feedstream, aApikey: string): string;
Var  LHttpClient: TALWininetHttpClient;  asp: TStringArray;
begin
  LHttpClient:= TALWininetHttpClient.create;
  asp:= splitStr(feedstream,',');
  LHttpClient.url:= Format(AURL,[HTTPEncode(asp[0]),(asp[1]),asp[2]]);
  LHttpClient.RequestMethod:= HTTPmt_Get; //HTTPrm_Post;
  LHttpClient.RequestHeader.UserAgent:=USERAGENTE;
  //LHttpClient.RequestHeader.CustomHeaders:=
  LHttpClient.RequestHeader.RawHeaderText:='x-key:'+aAPIkey;
  try
    result:= LHttpClient.Get1(LHttpClient.url); //overload;
  finally
    LHttpClient.Free;
  end;
end;
```

Any missing or incomplete data is also be visible on **AGSI.**
Next we step to the conversion of our **JSON** response for the plot with `TJSONObject`:

```
function letGenerateJSON2(HttpRqresponseText: string): TJSONArray;
var jo: TJSONObject;
begin
   jo:= TJSONObject.Create4(HttpRqresponseText);
   try
      //writeln(jo.getstring('data'));
      writeln(itoa(jo.getjsonarray('data').getjsonobject(0).length))
      writeln(itoa(jo.getjsonarray('data').length))
      result:= jo.getjsonarray('data');
      //write out to check
   for it:= 0 to result.length-1 do
      writeln(result.getjsonobject(it).getstring('gasDayStart')+':'+
            result.getjsonobject(it).getstring('injection'));
   except
      writeln('EJson: '+ExceptiontoString(exceptiontype, exceptionparam));
   end;
end;
```

And this JSON Array as above function returns, we pass to the next plot procedure:

```
procedure JSON2Plot(form1: TForm; jar: TJSONArray);
var chart1: TChart; cnt: integer; sumup,tmp2,tmp: double; gday: string;
begin
  form1.onclose:= @Form_CloseClick;
  chart1:= ChartInjector(form1);
  sumup:=0; tmp2:=0; tmp:=0;
  try
    for cnt:= 0 to jar.length-1 do
      begin
        //writeln(locate.getjsonobject(it).getstring('gasDayStart')+':'+
        tmp:= jar.getjsonobject(jar.length-1-cnt).getdouble('injection');
        tmp2:= jar.getjsonobject(jar.length-1-cnt).getdouble('full');
        sumup:= sumup+tmp;
        gday:= jar.getjsonobject(jar.length-1-cnt).getstring('gasDayStart');
        chart1.Series[0].Addxy(cnt,tmp,gday,clgreen);
        chart1.Series[1].Addxy(cnt,tmp2,'',clred);
        chart1.Series[2].Addxy(cnt,jar.getjsonobject(jar.length-1-cnt).getdouble('withdrawal'),'',clyellow);
      end;
  except
    writeln('EPlot: '+ExceptiontoString(exceptiontype, exceptionparam));
  end;
  PrintF('Landrange %d: Injection sum: %.2f',[jar.length-1,sumup]);
end;
```

As we can see we have 4 series to plot (including timeline):

❶	**Injection**	(Injection during gas day)
❷	**Full**	(Storage / WGV (in%))
❸	**Withdrawal**	(Withdrawal during gas day (2 digits accuracy)).
❹	**GasDayStart**	(The start of the gas day reported)

The time series is a result of the gas day and a trend is available.
"Gas day" means the period from 5:00 to 5:00 **UTC** the following day for winter time and from 4:00 to 4:00 **UTC** the following day when daylight saving is applied. Gas day is to be interpreted as `UTC+1` for **CET** or `UTC+2` in summer time for **CEST.** (Definition: see **CAM Network Code** specific-cations).
API access is provided in a **REST**-like interface (**Representational State Transfer**) exposing database resources in a **JSON** format with content-type in the mentioned `Response Header`.

Response Headers

X-Firefox-Http3	h3
alt-svc	h3=":443"; ma=86400, h3-29=":443"; ma=86400
cache-control	no-cache, private
cf-cache-status	DYNAMIC
cf-ray	7580b85f6dae0200-ZRH
content-encoding	br
content-type	application/json
date	Mon, 10 Oct 2022 16:26:53 GMT
server	cloudflare
x-robots-tag	noindex

Request Headers

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding	gzip, deflate, br
Accept-Language	en-GB,en;q=0.5
Alt-Used	agsi.gie.eu
Connection	keep-alive
Host	agsi.gie.eu
Sec-Fetch-Dest	document
Sec-Fetch-Mode	navigate
Sec-Fetch-Site	none
Sec-Fetch-User	?1
Sec-GPC	1
TE	trailers
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0

Figure 3.

The code of the data science vision contains example usage, and runs under **Python3, Delphi, Jupyter-Notebook, Lazarus** and **maXbox4**. **NOTE** that the **API** service is made available to the public free of charge. Only the data as currently available on the platforms is made available.
Tip: To extract data direct from the system, you can click on one of these links in a browser (web traffic versus **API** traffic): AGSI+ **https://agsi.gie.eu/api?type=eu**

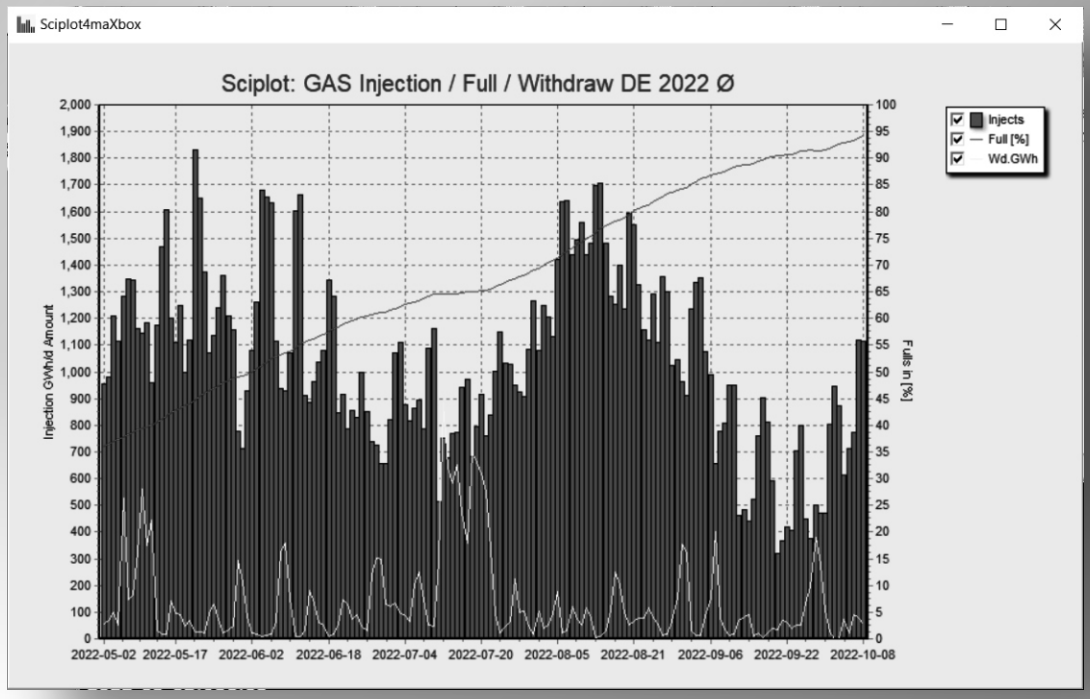


Figure 4.

AGSI DATA STORAGE

The scripts and images can be found: https://github.com/maxkleiner/agsi-data

REFERENCE:

https://agsi.gie.eu/api

https://www.gie.eu/transparency-platform/GIE_API_documentation_v006.pdf

https://svn.code.sf.net/p/alcinoe/code/demos/ALWinInetHTTPClient/_source/Unit1.pas

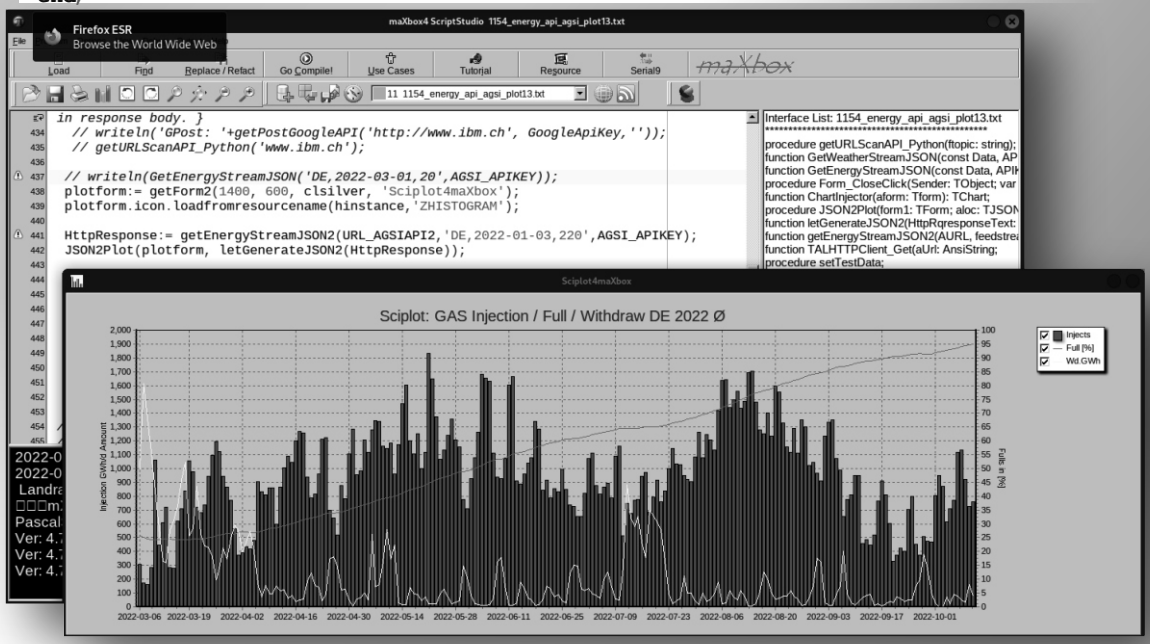
https://docwiki.embarcadero.com/Libraries/Sydney/en/System.Net.HttpClient.THTTPClient.Post

Doc and Tool: https://maxbox4.wordpress.com

Script Ref: 1154_energy_api_agsi_plot14.txt

APPENDIX: shows an WinAPIDownload class from maXbox4 integration

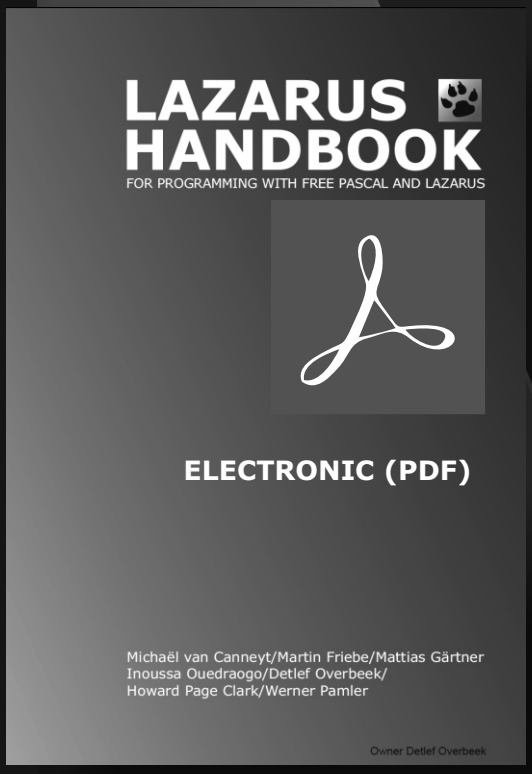
```
{*-------------------------------------------------------------------------*)
TWinApiDownload = class(TObject)
private
  fEventWorkStart: TEventWorkStart;
  fEventWork: TEventWork;
  fEventWorkEnd: TEventWorkEnd;
  fEventError: TEventError;
  fURL: string;
  fUserAgent: string;
  fStop: Boolean;
  fActive: Boolean;
  fCachingEnabled: Boolean;
  fProgressUpdateInterval: Cardinal;
  function GetIsActive: Boolean;
public
  constructor Create;
  destructor Destroy; override;
  function CheckURL(aURL: string): Integer;
  function Download(Stream: TStream): Integer; overload;
  function Download(var res: string): Integer; overload;
  function ErrorCodeToMessageString(aErrorCode: Integer): string;
  procedure Stop;
  procedure Clear;
  property UserAgent: string read fUserAgent write fUserAgent;
  property URL: string read fURL write fURL;
  property DownloadActive:Boolean read GetIsActive;
  property CachingEnabled:Boolean read fCachingEnabled write fCachingEnabled;
  property UpdateInterval:Cardinal read fProgressUpdateInterval
                  write fProgressUpdateInterval;
  property OnWorkStart: TEventWorkStart read fEventWorkStart
                    write fEventWorkStart;
  property OnWork: TEventWork read fEventWork write fEventWork;
  property OnWorkEnd: TEventWorkEnd read fEventWorkEnd write fEventWorkEnd;
  property OnError: TEventError read fEventError write fEventError;
end;
```

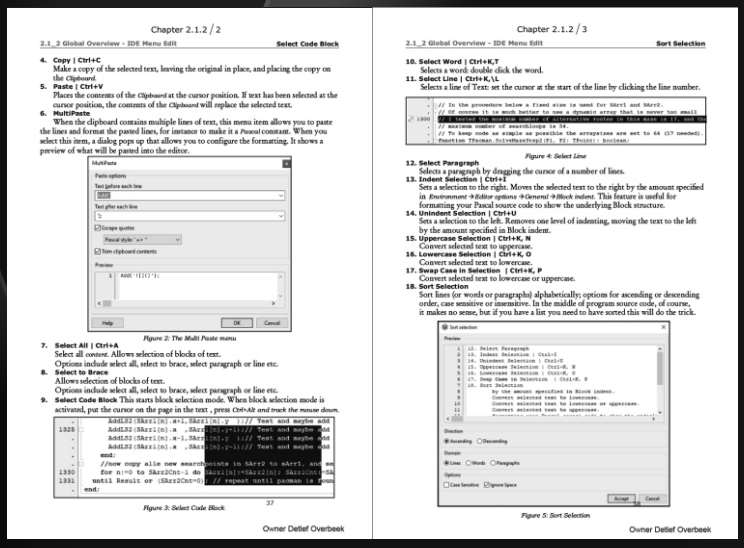


Lazarus Handbook PDF

- English
- Black & white edition
- 934 Pages
- Signed with your own name
- Electronic indexed
- Index of 240 pages
- List of chapters is completely clickable for moving to the page
- Including 40 Example projects and extra programs

- BPM PDF VIEWER for Windows included

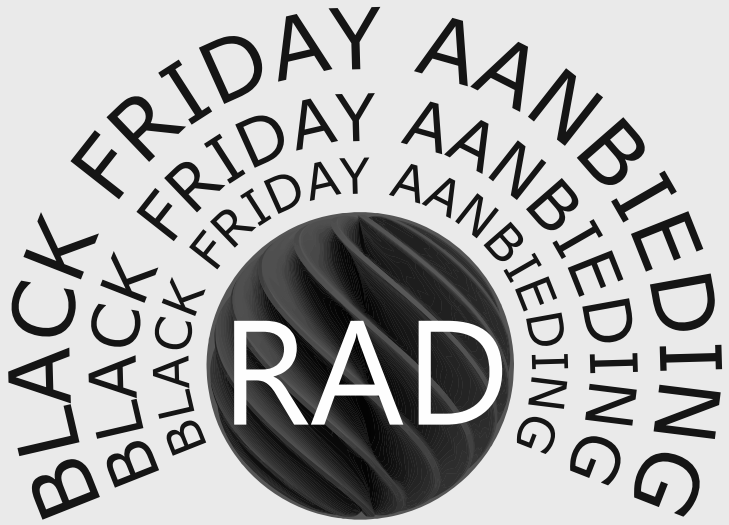




https://www.blaisepascalmagazine.eu/product-category/books/

barnsten

AANBIEDING GELDIG TOT&MET 2 DECEMBER 2022



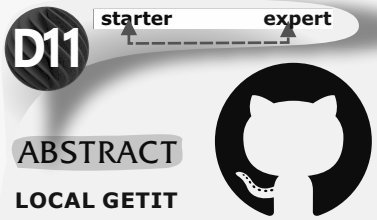
20% korting op Delphi,
C++Builder en
RAD Studio Professional licenties

30% korting op Delphi,
C++Builder en
RAD Studio Enterprise en Architect licenties

tel:+31 23 54 2222 7
https://www.barnsten.com/nl

A UNIVERSAL WAY TO INSTALL COMPONENTS AND LIBRARIES INTO DELPHI

BY DANNY WIND



ABSTRACT

LOCAL GETIT

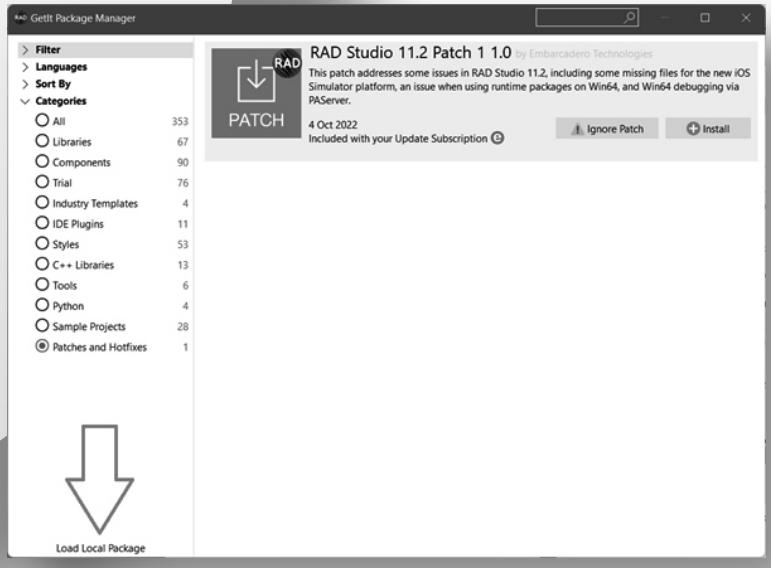
Contrary to its name, local **GetIt** is not just local.
It's an universal way to install components and libraries into Delphi using a JSON text format install file and supports both local files and remote files.
When using local files it can run without a network connection and you can use it for offline distribution. You can also point the elements in the JSON file to remote files from a website or a GitHub repository

INTRODUCTION

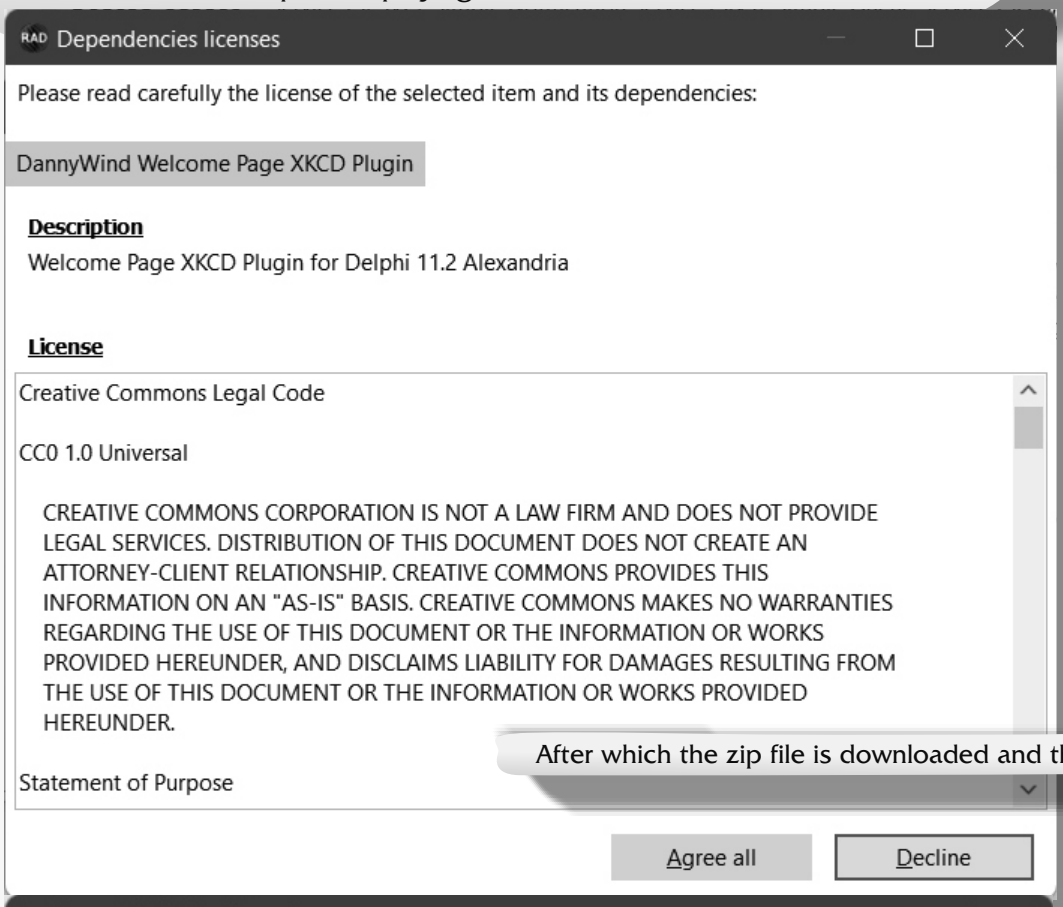
The most basic **JSON local GetIt** file will just have a few entries. Some descriptive info, and a link to a **`zip`** file, a local file or download from a **HTTPS** address.
It will then copy or download and unpack the zip file and compile the project, installing it into **Delphi.**
In this introduction to **local GetIt** we will be describing just such a basic **JSON local GetIt** file.

The really nice thing about this **JSON** file based install, is that it allows **Delphi** developers to share their components and libraries with other developers in a quick and easy way.
It relies on a human readable **JSON** text file, without having to use external installers or **`powershell`** scripts.
As a **Delphi** developer we just need to add one **JSON** file to our public **GitHub** repository, which makes installing, using and uninstalling our components so much easier.
Also, because it's human readable, anyone installing your component library will be able to see exactly what the installer does.
No hidden shenanigans.

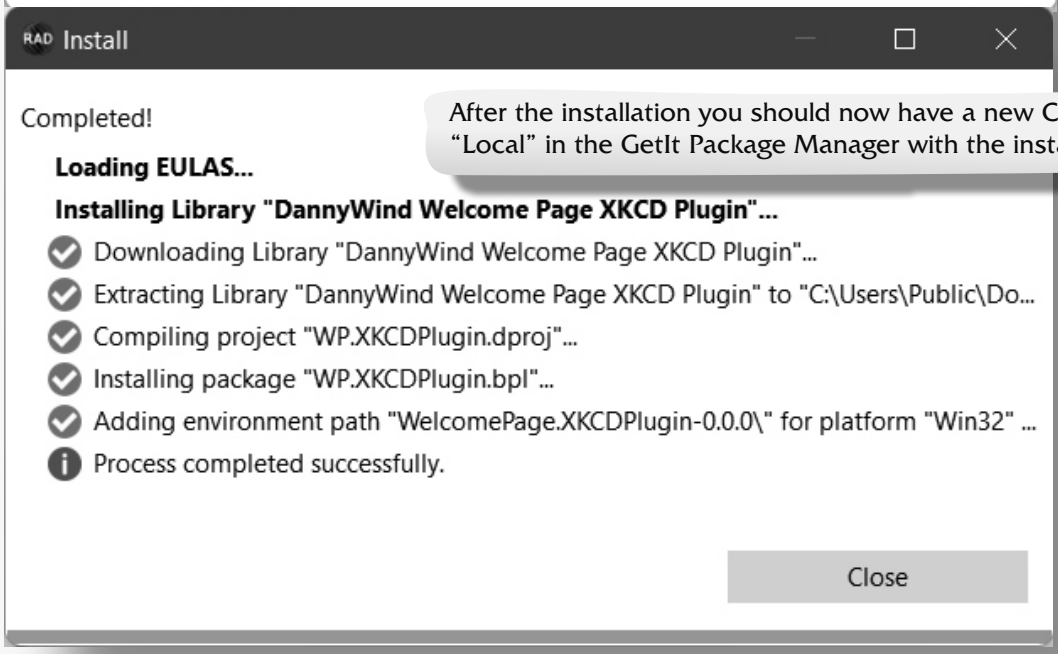
How does this work in the **Delphi IDE?**
You download a **GetIt JSON** file,
for example from a **GitHub** repository,
and save it locally.
Then you open the **GetIt Package Manager** from the **Tools** menu and click on the small button in the bottom left
"Load Local Package".



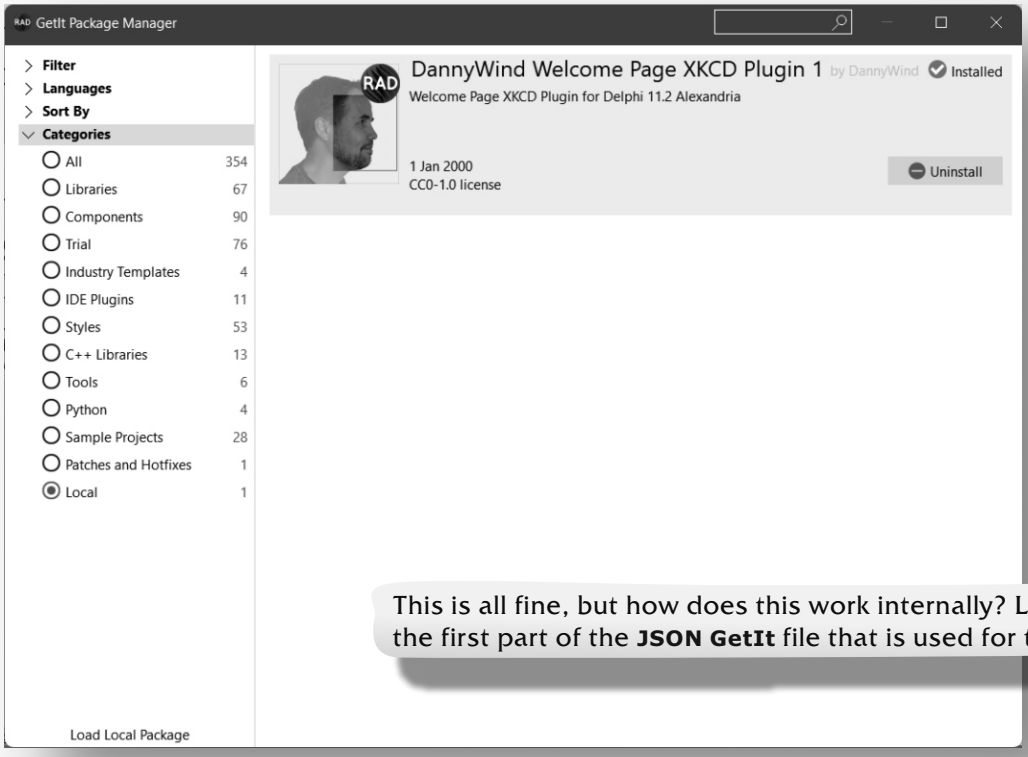
Currently the use of **local GetIt** is tied to having an active subscription.
If your subscription is not active you might not see this button.
Then select the **JSON file** and **local GetIt** will perform its actions to install it.
One of the default steps is displaying the license terms.



After which the zip file is downloaded and the installation is run.



After the installation you should now have a new Category
"Local" in the GetIt Package Manager with the installed package visible.



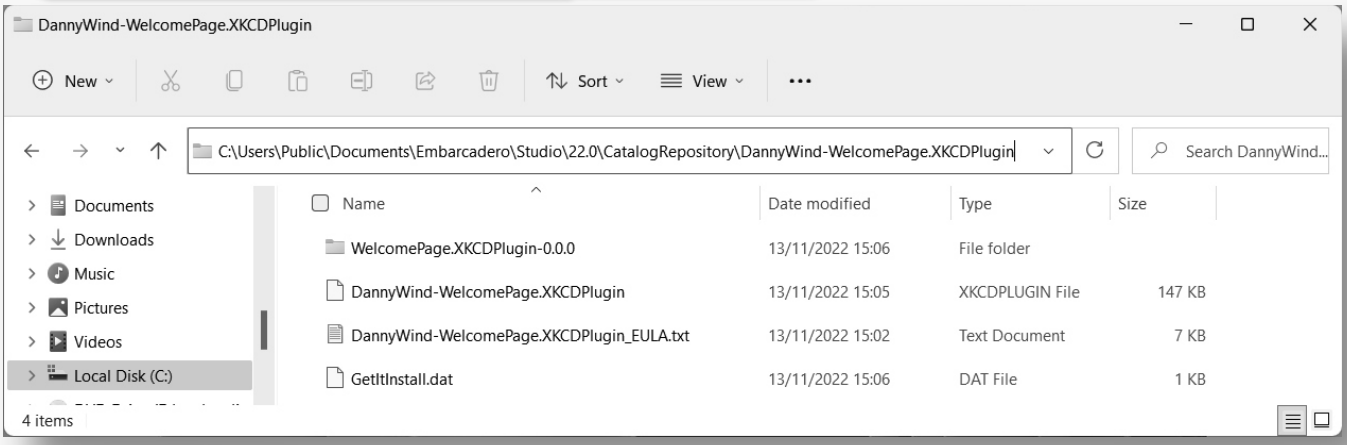
This is all fine, but how does this work internally? Let's take a look at the first part of the **JSON GetIt** file that is used for this install.

```
{
        "Id": "DannyWind-WelcomePage.XKCDPlugin",
        "Name": "DannyWind Welcome Page XKCD Plugin",
        "Version": "1",
        "Description": "Welcome Page XKCD Plugin for Delphi 11.2 Alexandria",
        "Vendor": "DannyWind",
        "VendorUrl": "https:\/\/dannywind.nl",
        "Image": "https:\/\/avatars.githubusercontent.com\/u\/78543577",
        "Tags": "DannyWind",
        "License":
        "https:\/\/github.com\/DannyWindnl\/WelcomePage.XKCDPlugin\/raw\/main\/LICENSE",
        "Url":
        "https:\/\/github.com\/DannyWindnl\/WelcomePage.XKCDPlugin\/raw\/main\/
                                                    WelcomePage.XKCDPlugin-0.0.0.zip",
        "ProjectUrl": "https:\/\/github.com\/DannyWindnl\/WelcomePage.XKCDPlugin",
        "Modified": "2022-09-19 16:42:00",
        "LicenseName": "CC0-1.0 license",
        "TargetPath": "",
        "RequireElevation": "0",
        "AllUsers": "1",
        "Actions": [ [
        ]
}
```

This **JSON** file installs a **Welcome Page** plugin into **Delphi 11.2** from a **GitHub** repository.
In order to avoid redirection we use the raw links to the **GitHub** files.
The license file is required, when installing the **GetIt** installer displays it and asks if the user is OK with it. If so, it downloads the **zip** file from the given **Url.**
Now it says Url, but you can also specify a local file here.

Using a local file in the same folder as the JSON file:
"Url": "WelcomePage.XKCDPlugin-0.0.0.zip",

It then proceeds to unzip this file in the **CatalogRepository** directory



The zip file contents are unzipped into the subdirectory of the package with the name of the zip file **`WelcomePage.XKCDPlugin-0.0.0`**.
Several things stand out from the above format and identifiers in the **JSON** file.
We need to escape forward slashes.
A Url that starts with **`"https://"`** with escape characters looks like **`"https:\/\/"`** in the **JSON** file.
Also make sure you use the correct simple double quotes and not copy paste them from a **Word** document.

In this example I used **`AllUsers`** with a value of 1 to install in **`$(BDSCatalogRepositoryAllUsers)`** as visible in the image, while the recommended default is 0, which installs in **`$(BDSCatalogRepository)`** and limits the install to the path of the current user.

The **`Id`** is required and needs to be unique. **`Id`** is also used to determine the order in which actions are executed.
The Image will be displayed after the **GetIt** in the Installed local packages page. The **`Name`** is used to search for the package in all of the installed GetIt packages.

With only the first part shown, the **`Actions`** array is still empty and we need to fill it with actions to actually install the plugin. Our first action will be to compile the project.

```
"Actions": [
{
        "Id": "1",
        "ActionId": "6",
        "Type": "2",
        "RequireElevation": "0",
        "Parameter": [
                {
                        "Parameter": "WelcomePage.XKCDPlugin-0.0.0\\WP.XKCDPlugin.dproj"
                },
                {
                        "Parameter": "Win32"
                },
                {
                        "Parameter": "Release"
                }
        ],
        "ActionName": "CompileProject",
        "Description": "Compile WP.XKCDPlugin.dproj"
},
```

A couple of things to note here. The **ActionId** determines the **Action** that the **Delphi IDE** should perform. For **ActionId** 6 the action is **CompileProject**. **NOTE** that the **ActionName** is completely optional and is just informative. However it is good to add this as it makes the **JSON** file more readable. **ActionId** 6 has three parameters, the project file to compile, the list of platforms to compile *(Win32)* to and the configuration to use *(Release)*.

After compiling we want to install the Package into the IDE.

```
{
	"Id": "4",
	"ActionId": "7",
	"Type": "3",
	"RequireElevation": "0",
	"Parameter": [
		{
			"Parameter": "WP.XKCDPlugin.bpl"
		}
	],
	"ActionName": "InstallPackage",
	"Description": "Installs WP.XKCDPlugin.bpl package"
},
```

ActionId 7 is InstallPackage and again the ActionName is purely descriptive.

NOTE that the **Type** here is 3 instead of 2.
The **Type** denotes the **Event** on which the **Action** needs to be performed.
For **Type** 2 the **Event** is **after download**.
For **Type** 3 the **Event** is **before install**.
As part of the install we also want to set the path for this package.

```
{
	"Id": "5",
	"ActionId": "1",
	"Type": "3",
	"RequireElevation": "0",
	"Parameter": [
		{
			"Parameter": "WelcomePage.XKCDPlugin-0.0.0\\"
		},
		{
			"Parameter": "cPasLibraryPath"
		},
		{
			"Parameter": "Delphi.Personality"
		},
		{
			"Parameter": "Win32"
		},
	],
	"ActionName": "AddOptionPath",
	"Description": "Add library path for Delphi"
},
```

`ActionId 1` is `AddOptionPath`,
where `Parameter 1` is the Path,
`Parameter 2` has different values for the **Delphi Pascal Library** `(cPasLibraryPath)`,
the Delphi Browsing Path `(cPasBrowsingPath)` and some more.
The third Parameter originates from the `ToolsAPI.pas` file, which lists the personalities that can be installed in the **Delphi IDE**.
The fourth parameter is optional and determines for which platform the path is added.
You can add an optional fifth parameter, a boolean, which when set to "`False`" only adds the path to one of the compilers combined with the second parameter.
The default value is "`true`", which in most cases works out just fine.

At some point the package will be uninstalled.

```
{
	"Id": "14",
	"ActionId": "8",
	"Type": "5",
	"RequireElevation": "0",
	"Parameter": [
		{
			"Parameter": "WP.XKCDPlugin.bpl"
		}
	],
	"ActionName": "UninstallPackage",
	"Description": "Uninstalls WP.XKCDPlugin.bpl package"
},
```

Where Type 5 is linked to Event "Before Uninstall". And the next step is to remove the added path.

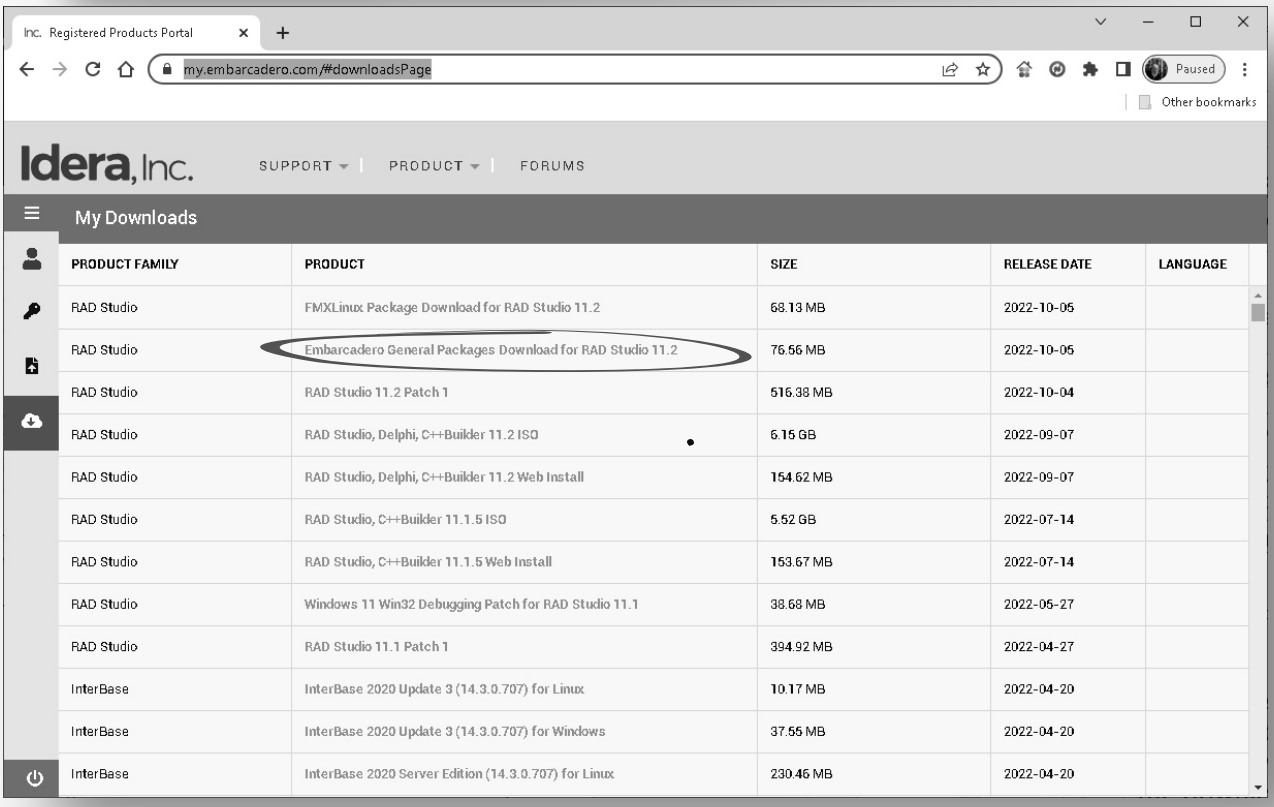
```
{
	"Id": "15",
	"ActionId": "2",
	"Type": "5",
	"RequireElevation": "0",
	"Parameter": [
		{
			"Parameter": "WelcomePage.XKCDPlugin-0.0.0\\"
		},
		{
			"Parameter": "cPasLibraryPath"
		},
		{
			"Parameter": "Delphi.Personality"
		},
		{
			"Parameter": "Win32"
		}
	],
	"ActionName": "RemoveOptionPath",
	"Description": "Remove library path for Delphi"
}
```

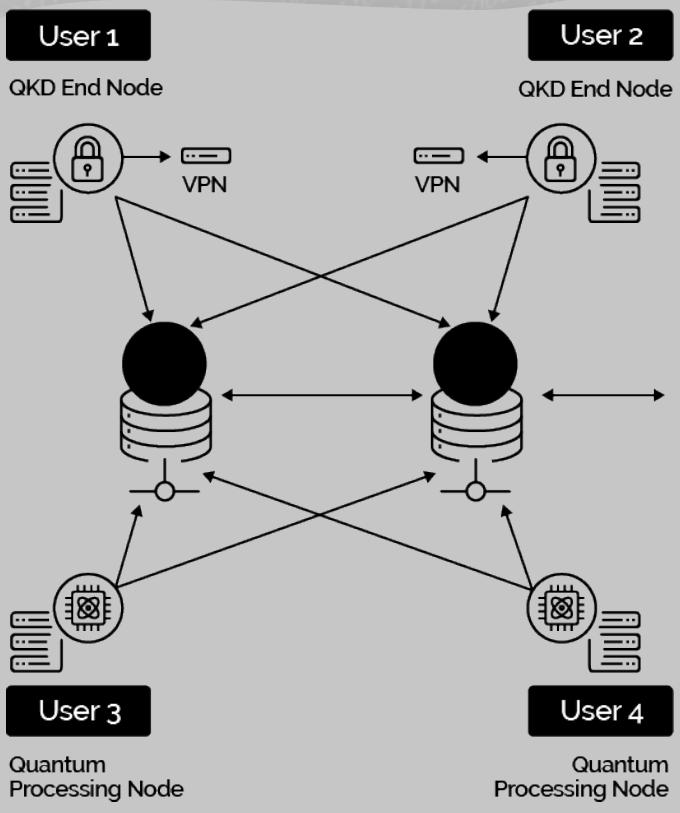
Also linked to the "Before Uninstall" event.
The six events you can use are listed in the table.

EventId "Type"	
❶ Before Download	Executed before the package zip is downloaded
❷ After Download	Executed after the package zip is downloaded and extracted to the CatalogRepository
❸ Before Install	Executed before the installation of the package
❹ After Install	Executed after the installation of the package
❺ Before Uninstall	Executed before the package is removed from the CatalogRepository
❻ After Uninstall	Executed after the package and its files have been removed from the CatalogRepository

Embarcadero is working on finalizing the documentation for **local GetIt** support and it is not yet available at the time of writing this article.
You can however download multiple samples of **local GetIt JSON** files through the downloads section at
`https://my.embarcadero.com/`
where you will find some bundled **local GetIt** installs meant for offline installation.
The "Embarcadero General Packages Download for RAD Studio 11.2" is a good place to get started and includes multiple **JSON** examples for installing packages that you can use as the base for your own local **GetIt JSON** file. *(See the picture below)*

The **local GetIt** sample for this article can be found at the **GitHub** project page at
`https://github.com/DannyWindnl/WelcomePage.XKCDPlugin`





Finally, the first steps have been taken to secure the Internet. This is much needed especially for Rotterdam's port authority. Just think of the financial and administrative dangers. Moreover, it could mean better exclusion of the Underworld. Port of Rotterdam stakeholders can participate and benefit from an "untapped", multi-user quantum network for their critical communication systems.

WHAT DOES THAT MEAN?

A whole range of enterprises and companies in different countries are already experimenting with quantum communications. But Dutch company **"QuTech"** with a spin-off company **"Q*Bird"** is the first to deploy a new type of secure quantum network. This can connect multiple users via a central hub in a scalable way (*allowing for permanent expansion*). This provides an untapped (*unbreakable*) internet connection between multiple users, spread across the entire port area.

The port of Rotterdam represents an important part of the Dutch economy. Rotterdam handles nearly 15 million sea containers every year, making it one of the largest ports in the world. Securing its communication systems improves the safety of tens of thousands of seagoing vessels every year and the resulting economic traffic. Securing information transmission over the internet is crucial for any large organisation. Fraud in information channels has significant financial consequences and can even be potentially life-threatening.

Q*bird
quantum
connectivity

ECONOMIC IMPACT

The port of Rotterdam is a crucial industrial and logistics hub, handling nearly 15 million sea containers annually, with dry bulk, liquid bulk (such as oil) and general cargo totalling nearly 500 million tonnes of transshipment.

The port's total economic added value represents 8.2% of Dutch GDP (€63 billion) and employs more than 500 000 people directly and indirectly. Eavesdropping on communication systems between parties could lead to significant financial losses, disruption of critical operations or physical damage.

A quantum communication infrastructure contributes to an untapped connection and improves the logistics chains of which Rotterdam is a part, and therefore a vital infrastructure on the European continent.

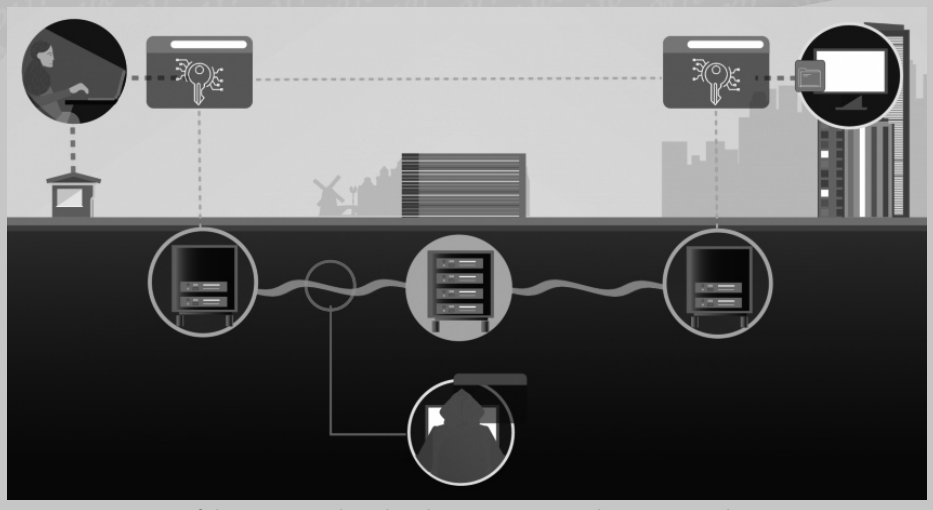


Figure 1: Diagram of the quantum key distribution system. Schema created using an Image: Simplot for QuTech.
A central node (C) connects users Al(left) and B(right).
When an eavesdropping hacker tries to brute-force the secret keys,
the laws of quantum mechanics ensure that the users are informed if the keys are compromised. A new set of keys is then created to securely encrypt further messages.

SPECIAL QUANTUM KEY DISTRIBUTION TECHNIQUE DEVELOPED BY Q*BIRD

Earlier, a team of scientists and engineers from **QuTech** - a collaboration between **Delft University of Technology** and **TNO** - demonstrated an alternative, untapped method of transmitting encrypted information.
The technology has such commercial potential that they have decided to split from **QuTech** under the name "Q*Bird".
This technology is based on a special implementation of quantum key distribution **(QKD)** which uses a central hub to connect users who want to exchange secure communications. That means it is possible to cost-effectively implement a quantum Internet, scalable to multiple end-users and using mostly off-the-shelf equipment."

WIKIPEDIA **(Quantum key distribution (QKD)** is a secure communication method which implements a cryptographic protocol involving components of quantum mechanics. It enables two parties to produce a shared random secret key known only to them, which can then be used to encrypt and decrypt messages. It is often incorrectly called quantum cryptography, as it is the best-known example of a quantum cryptographic task.

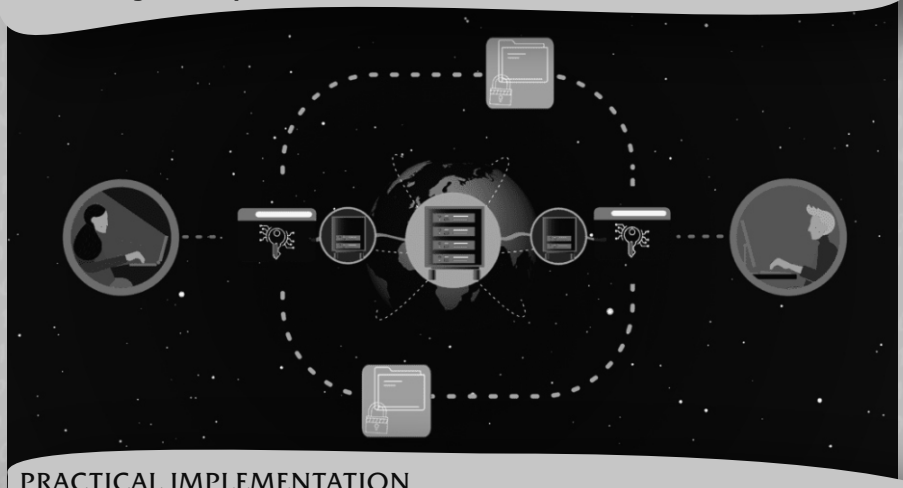
An important and unique property of quantum key distribution is the ability of the two communicating users to detect the presence of any third party trying to gain knowledge of the key. This results from a fundamental aspect of quantum mechanics: the process of measuring a quantum system in general disturbs the system.
A third party trying to eavesdrop on the key must in some way measure it, thus introducing detectable anomalies. By using quantum **superposition's** or quantum **entanglement** and transmitting information in quantum states, a communication system can be implemented that detects eavesdropping.
If the level of eavesdropping is below a certain threshold, a key can be produced that is guaranteed to be secure (*i.e., the eavesdropper has no information about it*), otherwise no secure key is possible and communication is aborted.

The security of encryption that uses quantum key distribution relies on the foundations of quantum mechanics, in contrast to traditional public key cryptography, which relies on the computational difficulty of certain mathematical functions, and cannot provide any mathematical proof as to the actual complexity of reversing the one-way functions used. **QKD** has provable security based on information theory, and forward secrecy.

(Quantum key distribution (QKD) continuation WIKIPEDIA

The main drawback of **quantum key distribution** is that it usually relies on having an authenticated classical channel of communications. In modern cryptography, having an authenticated classical channel means that one has either already exchanged a symmetric key of sufficient length or public keys of sufficient security level. With such information already available, in practice one can achieve authenticated and sufficiently secure communications without using **QKD,** such as by using the Galois/Counter Mode of the Advanced Encryption Standard. Thus **QKD** does the work of a stream cipher at many times the cost.

Quantum key distribution is only used to produce and distribute a key, not to transmit any message data. This key can then be used with any chosen encryption algorithm to encrypt (and decrypt) a message, which can then be transmitted over a standard communication channel. The algorithm most commonly associated with **QKD** is the one-time pad, as it is provably secure when used with a secret, random key. In real-world situations, it is often also used with encryption using symmetric key algorithms like the **Advanced Encryption Standard algorithm.)**



PRACTICAL IMPLEMENTATION

As a first step in the test, the central hub for the distribution of quantum keys will be hosted at the **Port of Rotterdam Authority**.

Based on this new technology, data will be exchanged between a number of parties in the port in a shielded environment. Participating in this test will be the **Port of Rotterdam Authority, Portbase and a number of nautical service providers.** The aim of the test is to further validate the technical capabilities of the system.

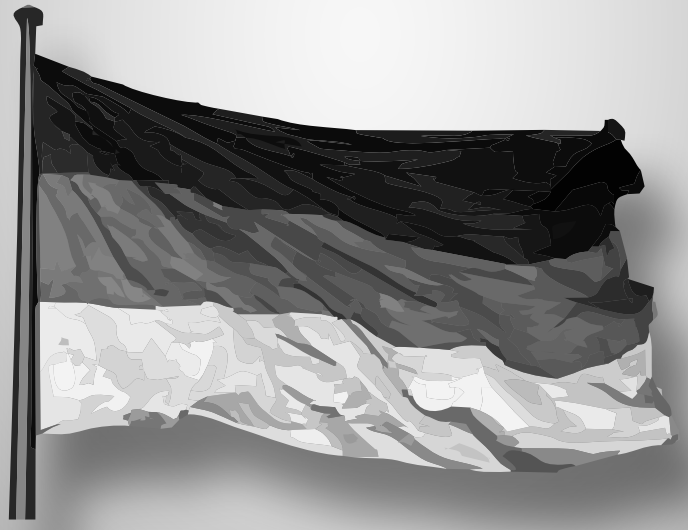
In the future, such a set-up will make it possible to provide all end users with a secure, undetectable connection.

Users will share keys generated using quantum technology (*and therefore inherently non-break-in, untappable as the keys are via entanglement at a stage where they cannot be accessed without changing the key*), which they will then use to encrypt messages using traditional technology.

The strength of this setup is the ease with which it can be extended to many more users, and the relatively low cost of expansion. Upon completion, the parties involved can rest assured that their communication line has not been tampered with.

From a technical perspective, this project enables close cooperation between **"Single Quantum"** and **"Q*bird"**: two Dutch high-tech companies with complementary expertise and global ambitions. By joining forces in this project, they will have a powerful social impact.

This issue is also available as a German Edition and downloadable for free



If you want a free German Issue
send your adressdetails and email address
to **editor@blaisepascalmagazine.eu**

A NEW (EMBEDDED) UNDO SYSTEM FOR AN ALGEBRAIC TEXT EDITOR

BY DAVID DIRKSE

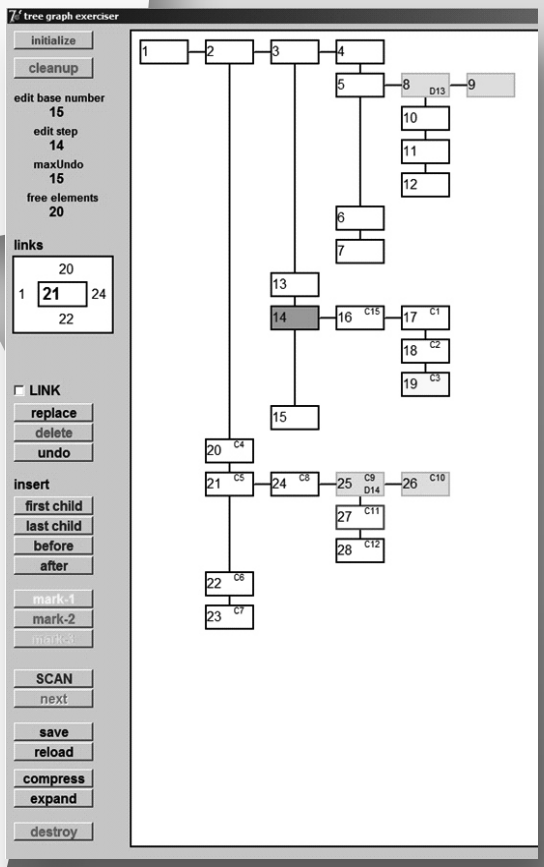
7 starter expert D11

INTRODUCTION

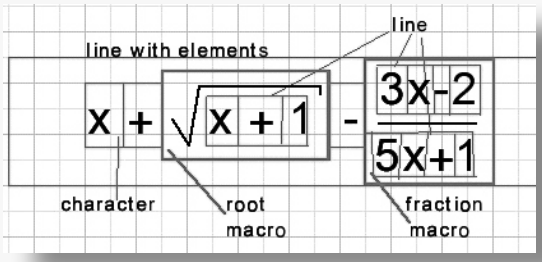
Algebraic text needs extra elements to implement roots, fractions, exponents, indices, vectors, matrices and also special symbols for geometry, sets and logic.
In this article I describe the basic structure of such an editor.
Not the drawing procedures but the edit processes which involve element insert, replace, delete and undo operations.
The editor is **WYSIWYG.** Text is typed in the usual way. Special elements are selectable from menus by a mouse-click. No scripts are used.
The picture below shows two examples, each typed in a few seconds:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

$$e = \lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x$$



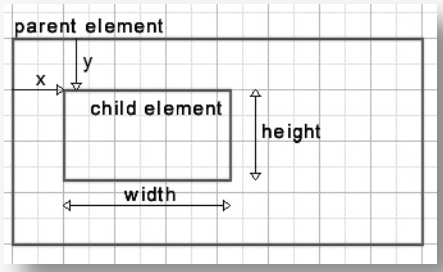
This is the general idea:
Elements are added to a page, which is the first (number 1) element.
Text element types are page, frame, table, character, line and macro.
Macro elements contain a line (or lines) and graphic elements for roots, parenthesis, fractions ...
A line holds characters and macro's.



Text alignment is fully automated. If text is typed in the line of a root, the root symbol adjusts its size. If text is typed in the denominator line of a fraction, the numerator's position is corrected to remain centered. Parenthesis around a line are enlarged if a fraction macro is added which increases the height of the line.

ROOTED TREES

If the dimensions of element A depend on the size of elements B and C we call A the parent of B and C.
B and C are the children of A.
For the position and dimension calculation an element needs these properties:



X: the left position relative to its parent
Y: the top position relative to its parent
Width, height : element size.

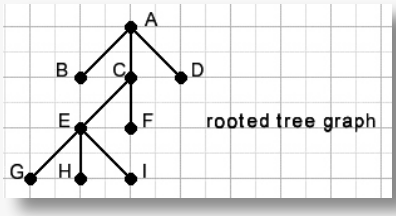
Note that changing position of the parent does not modify the x,y coordinates of the children.

The general idea is: if a child's size is modified, the parent recalculates the position of its children as well as its own size. If the parent's size changes, its parent recalculates….etc.

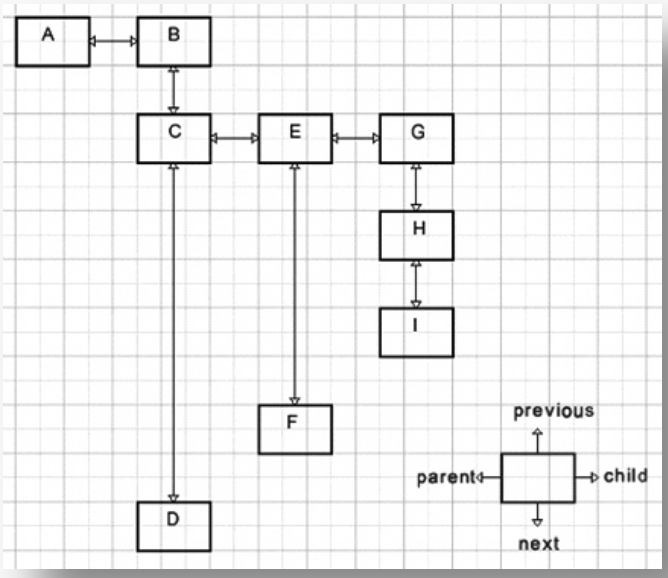
Element type specific procedures are:

- Calculating its own width and height
- (Re)Calculate the (x,y) position of its children
- Paint itself on a canvas

An element never calculates its own (x,y) position.
Edit procedures (insert, delete, replace, undo) are abstract, element type independent.
The diagram which describes the element dependencies is called a rooted tree which is a special graph. This is how a rooted tree is pictured in mathematics:



The dots are elements, downward lines connect to children.
(In mathematics, the dots are called vertices, the lines are called edges)
In tree graphs there is only one path between dots. No loops exist. In above picture B,C,D are children of root element A. C is the parent of E and F. G,H,I are children of E.
Note: this is an Australian tree, the root is at the top and the branches grow downward.
Above structure is also known from file systems with folders which contain files and folders….
For our purpose the mathematical representation is not suitable, the next picture is better:



To test the Xtree procedures an exerciser program is written.

```
const maxElement = 48; //maximum number of elements in exerciser
type TElementStatus = (esFree,esActive);//esFree: outside tree
 Telement = record
       status : TElementStatus;
       mark  : byte; //property 0,1,2,3 for background color
       posX,posY : byte;//position relative to parent
       parent : word;
       child : word;
       next  : word;
       previous : word;
       Cstep : byte;   //create step sequence number
       Dstep : byte;   //delete (replaced) step number
      end;
var element : array[1..maxelement] of Telement;
```

STATUS:

- **esFree** : element is not active, is not part of tree, no link to other element.
- **esActive** : element is in use, part of the tree

mark is a code for the background color of the element. 0 is white.
Elements are painted on a bitmap (map) which is regarded a table of columns and rows.

posX is the column, posY is the row.

Parent, child, next, previous :
links to other elements to define the tree structure. If no link, this value is 0.

Cstep, Dstep bytes enable undo operations, see later for explanation.

Note that the element array holds a bidirectional linked list. This is convenient for easy cursor movement.

EDIT OPERATIONS

Insert

- placeA……place new element after other element
- placeB……place new element before other element
- placeChildA….place new element as last child
- placeChildB….place new element as first child

Delete de-activates element including its children.

Replace combines a delete and insert operation resulting in element replacement. Children of the deleted element are connected to the replacing element.

Undo removes last added element or re-activates a deleted element.

THE NEW UNDO SYSTEM

In earlier versions of the math editor, the document was the first element, pages were children of the document. In later versions the document element was eliminated, the page became the root. This resulted in a much smaller element array, now containing only the page being edited. Page data is stored on the heap and loaded in the element array only if the page is edited.

The limited number of elements allow for this embedded undo system.
For undo operations edit actions must be remembered per page. Before, edit data was pushed on a stack and popped for undo. Switching between pages erased the stack data. This was inconvenient. In this new undo system deleted elements are no longer extracted from the tree but are only in-activated, holding their position in the tree.

For the undo processes two counters had to be introduced:

- **edStep** : edit step sequence number
- **edBase** : edit base number

and a constant

- **undoDepth** = 15, the number of edit steps that may be taken back

Initially `edBase=edStep = 0`. No editing took place yet

Before adding the first element:
edBase and **edStep** are both set to 1.
The new element's `Cstep := edStep` indicating that this element was created at step 1.
For each new character **edStep** is incremented and its Cstep value is set to edStep.
After adding 14 more characters **EdStep** is 15, **edBase** still is 1.
15 is the maximum number of steps to be remembered.

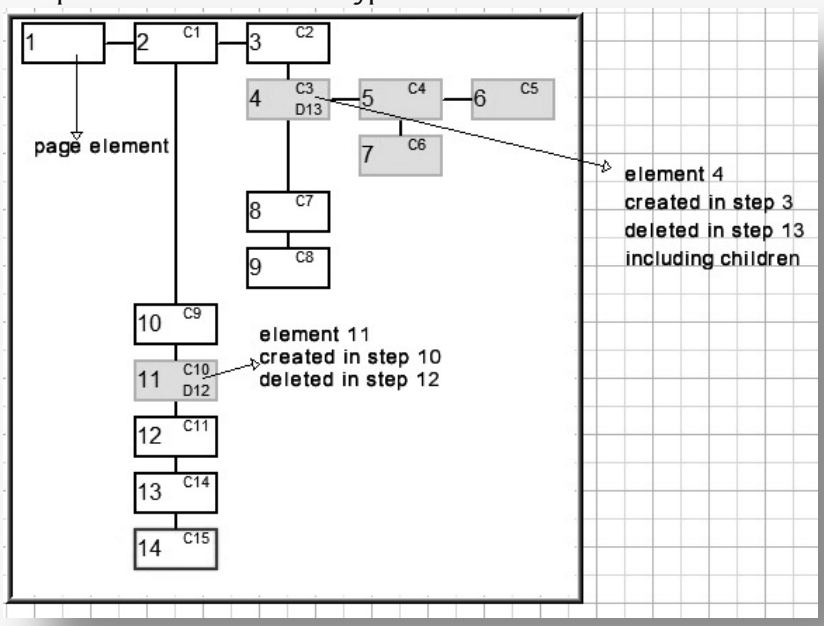
So before adding the next element edStep is incremented to 16, exceeding the maximum number (15) of steps. Now edStep is reset to 1 and edBase is incremented to 2,
The added element will have `Cstep:=1` but there already exists an element with `Cstep=1`;
So, for this element `Cstep:=0` placing it outside the undo range.
The new undo range is 2,3,….,15,1.
So, for each edit step the complete tree has to be searched.
This is no problem because the maximum number of elements for a single page of the math editor is only 7500.
If repeated undo's are issued from this point the elements with `Cstep=1,15,14,….3,2` are removed from the tree. If `edStep=edBase` after decrementing, there are no more undo actions possible and `edBase := 0` and `edStep:=0;`

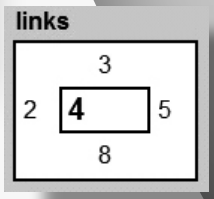
LINKING

Holding a character key down issues a row of the same character.
Adding a macro element also adds line elements.
It is convenient to remove these elements at a single undo.
Therefore a **`linkflag`** exists which prevents the edStep counter from incrementing.
The **`starttreelinking`** procedure sets the startflag. The next edit operation then sets the linkflag.

TREE EXERCISER DISPLAY

The picture below shows a typical tree structure:

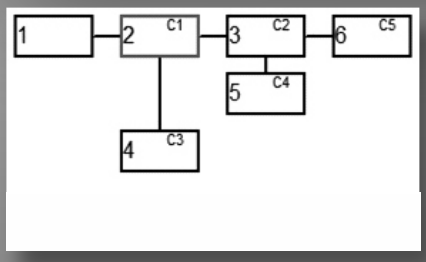


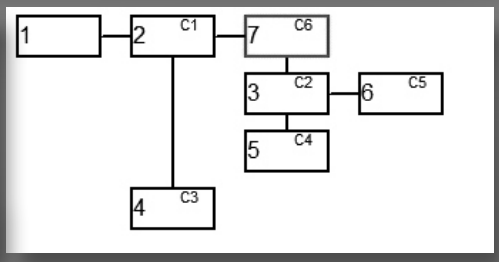


DELETED elements are painted with a grey background.
In above picture element 14 is the active (red edge) element.
Edit operations apply to this element.
The active element is selected by a mouse-click or by the cursor keys.
The parent-child links and also the previous-next links between elements are painted as a single line. To see the exact links place the mouse pointer over the element.
Then the links are displayed in a separate paintbox:

ADDING elements

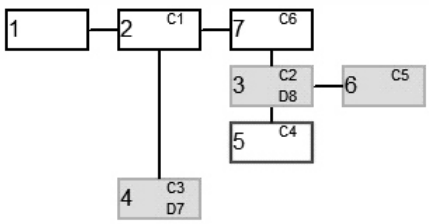
This involves selecting a free element, set its properties, set the links and call `procedure LinkIn( )` to connect the new element to the tree.
In the picture below element 7 is added as the first child of active element 2





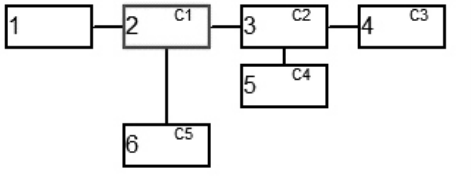
DELETING ELEMENTS

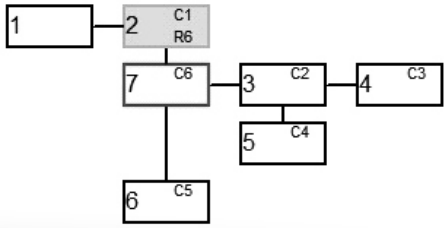
See picture. Elements 4 and 3 are subsequently deleted.
Note: element 6 has Dstep=0 but is painted grey because its parent (3) is deleted.



REPLACING ELEMENTS

In picture below, element 2 is replaced by a new element (7)





The new element is placed next to the replaced element. The children of element 2 are moved to the new element 7. See element 2: R6 means: replaced in edit step 6. The parent link of children 3,5 must be changed to 7.

Undo

If an element (say 20) is deleted in step 10 its Dstep value is set to 10.
If later Edstep increments to 10, element 20 falls outside the undo range and is destroyed together with its children.
Destroy means that it is removed from the tree (by procedure Extract), its links are set to 0 and its status becomes esFree making it available as new element.

Recognizing replaced elements

Deleting an element also deletes it's children.
A replaced element however has its children attached to the replacing element.
Undo of a replaced element must re-attach the children.
To distinguish creation and deletion from replacement bit 7 of the Cstep and Dstep values is used. This bit being set indicates replacement.
The edStep counter being of type byte leaves a maximum possible undo count of $2^7 - 1 = 127$.

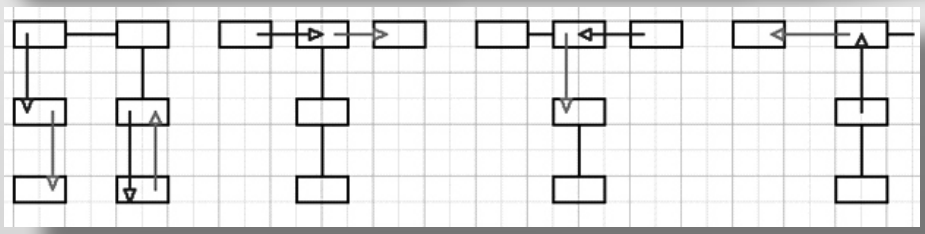
Scanning the tree

For several purposes, the tree must be searched in an orderly way.
The next scanning procedures are depth-first. Children are selected first, then next, then previous and at last the parent. The tree is traversed down to the bottom and then up to the root.

```
type TLink =(ltNone,ltParent,ltChild,ltNext,ltPrevious);
function ScanElement(var elmnt : word; lt : TLink): TLink;
```

lt is the link to element **elmnt**.
The **scanElement** function provides a new link to another element **elmnt** in the tree.

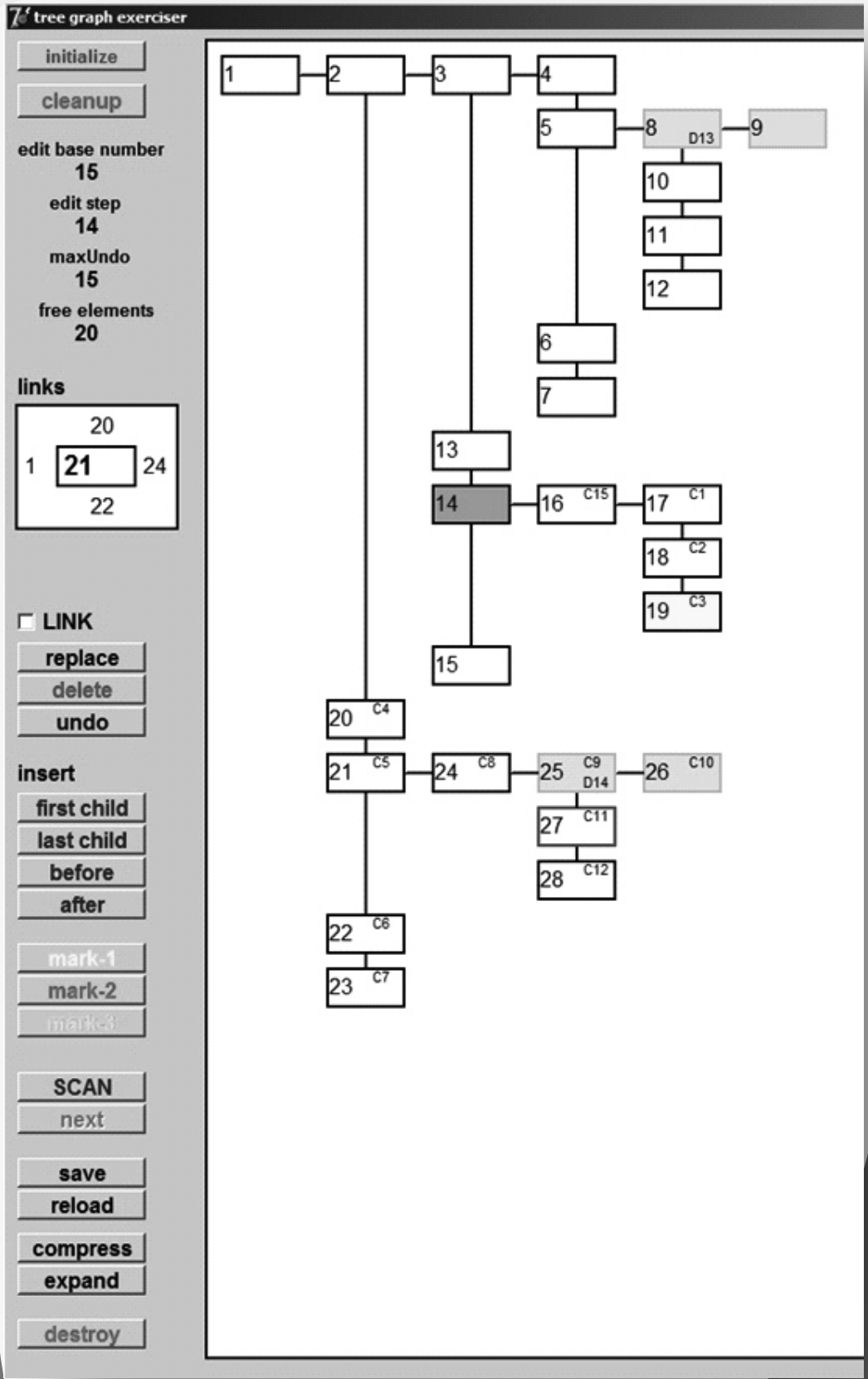
The next picture shows the scanning.
The blue line starts in the previous element and points to element elmnt. The green arrow points to the new element **elmnt** and the function result is the link.
Deleted elements are skipped.



A similar function `fullscan(var elmnt : word; lt : TLink) : TLink;`
Also selects the deleted elements in the tree.

SAVE AND RELOAD
To record the tree structure including the undo information a forward linked list (child, next links only) will do. There is no need to save the previous and parent links. After loading, the parent and previous links may be added.

COMPRESS AND EXPAND
Without saving the undo information, trees may be further compressed.
If elements B,C are children of A we write the tree as A(BC).
If D,E,F are children of B : A(B(DEF)C) defines the tree.
To save the tree structure , codes for '(' and ')' must be added but the links parent, child, next, previous may be neglected. There is also no need to store the element numbers.
While loading a compressed tree file new element numbers are assigned, starting at 1 for the page element.
The Xtree project
This project was built to test the new edit/undo procedures before their implementation in the math editor. Two units and one form exist:
Form1/Unit1: buttons, paintboxes, control, events, tree painting
Xtree2_unit: edit procedures.


tree graph exerciser
initialize
cleanup
edit base number
15
edit step
14
maxUndo
15
free elements
20
links
20
1
21
24
22
LINK
replace
delete
undo
insert
first child
last child
before
after
mark-1
mark-2
mark-3
SCAN
next
save
reload
compress
expand
destroy
1
2
3
4
5
8
D13
9
10
11
12
6
7
13
14
16
C15
17
C1
18
C2
19
C3
15
20
C4
21
C5
24
C8
25
C9
D14
26
C10
27
C11
28
C12
22
C6
23
C7

BUTTONS

Initialize	Destroys the complete tree. Set element 1 status to esActive, this is the root.
cleanup	Destroys deleted elements. Set **Cstep=0** for all elements. All **undo** information is erased.
Link	**record next** edit actions as one action by using the same **edStep** value.
Replace	selected element is replaced
delete	de-activate selected element and its children.
Undo	**Destroy** last added element and / or re-activate deleted elements.
insert	new element added (*after/ before selected element or as 1st/last child*)
mark (1,2,3)	**apply/remove** background color to/from selected element
scan	press **next** to select new element, children first.
Save/reload	tree including **undo** information
Compress/expand	**save/reload** tree without **undo** information, new element numbers are assigned on reload.
Destroy	remove selected element from tree.

DESCRIPTION OF THE XTREE2_UNIT

Please refer to the source code for details.

NAVIGATING PROCEDURES

```
function hasParent(el: word) : boolean;
function hasChild(el:word) : boolean;
function hasNext(el: word) : boolean;
function hasPrevious(el: word) : boolean;
```

Return true if element **el** has parent (**child,next,previous**) which is not deleted.

```
function getParent(var par : word; el : word) : boolean;
function getChild(var chld : word; el : word) : boolean;
function getNext(var nxt : word; el : word) : boolean;
function getPrevious(var prev : word; el : word) : boolean;
```

if element **el** has parent (child,next,previous) return true and the element number.

```
function ScanElement(var elmnt : word; lt : TLink) : TLink;
```

if element **el** has parent (child,next,previous) return true and the element number.

```
function ScanElement(var elmnt : word; lt : TLink) : TLink;
```

elmnt is the supplied element, lt is the link (from parent,child,next,previous) to **elmnt.**
A next element in the tree is returned in **elmnt,** the new link is the function result.

```
function fullScan(var elmnt : word; lt : Tlink) : Tlink;
```

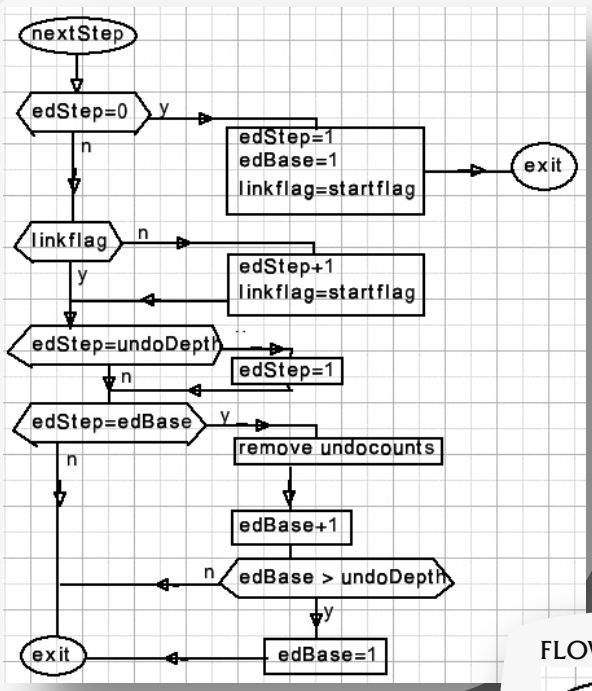
Is the same as **ScanElement** but does not skip deleted elements. This function is used to add previous and parent links after reloading and also for searching the tree to find elements outside the undo range.

EDITING PROCEDURES

`procedure PlaceA(del,sel : word);`	place element sel after del
`procedure PlaceB(del,sel : word);`	place element sel before del
`procedure PlaceChildA(del,sel : word);`	place element sel as last child of del
`procedure PlaceChildB(del,sel : word);`	place element sel as 1st child of del
`procedure replaceEL(del,sel : word);`	replace element del by sel.

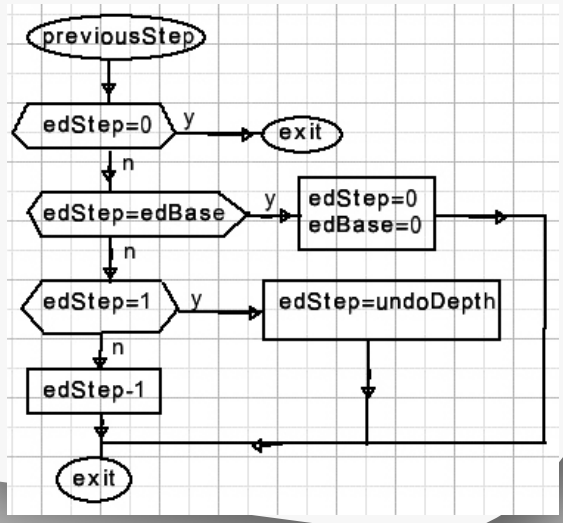
HIDDEN PROCEDURES:

`procedure Extract(el : word);`	remove element **el** from tree, repair links. note: **element[1]** (the root) cannot be extracted
`procedure LinkIn(el : word);`	**element[el]** has correct links, set links to el from parent child next previous elements.
`procedure NextStep;`	increment the `undoBase/undoStep` counter before an `insert` or `delete` operation. Search tree for elements having `Cstep=Edstep` or `Dstep=edStep.`



procedure previousStep;
decrement undoBase/undoStep counter after undo operation.

FLOWCHART:



procedure removeUndoCounts;

Test all elements:

If Cstep=edStep then set Cstep = 0,	this places the element outside the undo-range.
If Dstep = edStep then destroy element	as it falls outside the undo-range.
procedure setParentLinks(par : word);	set link of children to element par.

LINKING

Procedure StartTreeLinking	Sets the **startFlag.** At the next increment of **edStep,** the **linkFlag** is set which prevents further **EdStep** updating.
procedure stopTreeLinking	Clears both the **startFlag** and the **linkFlag.**

function TestELdeleted(el : word) : boolean;

Returns true if element or a parent was deleted. This function is called by the tree painting procedure.

ELEMENT DESTROY NOTIFICATION

Elements like frames, tables, matrices, coordinate systems and pictures need more properties and the element points to a property list. When such an element is destroyed in some cases also the entries in these lists have to be removed. For that reason before each destroy operation the xtree2_unit calls

procedure `destroyNotification(del:word);` to inform the editor that element del will be destroyed.

Unit1 description

The tree is painted on the canvas of bitmap map which is later copied to paintbox1.
It's size is 1000*980 pixels (H*V) divided in 10 columns and 24 rows of 80*40 pixels.
This leaves 2 columns at the right side were unused elements are pictured in the color grey.
The posX, posY properties of an element denotes the column and row.

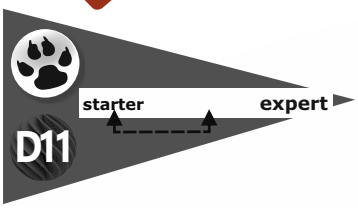
procedure `getActiveXY(var x,y:smallInt;px,py:byte);` translates the x, y column/row to pixel positions px,py.

For idle elements this calculation is done by **procedure** `getIdleXY(var x,y:smallInt;elnr:byte);`
An element is painted by **procedure** `paintElement(el:word);`
using the properties posX, posY which were calculated by
procedure `paintmap;` which repaints the complete tree.

Using **`procedure fullscan(..)`** the tree is traversed and in case of a child or next link the **`PFlag`** is set causing calculation of posX, posY and a call to paintElement. The highest Y position is saved in variable topY preventing elements to overlap.

procedure `repaintElement(elmnt:word);`
paints element element in the map and transfers the modified rectangle to paintbox1 on Form1. This is used when a new element is selected or the mark property is changed.

This concludes the description.



In recent years, QR codes have become an everyday part of our lives. They are a two-dimensional barcodes that can be easily read by a digital device and that store information as a series of pixels. They have become widely used in trade, logistics, and production.

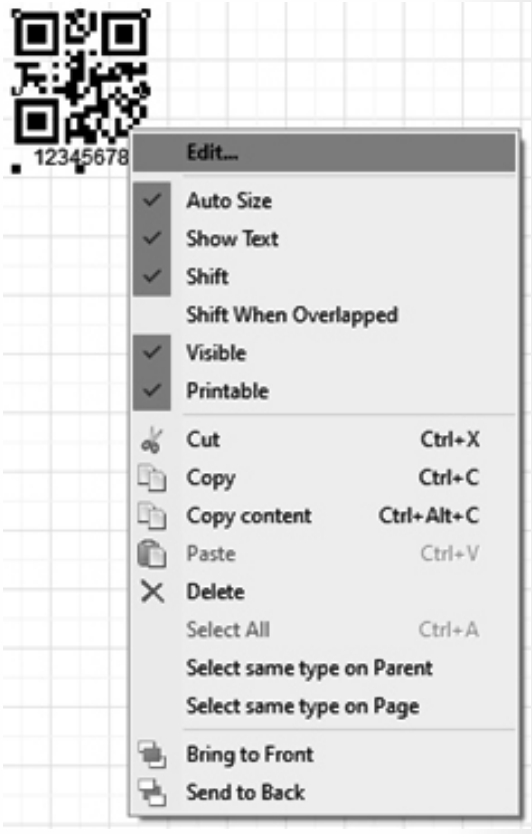
Unlike a simple barcode, a QR code is read horizontally and vertically. Thus, they store more data. The ease of recognition and ease of use of QR codes predetermined their popularity. With FastReport VCL you can easily use QR codes in your reports. Let's look at this possibility more closely.

Launch the report generator designer. Select the "QRCode" component from the pop-up menu of the "Barcode object" tool.

Selecting the "QRCode" component

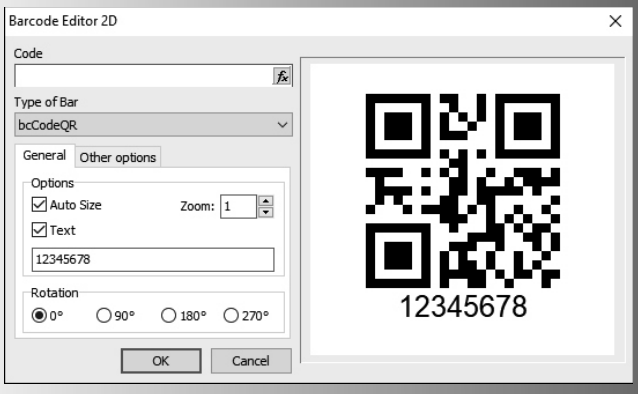
When added to a page, the QRCode will look like this:

Double click on it to open the 2D barcode editor.
This can also be done from the context menu by selecting the "Edit" item:



Barcode editing

This is what the barcode editor looks like:



2D barcode editor

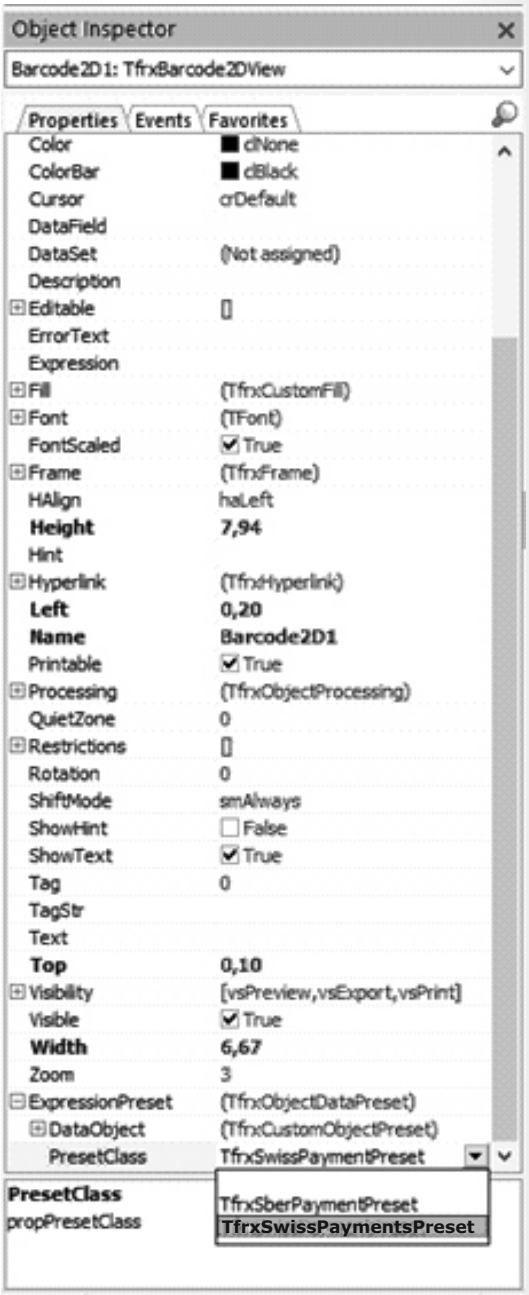
Let's look at its main functions in detail:

❶ In the "General" tab, you can change the size of the barcode, add a signature or rotate it.

❷ On the "Other options" tab:
- If CodePage <> 0 - ECI mode is used;
- You can select the text encoding in the Encoding property;
- The ErrorLevels property ensures redundancy for proper reading of data with a partially corrupted code image;
- QuietZone detects if the QR code has a white frame.

❸ In the expression editor, on the "Code" line, you can:
- Access data source fields;
- Use system variables;
- Use various functions

In the "ExpressionPreset" object property, you can select presets for generating receipts according to the specifications of the SBER code and Swiss code:



Selecting presets for the QR code

You can generate your own QR codes of various types, for this you need to specify a string of a certain format and set it in the Text property.

Let's take a closer look at these types with examples of barcodes and data:

URI is a Uniform Resource Identifier. It is a string to identify various files, documents, images, email, web service, etc.:

`https://www.fast-report.com/en/`

`https://www.fastreport.ru/ru/`

- EmailAddress – E-mail address:

support@fast-report.com

- SMS –text message:

SMSTO:(71) 555-4444:Hello, Dolly! I'm fine!

- EmailMessage –email text:

MATMSG:TO:support@fast-report.com;
SUB:FastReport VCL question;B
ODY:Hello, I have a question about FastReport VCL.;

- Geolocation – coordinates for the real geographic location:

geo:-50.737563,-79.490016,120

- Call – telephone number:

tel:(71) 555-4444

- Wi -fi – information for connecting to wi-fi:

WIFI:T:WPA;S:Honeypot;P:youarewelcome;H:true;

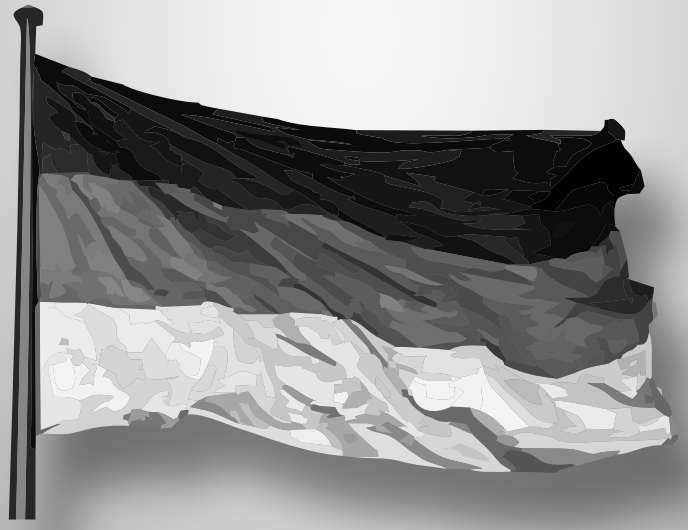
We have looked at how to use QR codes in FastReport VCL.
This component brings more possibilities for using this report generator in modern workflow.

Let's create a QR code from Delphi/Lazarus code:

```
uses frxBarcode2D;
procedure TForm1.Button1Click(Sender: TObject);
var
 bcQR: TfrxBarcode2DView;
begin
 bcQR := TfrxBarcode2DView(frxReport1.FindObject('Barcode2D1'));
 { Set the barcode type }
 bcQR.BarType := bcCodeQR;

 { Depending on the type you want to use, you need to
   leave one line that assigns the text of a certain format }

 { If you want to set url type: }
 bcQR.Text := 'https://www.fast-report.com/en/';

 { If you want to set EmailAddress type: }
 bcQR.Text := 'support@fast-report.com';

 { If you want to set EmailMessage type: }
 bcQR.Text :=
  'MATMSG:TO:support@fast-report.com;SUB:FastReport.Net question;BODY:Hello, I have a question about FastReport VCL.;';

 { If you want to set Geolocation type: }
 bcQR.Text := 'geo:-50.737563,-79.490016,120';

 { If you want to set SMS type: }
 bcQR.Text := 'SMSTO:(71) 555-4444:Hello, Dolly! I" m fine ! ';

 { If you want to set Call type: }
 bcQR.Text := 'tel:(71) 555-4444';

 { If you want to set Wi-Fi type: }
 bcQR.Text := 'WIFI:T:WPA;S:Honeypot;P:youarewelcome;H:true;';
 frxReport1.ShowReport();
end;
```

BLAISE PASCAL MAGAZINE

Wir suchen Hilfe bei der Korrektur von bereits
ins Deutsche übersetzte Artikel,
um auf idiomatische Ungenauigkeiten
und (Formatierungs-)Fehler
in den Artikel hinzuweisen.



Wenn Sie uns helfen möchten,
senden Sie bitte Ihre Reaktion an
admin@blaisepascalmagazine.eu

TRANSLATING YOUR PAS2JS WEB APPLICATION

By Michaël Van Canneyt

starter expert

ABSTRACT

Webapplications and websites are often visible to the world, and allowing the user to use your application or website in his/her own language is clearly improvement of the user interface. Here we show how you can translate your **PAS2JS** application.

1 INTRODUCTION

If you have a public website or public web application, it is visible to the whole world.
Users with any mother tongue will be able to access it, and of course it is more pleasant for them if they can use your site in their familiar language, rather than the lingua franca on the internet: **English.**
When translating a web application, there are 2 parts to consider when you want the user to be able to use your site:

❶ The messages generated from code.
❷ The texts that are inserted in the **HTML.**

Both kinds of translation require a dierent approach.

2 TRANSLATING MESSAGES IN CODE

Delphi and **Free Pascal** have long supported the concept of resource strings:
These are named string constants in code, which work almost like real constants.
The difference in code is that they are in a 'resourcestring' section in the code instead of a 'const' section.
The difference at runtime is that the value of a resource string is fetched at runtime, based on its name. The mechanism to get the value of the string at runtime is pluggable in **FPC.**
You can get the strings from a **DLL** (*the default mechanism in* **Delphi),** or a `.po` file (*common on* **unix)** or invent your own mechanism.
By switching the source of the resource strings, you can provide a translated version of the strings.

In **PAS2JS,** the concept of a resource string is kept.
The generated **Javascript** contains the default version of the resourcestrings, as defined in the source code.
However, value of the strings are kept in a structured manner, and fetched using a special helper function based on the name of the resource string constant.
A method is provided to set the values in the **javascript** sources at runtime:
The **PAS2JS RTL** comes with a unit that allows you to set the values of the resource strings in the program.
Just as the **FPC** compiler does, the **PAS2JS** compiler can generate a **JSON** file with the default resource string denitions. There is a command-line option that controls the generation of this file:

- `-Jrnone` No file is generated. This is the default.
- `-Jrunit` The compiler generates a **JSON** file per used unit.
- `-Jrprogram` The compiler generates a single **JSON** file with all the used resource strings in the program.

The **JSON** file has extension `jrs`.

The file generated with `-Jrprogram` looks like this:

```
{
   "mystrings" : {
      "Header" : "Translation using resource strings",
      "Paragraph" : "This text will be translated.",
      "TranslateDirect" : "The direct API is used for this example.",
      "TranslateJSON" : "A JSON object is used for this example.",
      "TranslateURL" : "A URL is used for this example.",
      "Button" : "Translate this page"
},
   "SysUtils" : {
      "SAbortError" : "Operation aborted",
      "SApplicationException" : "Application raised an exception: ",
      „SErrUnknownExceptionType" : "Caught unknown exception type : "
},
   "translate_basic" : {
      "BasicTitle" : "Translation using resource strings - Basic API"
   }
}
```

As you can see, the strings are grouped in a **JSON** object per unit, and there is a special section used for the program.
When compiling for a single unit, the following file would be generated for the mystrings unit:

```
{
  "mystrings" : {
     "Header" : "Translation using resource strings",
     "Paragraph" : "This text will be translated.",
     "TranslateDirect" : "The direct API is used for this example.",
     "TranslateJSON" : "A JSON object is used for this example.",
     "TranslateURL" : "A URL is used for this example.",
     "Button" : "Translate this page"
  }
}
```

This file can be used to translate the strings in an easy manner using any external tool you want. The **JSON** format lends itself very well to manipulation using various tools, and merging is easy. To actually translate the strings, the **PAS2JS** RTL contains a unit `rstranslate`, which can be used to translate the strings at runtime.
The unit contains the following denitions:

```
Type

{ TResourceTranslator }
TLoadFailEvent = Reference to Procedure (Sender : TObject; aCode : Integer; aError : String);
TOnTranslatedEvent = Reference to Procedure (Sender : TObject; aURL : String);

TResourceTranslator = Class
   Class Function Instance : TResourceTranslator;
   Procedure Translate(Const aUnit,aString,aTranslation : String);
   Procedure Translate(Const aTranslations : TJSOBject);
   procedure Translate(const aURL: string; aOnTranslated : TOnTranslatedEvent = Nil);
   Procedure ResetTranslation(Const aUnit : String;  const aString : String = '');
   Property  OnLoadFail : TLoadFailEvent;
   Property OnURLTranslated : TOnTranslatedEvent;
end;

Function ResourceTranslator : TResourceTranslator;

Procedure Translate(Const aURL : String; aOnTranslated : TOnTranslatedEvent = Nil);
Procedure Translate(Const aUnit,aString,aTranslation : String);
Procedure Translate(Const aTranslations : TJSOBject);

Procedure ResetTranslation(Const aUnit : String; Const aString : String = '');
```

The usage of this unit is quite straightforward:
The `TResourceTranslator` is the class that does the actual work of translating the resourcestring structures generated by the compiler. The unit creates a global instance (available in `TResourecTranslator.Instance`, with the shortcut function `ResourceTranslator`), so you don't need to create an instance.
The `Translate` functions do the actual translation. The first form sets the value of a single resource string:

```
Procedure Translate(Const aUnit,aString,aTranslation : String); overload;
```

The `aUnit` and `aString` parameters identify the resource string, and `aTranslation` is the value to set for the resource string.
The second form accepts a **JSON** object in the same form as produced by the compiler:

```
Procedure Translate(Const aTranslations : TJSOBject); overload;
```

It will iterate over all strings in the **JSON** object, and set the translation for every string.
The last form loads a **JSON** file from a **URL,** converts the file to a **JSON** object, and calls the previous

```
procedure Translate(const aURL: string; aOnTranslated : TOnTranslatedEvent = Nil); overload;
```

Since loading a file from a **URL** is asynchrone, there is a callback: when done, the `aOnTranslated` event is called. The objects event `OnURLTranslated` is also called.
If loading of the **JSON** file fails, the `OnLoadFail` event is called.
The translation mechanism used by **PAS2JS** does not destroy the original values of the resource strings. They can be reset to their original values if so desired.
This can be done with the `ResetTranslation` call. This call can be use to reset the translation for a single unit or for a single resource string.
The translation of the resource strings can of course be done at any moment: as of that moment, using a resource string will from then on use the translated value.
This makes the mechanism suitable for the situation where a dropdown is presented to let the the user select his preferred language.
To demonstrate, let's look at the following small webpage, which we will save in a file called `index.html`:

```
<!doctype html>
 <html lang="en">
 <head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bulma.min.css">
  <title id="translate-title"></title>
  <script src="translate.js"></script>
 </head>

 <body>
  <div class="section">
   <div class="box">
    <h2 class="title is-2" id="translate-header"></h2>
    <p class="block"id="translate-text"></p>
    <button id="btn-translate" class="button is-link"></button>
   </div>
  </div>
  <script>
   window.onload=rtl.run;
  </script>
 </body>
</html>
```

You can see that there are no texts in this **HTML** page. The texts will be entered in the tags by the program.
Here is part of the program. It has a `mystrings` unit which contains the most of the resource strings. The `PageTitle` resource string is in the program, in order to demonstrate the different sections in which the resource strings are located.
The program has some variables that correspond to the various **HTML** tags with an **ID** attribute, the variables are initialized in the `Init` routine:

```
program translation;
{$mode objfpc}
uses
  JS, Classes, SysUtils, Web, mystrings, rstranslate;

ResourceString
  PageTitle = 'Translation using resource strings - Manual';
Var
  aHeader : TJSHTMLELement;
  aPar : TJSHTMLELement;
  aButton : TJSHTMLButtonELement;
  IsDutch : Boolean;

procedure SetTexts;
begin
  aHeader.InnerHTML:=Header;
  aButton.InnerHTML:=Button;
  aPar.InnerHtml:=Paragraph+' '+TranslateDirect;
  Document.title:=PageTitle;
end;

Procedure Init;

  function GetEl(const aName: string): TJSHTMLELement;
  begin
    Result:=TJSHTMLELement(Document.getElementById(aName));
  end;

begin
  aHeader:=GetEl('translate-header');
  aPar:=GetEl('translate-text');
  aButton:=TJSHTMLButtonELement(GetEl('btn-translate'));
  aButton.onclick:=@DoTranslation;
end;

begin
  Init;
  SetTexts;
end.
```

The actual texts of the **HTML** page are set in the `SetTexts` routine: as you can see, the texts are in constant (resource)strings, located in the `MyStrings` unit.
The resulting page looks like *figure 1 on page 5 of this article.*
The `onclick` handler of the `aButton` button `(DoTranslation)` was omitted from the code for clarity. Clicking the button will translate the page from the default **(English)** to **Dutch** and vice versa.

The handler can be implemented as follows:

```
function DoTranslation(aEvent: TJSMouseEvent): boolean;
begin
  IsDutch:=Not IsDutch;
  if IsDutch then
    begin
      Translate('program','PageTitle','Vertaling met resourcestrings - directe API');
      Translate('mystrings','Button','Vertaal deze pagina');
      Translate('mystrings','Header','Vertaling met resourcestrings');
      Translate('mystrings','Paragraph','Deze tekst wordt vertaald.');
      Translate('mystrings','TranslateDirect','De directe API wordt gebruikt voor dit voorbeeld.');
      Translate('mystrings','TranslateJSON','Een JSON object wordt gebruikt voor dit voorbeeld.');
      Translate('mystrings','TranslateURL','Een URL wordt gebruikt voor dit voorbeeld.');
    end
  else
    begin
      // Single string of a module
      ResetTranslation('program','PageTitle');
      // All strings in a module
      ResetTranslation('mystrings');
    end;
  SetTexts;
end;
```

Figure 1: The page in its default language

As you can see, all the strings are translated manually to dutch by directly setting the dutch value for the resourcestring using the `translate` call. To go back to English, the `ResetTranslation` routine is called. At the end, the `SetText` routine is called, which simply re-applies the resource strings.
The page will then look like *figure 2 on page 6 of the article.*
Obviously, the above is not a recommended method to translate your program:
If you need to support lots of languages and have lots of strings to translate, the amount of code needed to do so would become really big. Rather, it is better to load the translations in some structured format **(JSON)** and have a small routine that translates the strings. The `rstranslate` routine unit contains a routine which accepts a **JSON** object with the same format as the le that the compiler creates, and uses that to translate all strings.

One way to do this is to include the translation in the **HTML** page.

For this, we create a small **Javascript** file that defines the **JSON** object with the translations:

```
var dutch = {
  "program" : {
     "URLTitle":"Vertaling met resourcestrings - URL API"
  },
  "mystrings": {
     "Button":"Vertaal deze pagina",
     "Header":"Vertaling met resourcestrings",
     "Paragraph":"Deze tekst wordt vertaald.",
     "TranslateDirect":"De directe API wordt gebruikt voor dit voorbeeld.",
     "TranslateJSON":"Een JSON object wordt gebruikt voor dit voorbeeld.“
     "TranslateURL":"Een URL wordt gebruikt voor dit voorbeeld.“
     }
  };
```

Figure 2: The page in dutch

We include this in the **HTML** file by adding the following line:

```
<script src="dutch.js"></script>
```

and to be able to access the dutch variable in our program, we define an external variable in our pascal program as follows:

```
Var
  Dutch : TJSObject; external name 'dutch';
```

The value of the **Dutch** variable is then the **JSON** object with the translations. The `DoTranslate` routine now becomes simply the following:

```
function DoTranslation(aEvent: TJSMouseEvent): boolean;

begin
  Result:=True;
  IsDutch:=Not IsDutch;
  if IsDutch then
    Translate(Dutch)
else
  begin
    ResetTranslation('program','PageTitle');
    ResetTranslation('mystrings');
    SetTexts;
  end;
end;
```

By defining variables for different languages, one can switch between different languages.
Of course, at the startup of the program, all languages will be loaded.
This can be avoided by dynamically injecting the script tag in the **HTML** as soon as the language needs to be switched: the browser will then load the new translations.

By giving all translations the same name for the variable that holds the **JSON** object, only the last injected language will be kept in memory in the browser.
A disadvantage of this method is still that you need an extra **Javascript** source file.
This necessitates the conversion of a **JSON** file as produced by the compiler to a **Javascript** source file.
It would be easier if the **JSON** file could be used directly.
Fortunately, this is possible.

The `Translate` call has an overloaded version which can be passed an **URL:** this **URL** should point to a **JSON** file with the translations of all strings, in the format that the compiler provides.
The call will download this file and use the translations to translate the strings found in the **JSON f**ile.
There is callback that can be used to be notified when the translation is complete
(*downloading a file is asynchronous*), this callback can be used to refresh the display.

The third version of our program shows how this can be used.
Adapting our program consists of 3 steps:

❶ Remove the script tag with the reference to the `dutch.js` file.
❷ Remove the external variable definition `Dutch`
❸ Change the `DoTranslation` routine to use the **URL.**

The `DoTranslation` routine will now look as follows:

```
function DoTranslation(aEvent: TJSMouseEvent): boolean;

  Procedure DoTranslated(Sender : TObject; aURL : String);
  begin
    SetTexts;
  end;

begin
  Result:=True;
  IsDutch:=Not IsDutch;
  if IsDutch then
    Translate('dutch.json',@DoTranslated)
  else
    begin
      ResetTranslation('program','PageTitle');
      ResetTranslation('mystrings');
      SetTexts;
    end;
end;
```

NOTE that the `SetTexts` must be called from the `OnTranslated` callback when loading the **JSON** file (*because only then will the texts actually be translated*), but can be called directly when resetting the translations.

3 TRANSLATING HTML CONTENT

In practice, you will probably not fill up your **HTML** page with texts from resource strings. One disadvantage is that every location that needs to be translated needs an **ID** attribute. Second disadvantage is that the **Javascript** must be loaded before the text appears in your **HTML** page. If formatting is applied, then the text will be cut into different bits, which is unnatural.

Instead, your **HTML** will most likely contain the texts in the default language when the page is loaded.
So, how can we translate the texts in the **HTML?** There are many possibilities.
We could use resource strings and by assigning an **ID** to all html tags we can translate all texts. This has much of the disadvantages of the previous method.
We can make a **HTML** page for every language; This is a long and cumbersome process.

Here we'll present an alternate approach, implemented in the `Rtl.HTMLTranslate` unit.
The first problem is that we need to know what texts we need to translate.
We can simply scan the **HTML** text and extract all text fragments: this is simple enough.
Then we need to find the translation. Using a hash mechanism, we can quickly search for translations in a file with translations.
This has the disadvantage that all context is lost, risking to simply translate wrongly in the case of single words. For example, the english word `'Save'` can be translated in several ways to **Dutch.** Which translation to use depends on the context, and cannot be determined from the hash alone.
A better approach is to give a unique identier to all texts that needs to be used, so the context of the word or sentence can be looked up. But preferably not the **ID,** to avoid clashes in case various **HTML** fragments are combined into a single page.
We can use the data attribute for this, it is treated specially in the **Javascript DOM API.**
Practically, this means that all tags that need translation must be marked as follows:

```
<h2 data-translate="Header">
Translation using HTMLTranslator
</h2
```

The `data-translate` attribute is set to header `Header`. The translation routine will look for a text named `Header` and when found, will replace the elements `InnerText` or `InnerHTML` with the translated text.

Sometimes, one needs to translate not the inner text, but an attribute of the **HTML** tag. A prime example is the placeholder attribute of the `Input` and `TextArea` tags.

For this, the `Rtl.HTMLTranslate` unit uses a convention: the name of attribute is part of the translate name: everything after the dash is considered an attribute name.

```
<textarea data-translate="memo-placeholder"
          placeholder="Type your remark here">
  Translation using HTMLTranslator
</textarea>
```

When the translation engine encounters this, it will look for a translation named 'memo-placeholder' in the translated texts, and will place whatever it finds in the placeholder attribute. This convention of course means that you cannot use dashes in your translation names.
The `TextArea` can potentially have 2 texts to translate: the placeholder attribute and the actual text in the text area. Depending on how **HTML** is used, more attributes may need translation. Since we have only a single data attribute, how can we cater for this? Simple: the data-translate mechanism can handle multiple names, separated by a semicolon.

```
<textarea data-translate="memo;memo-placeholder"
placeholder="Type your remark here">
Translation using HTMLTranslator
</textarea>
```

When handling this tag, the translation engine will replace the inner text with the contents of the named translation memo, and will replace the placeholder with the content of memo-placeholder.

Many **Javascript APIs** use data- tags to configure their rendering (*for example grid* **APIs** *commonly use this to configure columns*), and the above mechanism can be used to translate these attributes as well. The **API** of the `Rtl.HTMLTranslate` unit is quite simple:

```
THTMLTagTranslator = class(TComponent)
Public
  procedure SetLanguageData(aData : TJSObject);
  Procedure TranslateHTMLTag(aEl : TJSHTMLElement; aScope : String = ''); overload;
  Procedure TranslateBelowHTMLTag(aEl : TJSHTMLElement; aScope : String = ''); overload;
  function GetMessageByName(const aScope, aName: string): String; overload;
  Function GetMessageByName(aRoot : TJSObject; const aScope,aName: string): String; overload;
  Function GetMessageByName(aScope : TJSObject; aName : string) : String; overload;
  Function GetMessageByName(aScope,aSeealsoScope : TJSObject; aName : string) : String; overload;
  Function HasLanguage(aLanguage : String) : Boolean;
  Function GetScope(const aScope : String) : TJSObject; overload;
  Function GetScope(aRoot : TJSObject; const aScope : String) : TJSObject; overload;
  Property CurrLanguageStrings : TJSObject Read FCurrLanguageStrings;

Published
  Property DataTagName : String Read GetDataTagName Write FDataTagName;
  Property DefaultScope : String Read FDefaultScopeName Write FDefaultScopeName;
  Property ContinueKey : String Read FContinueKey Write FContinueKey;
  Property Language : String Read FLanguage Write SetLanguage;
  Property LanguageSource : TLanguageSource Read FFileMode Write FFileMode;
  Property LanguageFileURL : String Read FLanguageFileURL Write FLanguageFileURL;
  Property LanguageVarName: String read FLanguageVarName write FLanguageVarName;
  Property TextMode : TTextMode Read FTextMode Write FTextMode;
  Property OnLanguageLoaded : TLanguageLoadedEvent;
  Property OnLanguageLoadError : TLanguageLoadErrorEvent;
end;
```

The class has the following public/published **properties:**

CurrLanguageStrings
the data for the current language.

DataTagName
Data attribute name. You can change this to use another data attribute. Default is 'translate'.

DefaultScope
Default scope to use when looking for translations. First the scope passed in `TranslateHTMLTag` is checked, then the default scope, then `ContinueKey` is used.

ContinueKey
Continue key name: When set, indicates the name of a scope in which to continue searching for a translation term. This can be used for form inheritance; `ContinueKey` can be set to continue searching in the inherited scope. By default it is empty.

Language
Current Language name (*will be lowercased*)

LanguageSource
Language source: All languages in a single file `(lsSingle)`, languages reside in multiple files `(lsMulti)` or a variable in the **Javascript** global Scope `(lsScoped)` contains all languages.

LanguageFileURL
The URL to the file to load language strings from. If `LanguageSource=lsMulti` this should be a format template in which the language code is substituted (use for Example: `/lang-%s.json`).

LanguageVarName
if `LanguageSource=lsScoped` then `LanguageVarName` is the name of the global `Window` property that contains the translations.

TextMode
Textmode determines whether the translator uses `InnerText (tmText)` or `InnerHTML (tmHTML)` to set the translated text. **NOTE** that using `tmHTML` can result in a security leak as **Javascript** could be injected.

OnLanguageLoaded
An event triggered when a language is loaded (*whatever the mechanism*).

OnLanguageLoadError
An event called when an error occurs during loading of a language file.

The class has the following public **methods:**

SetLanguageData
Directly set the language data object to `aData` instead of using an **URL.** The data will be interpreted according to `LanguageSource`.

TranslateHTMLTag
Translate a single **HTML** tag `aEl`, using indicated scope `aScope`. If the scope is not given, the root scope of the current language is used or the default scope.

TranslateBelowHTMLTag
Translate the **HTML** tag `aEl` and everything below it, using indicated scope `aScope`. If the scope is not given, the root scope of the current language is used.

GetMessageByName
Search for a message named `aName` in the indicated `aScope`, optionally `aSeeAlsoScope`. In case the scope name is given you can start the search for the scope in `aRoot` The message is empty if no message was found.

HasLanguage
check if the given language is available. In case of `lsMulti`, the current language must match the passed `aLanguage`.

GetScope
find the scope object named aScope. if `aRoot` is given, start the search at `aRoot`.

NOTE that the notion of 'scopes' is used. This is done for use in **SPA** applications. Each "page" in the **SPA** can use its own scope when looking for translations: this means that there is no need to invent globally unique names for the texts: names only need to be unique in a scope.

So, how to use this **API** to translate our application? First of all, our **HTML** must be extended with the data-translate attribute:

```
<!doctype html>
<html lang="en">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="bulma.min.css">
<title data-translate="Title">Translation using resource strings - Manual</title>
<script src="translate.js"></script>
  </head>

  <body>
    <div class="section">
      <div class="box">
        <h2 class="title is-2"
            data-translate="Header">
            Translation using HTMLTranslator
        </h2>
      <p class="block"
         data-translate="Paragraph">
         This text will be translated using data-translate.
      </p>
      <div class="field is-horizontal">
        <div class="field-label is-normal">
          <label class="label"
              data-translate="Name">
            Name
          </label>
        </div>
          <div class="field-body">
            <div class="field">
              <p class="control">
                <input class="input"
                       type="text"
                       placeholder="Type your name"
                       data-translate="Input-placeholder">
                </p>
              </div>
            </div>
          </div>
        <button
          id="btn-translate"
          class="button is-link"
          data-translate="Button">
          Translate this page
        </button>
        </div>
      </div>
      <script>
         window.onload=rtl.run;
      </script>
  </body>
</html>
```

Secondly, we need to create a file with the translations.
NOTE that, contrary to the resource string mechanism, all translations must be present, also the original language: there is no way to reset the translation.
The file looks as follows:

```
{
  "nl" : {
    "html" : {
      "Title":"Vertaling met HTMLTranslator",
      "Button":"Vertaal deze pagina",
      "Header":"Vertaling met resourcestrings",
      "Paragraph":"Deze tekst wordt vertaald.",
      "Input-placeholder":"Typ uw naam",
      "Name":"Naam"
    }
  },
  "en" : {
    "html" : {
      "Title" : "Translation using HTMLTranslator",
      "Header" : "Translation using HTMLTranslator",
      "Button": "Translate this page",
      "Paragraph" : "This text will be translated using data-translate",
      "Input-placeholder":"Type your name",
      "Name":"Name"
    }
  }
}
```

As you can see, we've elected to use the scope "html". We save this file as `lang.json`, next to our `index.html` page.

Lastly, we adapt the code. The program starts by initializing an instance of `THTMLTagTranslator`. It sets the default scope to 'html' and set the `LanguageSource` to `lsMulti`, since we have multiple languages in our file.
The `LanguageFileURL` is set to `lang.json`, the name of the file we saved the translations in, it will be downloaded from the same location as the `index.html` page.

```
uses
  JS, Classes, SysUtils, Web, Rtl.HTMLTranslate;
Var
  aButton : TJSHTMLButtonELement;
  aTranslator : THTMLTagTranslator;
  IsDutch : Boolean;
  // some code removed for clarity...

Procedure Init;
begin
  aTranslator:=THTMLTagTranslator.Create(nil);
  aTranslator.LanguageSource:=lsMulti;
  aTranslator.LanguageFileURL:='lang.json';
  aTranslator.OnLanguageLoaded:=@DoLangLoaded;
  aTranslator.DefaultScope:='html';
  aButton:=TJSHTMLButtonELement(Document.GetElementByID('btn-translate'));
  aButton.onclick:=@DoTranslation;
end;

begin
  Init;
end.
```

As you can see, the code to translate is quite simple. As soon as the language is set, the `DoLangLoaded` is triggered, and the `TranslateBelowHTMLTag` can be used to do the actual translation.
Note that the code does not pass a scope to use to the `TranslateBelowHTMLTag` call: the default scope is set to `html`, and this is sufficient. Lastly, the `OnLanguageLoaded` is set to `DoLangLoaded`, where we do the actual translation:

```
procedure DoLangLoaded(Sender: TObject; aLanguage: String);
begin
  aTranslator.TranslateBelowHTMLTag(TJSHTMLElement(Document.body));
  aTranslator.TranslateBelowHTMLTag(TJSHTMLElement(Document.head));
end;
```

As you can see, we call the `TranslateBelowHTMLTag` method twice:

- once for the body element.
- once for the head element. This call will translate the title tag, thus setting the page title.

NOTE that we do not pass a scope name to the `TranslateBelowHTMLTag` call.
This will cause the code to use the default scope, 'html'.
The `onclick` handler of our button is now quite simple:

```
function DoTranslation(aEvent: TJSMouseEvent): boolean;
begin
  Result:=True;
  IsDutch:=Not IsDutch;

  if IsDutch
  then aTranslator.Language:='nl'
  else aTranslator.Language:='en'
end;
```

Setting the language will load the translation file if necessary, and when loaded, will call our `DoLangLoaded` event handler. That's all there is to it.

4 CREATING A FILE WITH TRANSLATIONS

It is not necessary to create the **JSON** file completely by hand: the **PAS2JS** repository contains a tool that allows you to extract all texts that must be translated from the **HTML** files. The tool is called **`extractlang`**, it can be found under the tools directory of the **PAS2JS** distribution.
Currently, only a command-line tool is available. When running it with the `-h` command-line option, it displays a help page:

```
Usage: /home/michael/P2JS/main/tools/extractlang/extractlang [options]
Where options is one or more of:
-h --help                  This help text
-c --clear                 Clear output JSON file (Default is to update existing file).
-d --html-dir=DIR          Directory with HTML files to scan
-f --file-mode=MODE        Set file mode: one of single or multiple
-o --output=FILE           File to write JSON translations (may get suffix depending on file mode)
-l --languages=LIST        Comma-separated list of languages to create
-m --minify                Minify output
-n --name=NAME             Set name of data-tag to NAME (data-NAME)
-r --recurse               Recurse into subdirectories of the HTML directory
-s --single-scope=SCOPE    Put all translation names in a single scope
-t --trash-values          Trash values for other languages
```

To operate, the tool needs at least 2 options: `-h` and `-o`.
The `-h` option specifies a directory where the tool will look for **HTML** files.
The `-o` option species the name of a **JSON** language file to create or update.

With these options, the tool will scan the directory for **HTML** files (a*nd subdirectories if -r is specied*), collect all **HTML** tags with attribute data-translate, and will write the inner text of this tag in a **JSON** file.

By default, only a english language will be created.
But you can also create the text for additional languages using the `-l` option: for each language, the tool will check if the named text is there, and if not, it will add the text.

If we run the program in the directory of our sample application using the following command-line:

```
extractlang -d . -o lang.json -l 'en,nl' -s html
```

We specify the `-s option` because that's the scope we used in code.
If no scope is specied, for each processed **HTML fi**le a separate scope will be used, where the scope name is the **HTML** filename, lowercased and without extension.
The tool will give you some diagnostic output:

```
Searching ./index.html for translatable terms, adding to scope : html
Found 6 translatable terms
Collected 1 message scopes
Copied 1 new scopes with 6 words, added 0 new words in existing scopes.
```

And we will end up with the following file:

```
{
  „en" : {
    "html" : {
      "Title" : "Translation using resource strings - Manual",
      "Header" : "Translation using HTMLTranslator",
      "Paragraph" : "This text will be translated using data-translate.",
      "Name" : "Name",
      "Input-placeholder" : "Type your name",
      "Button" : "Translate this page"
    }
  },
      "nl" : {
      "html" : {
      "Title" : "Translation using resource strings - Manual",
      "Header" : "Translation using HTMLTranslator",
      "Paragraph" : "This text will be translated using data-translate.",
      "Name" : "Name",
      "Input-placeholder" : "Type your name",
      "Button" : "Translate this page"
    }
  }
}
```

This file is ready to be translated. (*Note that if you use the minify option, the output is not human-readable and difficult to translate manually*)

By default, the tool will load an existing file and will simply add new texts, which makes it suitable for updating existing translations.

As an aid in translating, you can specify the -t (trash) option. In that case, the first language will contain the original text from the **HTML** file, and all new words in other languages will contain some chinese characters. Doing so can aid as a visual check when changing the language in the actual **HTML** page: any non-translated terms will appear as Chinese characters which (*for non Chinees people*) will stand out.

5 CONCLUSION

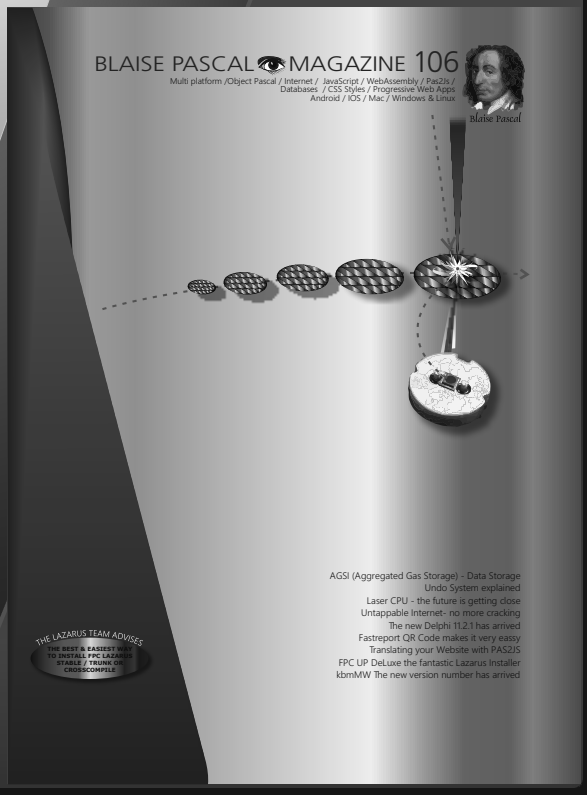
PAS2JS offers all tools needed to make a localized application. The mechanisms for translation of resource strings and **HTML** pages are similar in the sense that they use a **JSON** file: this allows you to have translations of both resourcestrings and **HTML** tags in a single **JSON** file. As always, there is some room for improvement, such as integration in the **Lazarus IDE,** a **GUI** program for the extractor tool and a tool to manage translations. These will be created in time. But all the basic mechanisms are there, so the impatient developer can already get started.

https://www.blaisepascalmagazine.eu/product-category/special-offer/

Lazarus Handbook Pocket + Subscription + PDF

- English
- Printed black & white
- 2 Volumes
- Sewn +2 ribbons
- PDF included
- 934 Pages
- Weight: 2kg
- Shipment excluded
- Extra protected
- Including 40 Example projects and extra programs

SPECIAL OFFER € 75



printed on FSC paper
https://fsc.org/en/forest-management-certification

Installation of Freepascal and Lazarus by using FPCUPdeluxe on Linux, Mac, Windows and other OS's.

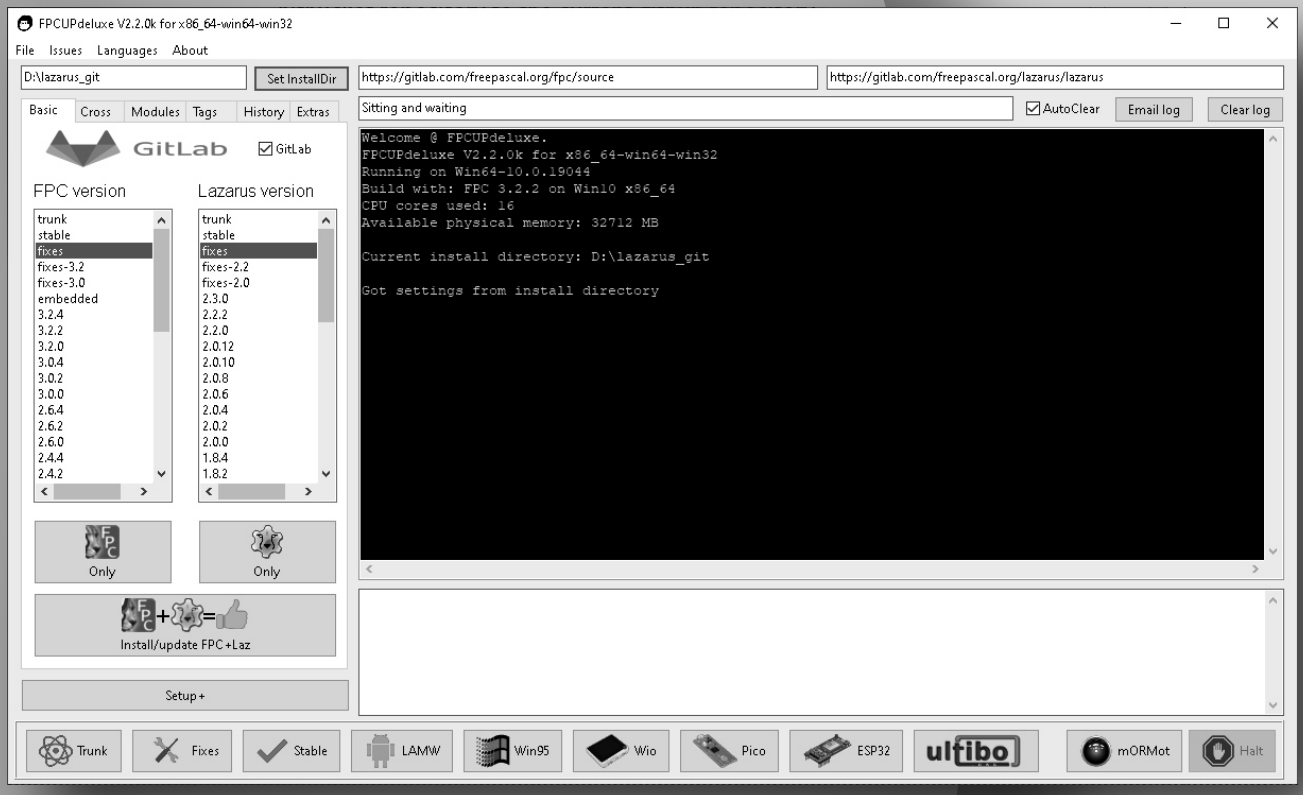
starter expert

ABSTRACT

By Jos Wegman

There are several ways to install FPC and Lazarus.
In this article I will describe a method where I use fpcupdeluxe.
By using this method you can choose between various versions of FPC and Lazarus. Including cross-compilers and/or modules.

Keywords: Fpcupdeluxe, FPC, Lazarus, installer, cross-compilers, modules.



1 INTRODUCTION

The program is free.
There are many ways to install **Lazarus** and / or **FPC.** For the popular platforms there are installers available for the stable versions on the website `https://www.lazaru-ide.org` in the download section.

In this article I want to present a different approach.
The application **FPCUPDELUXE,** maintained by the person with the github-name **"LongDirtyAnimAlf".**

The original project is started by Reinier Olieslagers.
Reinier sadly passed away on December 4, 2014.
At August 2015 **LongDirtyAnimAlf** (Alias Alfred Glänzer) pushed all the commits from Reiniers bitbucket repository to the current **github** repository.
From there on the program has been steadily improved and expanded. Here I will show the installation process to install a complete version of **FPC** and **Lazarus** compiled from sources using **FPCUPDELUXE.**

THE LAZARUS TEAM ADVISES

THE BEST & EASIEST WAY TO INSTALL FPC LAZARUS STABLE / TRUNK OR CROSSCOMPILE

2 DOWNLOAD AND PREREQUISITES

At the repository you can find a `READ.ME` file that gives the necessary information on prerequisites for installation.

- For **Windows** there are no prerequisites.
- For **Linux** there are several packages that have to be installed.
- All files of the installation are placeds in the same directory, so there is no need to search for anything except in that installation directory. This means all installations of the program are independant and you can create as many as you want giving each one a different directory name.

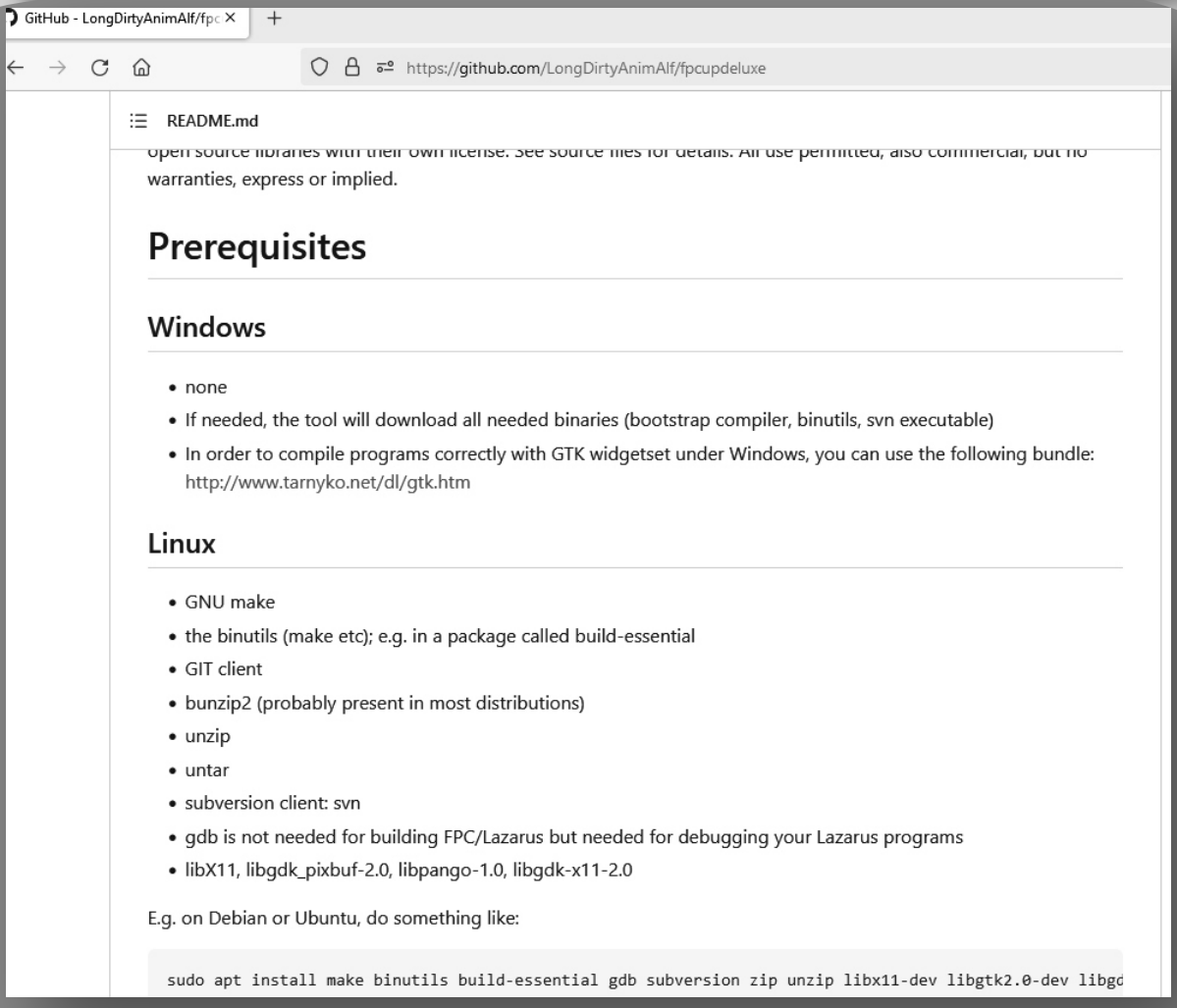


Figure 1: Prerequisites

These packages are mentioned in the `READ.ME` file.
I will perform the install under **Windows.** So you do **NOT** need extra prerequisites.
The program **FPCUPDELUXE** can be downloaded from the release site of the repository.

Links
https://github.com/LongDirtyAnimAlf/fpcupdeluxe
https://forum.lazarus.freepascal.org/index.php/topic,26726.0.html
https://github.com/LongDirtyAnimAlf/fpcupdeluxe/releases
http://www.batterybutcher.com:8880/root/getinfohtml

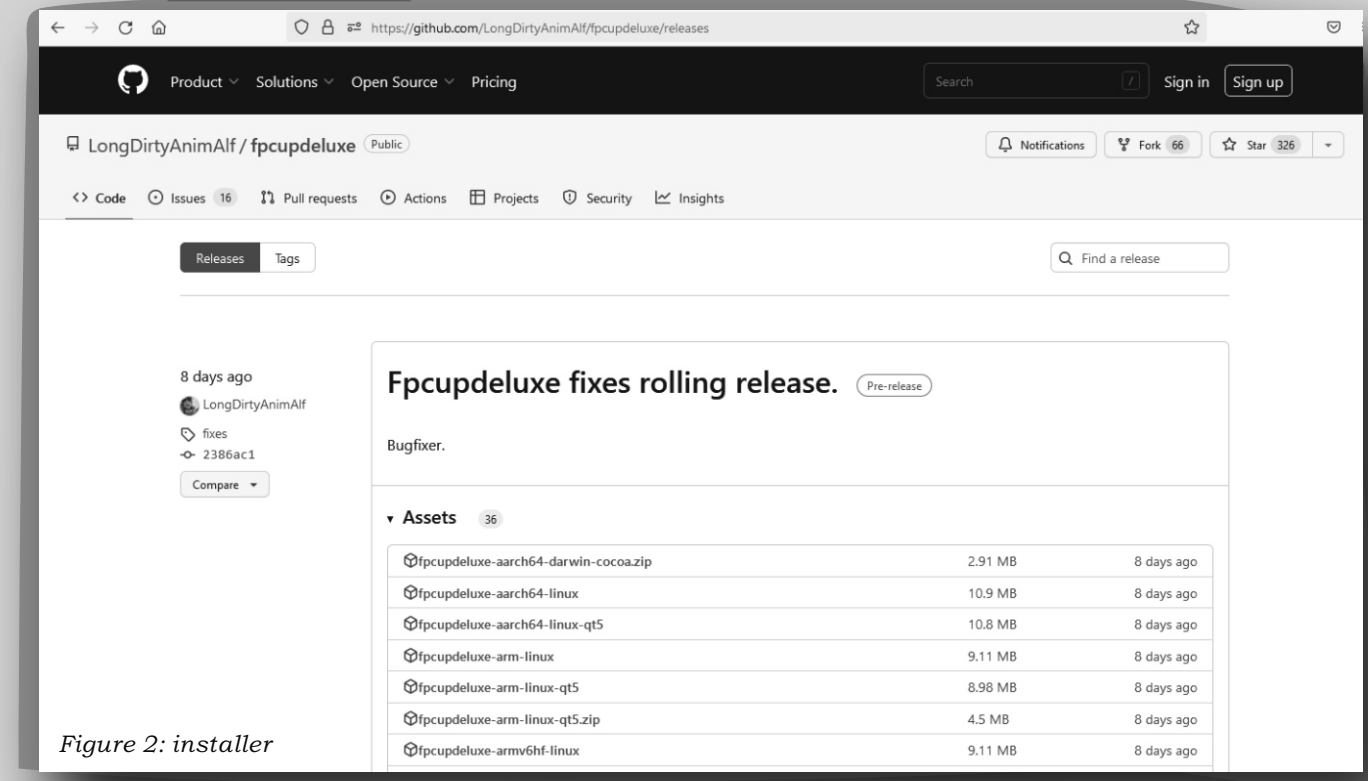
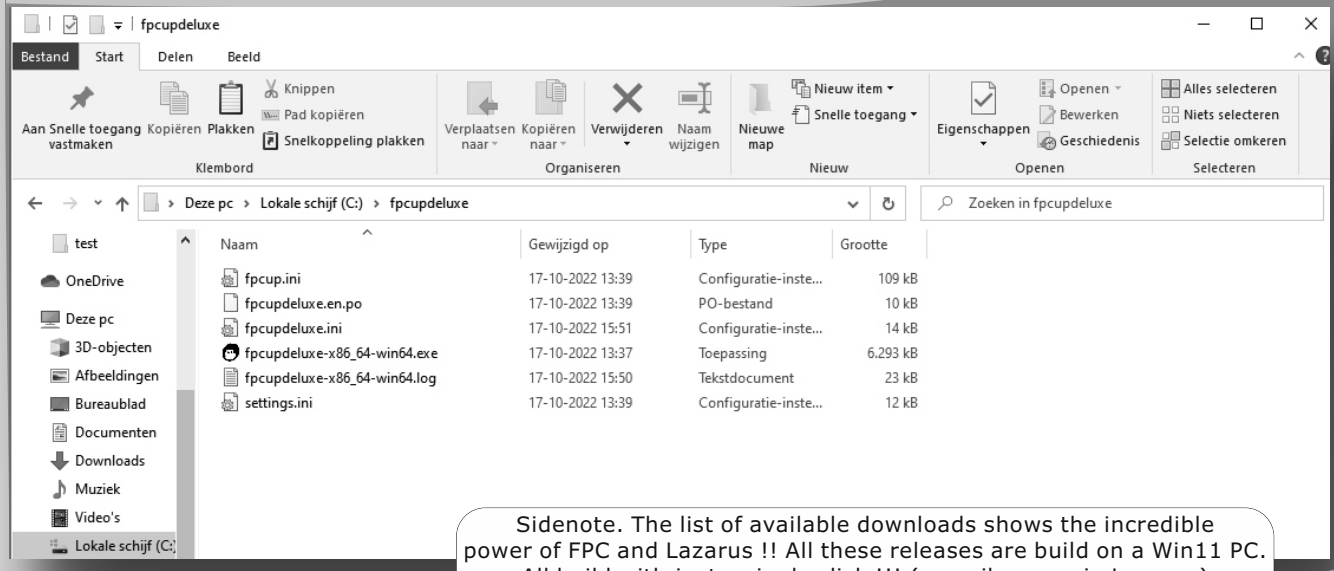


Figure 2: installer

Pick the application suited for your platform.
In my situation this is the `fpcupdeluxe-x86_64-win64.exe`.
For the sake of separation I created a directory on my system where I placed the downloaded program.
If **FPCUPDELUXE** is started for the first time it will create several config files needed by **FPCUPDELUXE.**
The config files are stored in the same directory as **FPCUPDELUXE.**

NOTE: As for your interest, FPC itself advices the 32bit version to be installed on **Windows**. This due to missing 80bit extended support on win64, that will hinder the install of (32bit) cross-compilers supporting this 80bit value. So, crossing from **win32** to **arm32** (RPi) works out of the box. Crossing from win64 to arm32 give a warning and causes the softfloat library to be activated [-dFPC_SOFT_FPUX80]. **Fpcupdeluxe** will detect this and give a warning.

Links
`http://www.jhauser.us/arithmetic/SoftFloat-3/doc/SoftFloat.html`



Sidenote. The list of available downloads shows the incredible power of FPC and Lazarus !! All these releases are build on a Win11 PC. All build with just a single click !!! (compile many in Lazarus)

3 SELECT OF INSTALL FEATURES

Start **FPCUPDELUXE.** On **Linux** the proper permissions must be set before **FPCUPDELUXE** can be started. On **Windows** just double-click the fpcupdeluxe icon.
If the program is started for the first time a pop-up will be displayed.

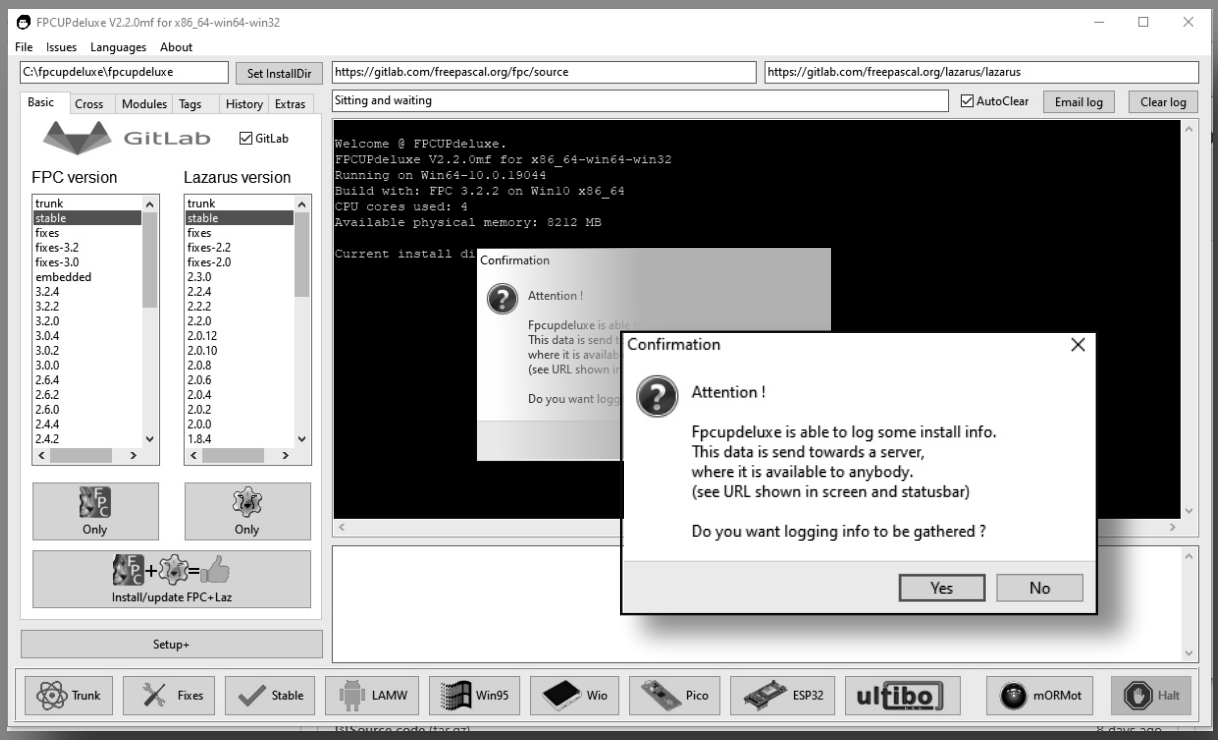


Figure 4: first run

You can choose either way, it will not affect your installation.
If you choose to share you're install data, you can find in the `READ.ME` file what data where is stored.

Before you can install **FPC** and **Lazarus** you have to set a few parameters.
First you have to decide in what directory you want to install **FPC** and **Lazarus.**
See the edit box in the upper left corner.
Use the "Set InstallDir" button to select the directory of choice.
The next step is to select the **FPC** version, in this situation "stable".
The last parameter is the **Lazarus** version, also "stable".

Now the application is set to install the requested versions of **FPC** and **Lazarus.**
To start the installation press the "Install/update FPC + Laz" button.
You will get a confirmation dialog. *(See figure 5 of page 5 of this article).*

NOTE: In the case of a Windows installation, it is advisable to choose "Win32" because this version automatically includes the compilation for "Win64", making it unnecessary to install it as well.

3 SELECT OF INSTALL FEATURES - CONTINUATION

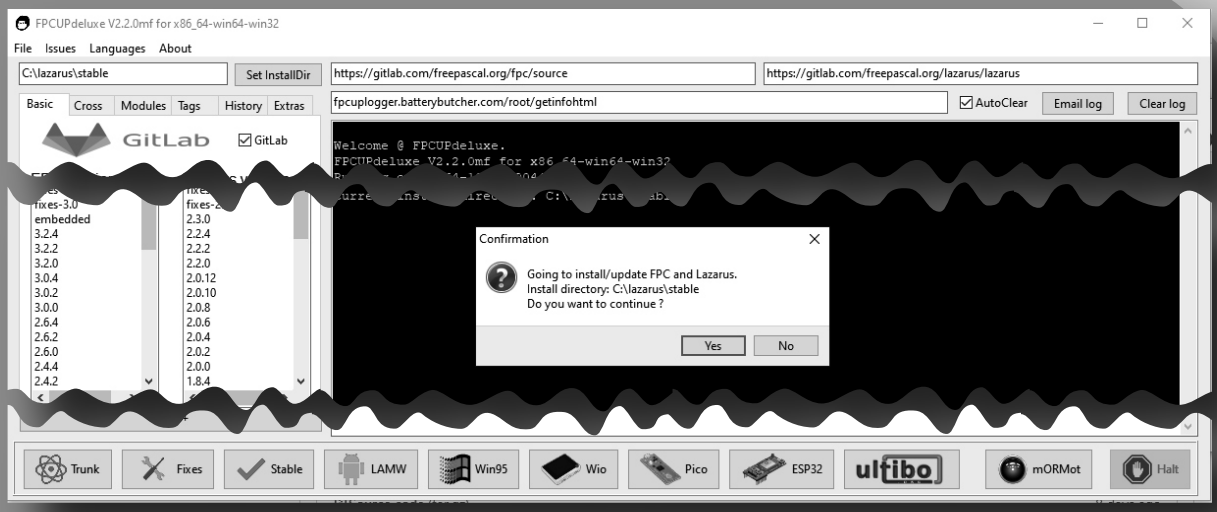


Figure 5: confirmation

If pressed "Yes", you can sit-back and relax. Wait for good things to happen.
The installation process will take place.
Progress- and action-messages are shown in the terminal window of **FPCUPDELUXE.**
An example of the messages are shown at figure 6.

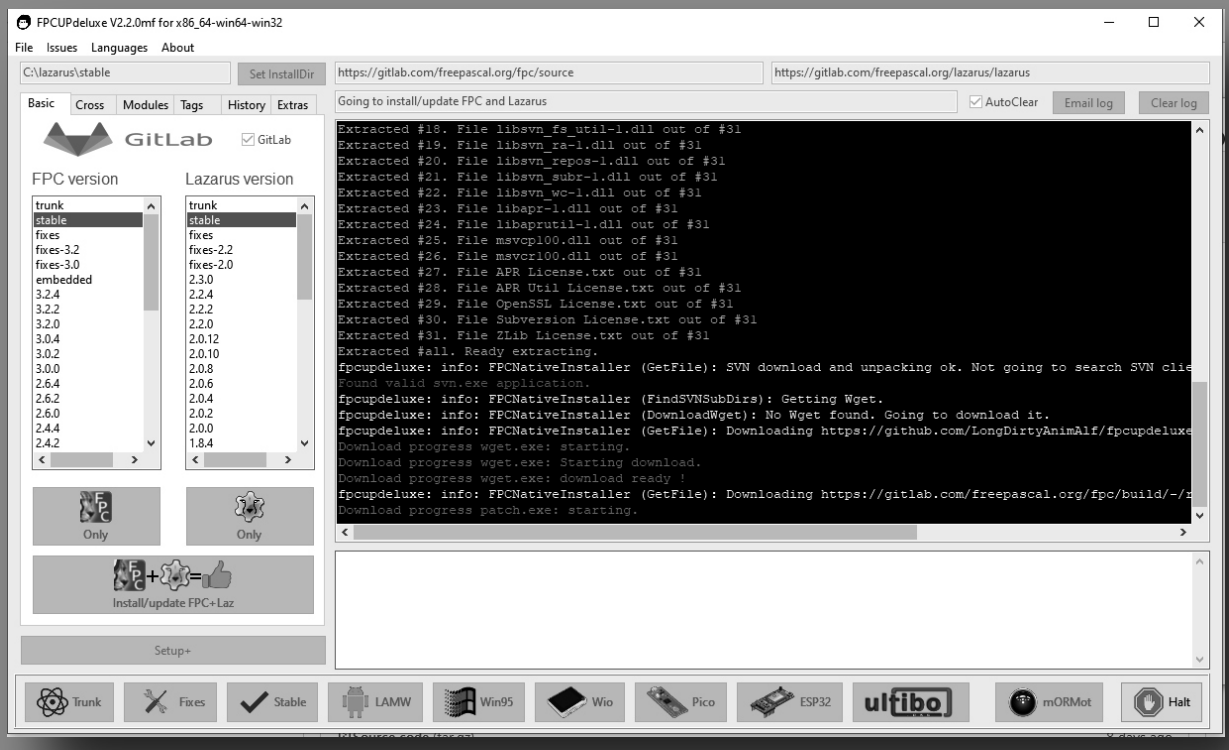


Figure 6: Comments .
if the installation was successful than you will see image 7. See next page

3 SELECT OF INSTALL FEATURES - CONTINUATION

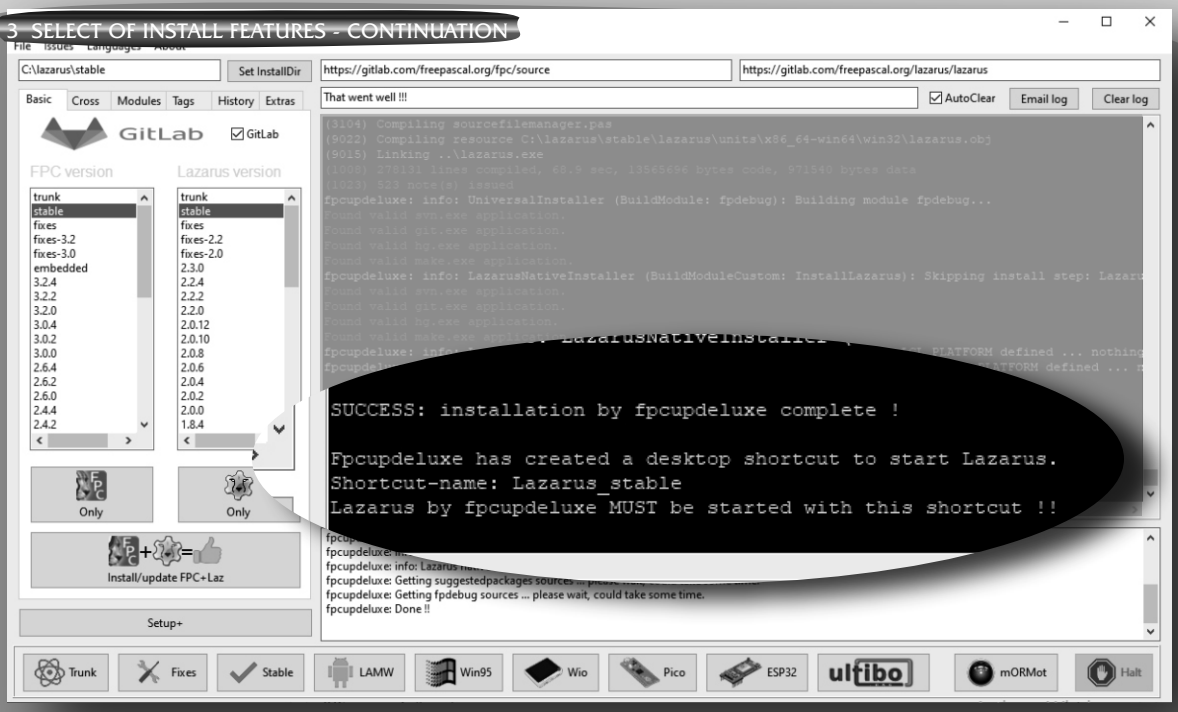


Figure 7: Success

Please pay attention to the last sentence in the terminal-window.
FPCUPDELUXE has created an icon on the desktop to start **Lazarus.**
This icon is **mandatory** to use for starting **Lazarus.**

4 INSTALLATION OF CROSS-COMPILER(S)

At this point **Lazarus** is ready to use as normal.
Now we will install a cross-compiler. **FPCUPDELUXE** is still running.

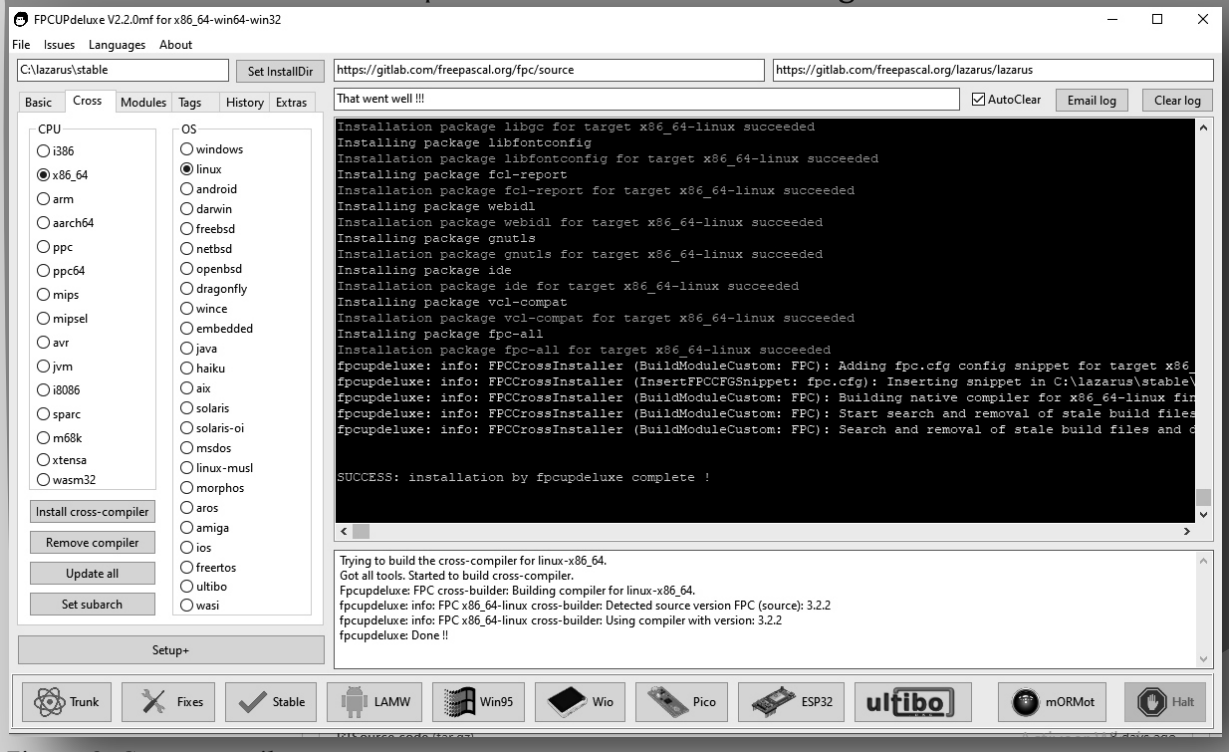


Figure 8: Cross-compiler

4 INSTALLATION OF CROSS-COMPILER(S) -CONTINUATION

To install a cross-compiler a few parameters must be set. First select the tab “cross”.
On this tab you can find the different **CPU** and **OS** possibilities. Choose the combination of the cross-compiler that you want to install. In this example `CPU = “x86_64”` and the `OS = “Linux”`.
This combination will install the cross-compiler for 64 bits **Linux** applications.
NOTE: During a first cross-install, a failure will be reported due to missing tools.
Allow **fpcupdeluxe** to download these tools and continue.
Press the button install cross-compiler and let fpcupdeluxe do the work.
After installation of the cross-compiler a success message is shown in the terminal window.

5 INSTALLATION OF MODULES

FPCUPDELUXE is capable to install several component packages. The so called modules.
First select the tab “modules”.

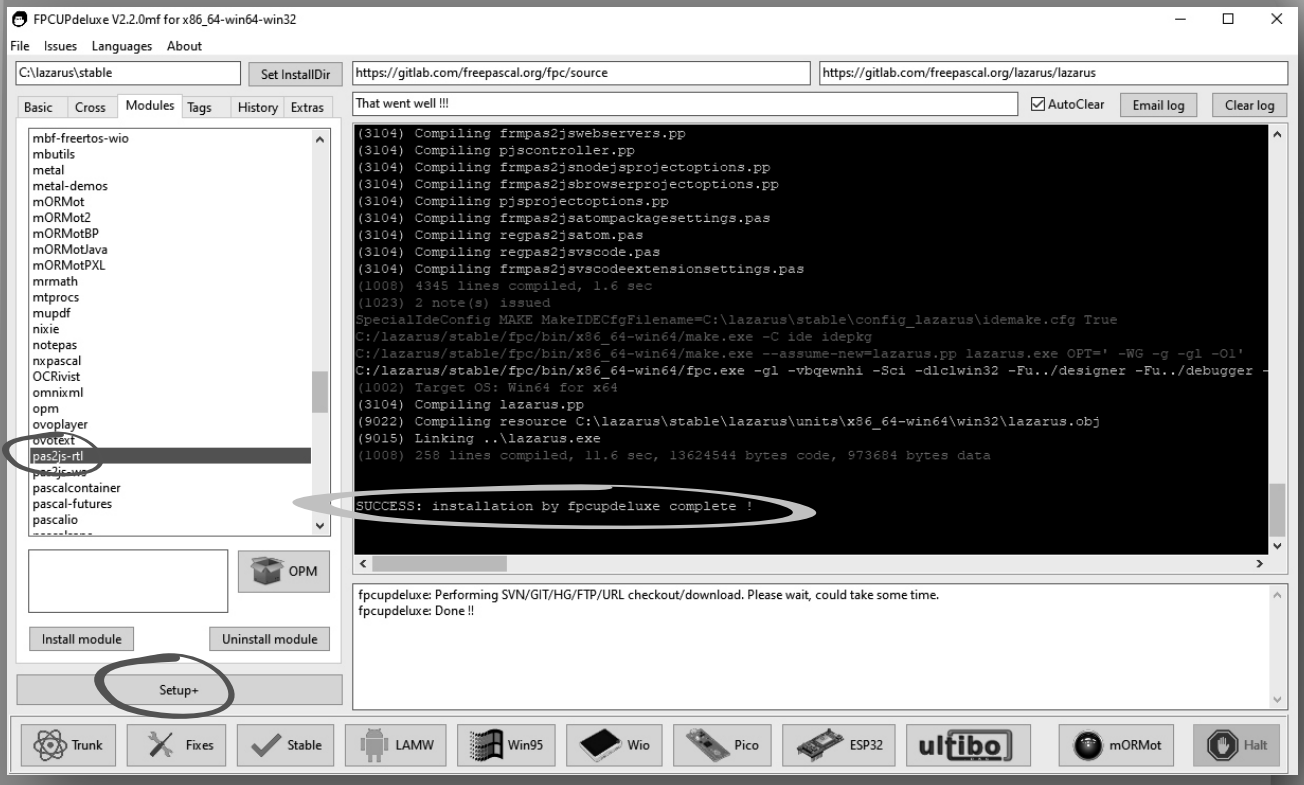


Figure 9, Modules

For instance, to install the pas2js compiler, project and rtl files you can select the module “`pas2js-rtl`”, press the button “install module” and the **PAS2JS** compiler will be installed.
This process can be repeated for every module in the listbox. After a successful installation of a module there will be a success message in the terminal window.

6 FINISH THE INSTALLATION AND START WORKING.

Figure 10: The starter icon

At this point the full installation of **FPC, Lazarus,** cross-compiler and modules is completed. **FPCUPDELUXE** can be terminated. As mentioned in earlier, **FPCUPDELUXE** creates an icon on the desktop to start **Lazarus.**
You must use this icon to start **Lazarus.** You can check the properties of the program starter to see what command-line parameters are given to start **Lazarus** properly.
During the use of **FPCUPDELUXE** I did not change a lot of options of the program.
On the main screen left below there is a button `Setup+`.
If you press this button the figure on the next page will be shown.

6 FINISH THE INSTALLATION AND START WORKING - CONTINUATION.

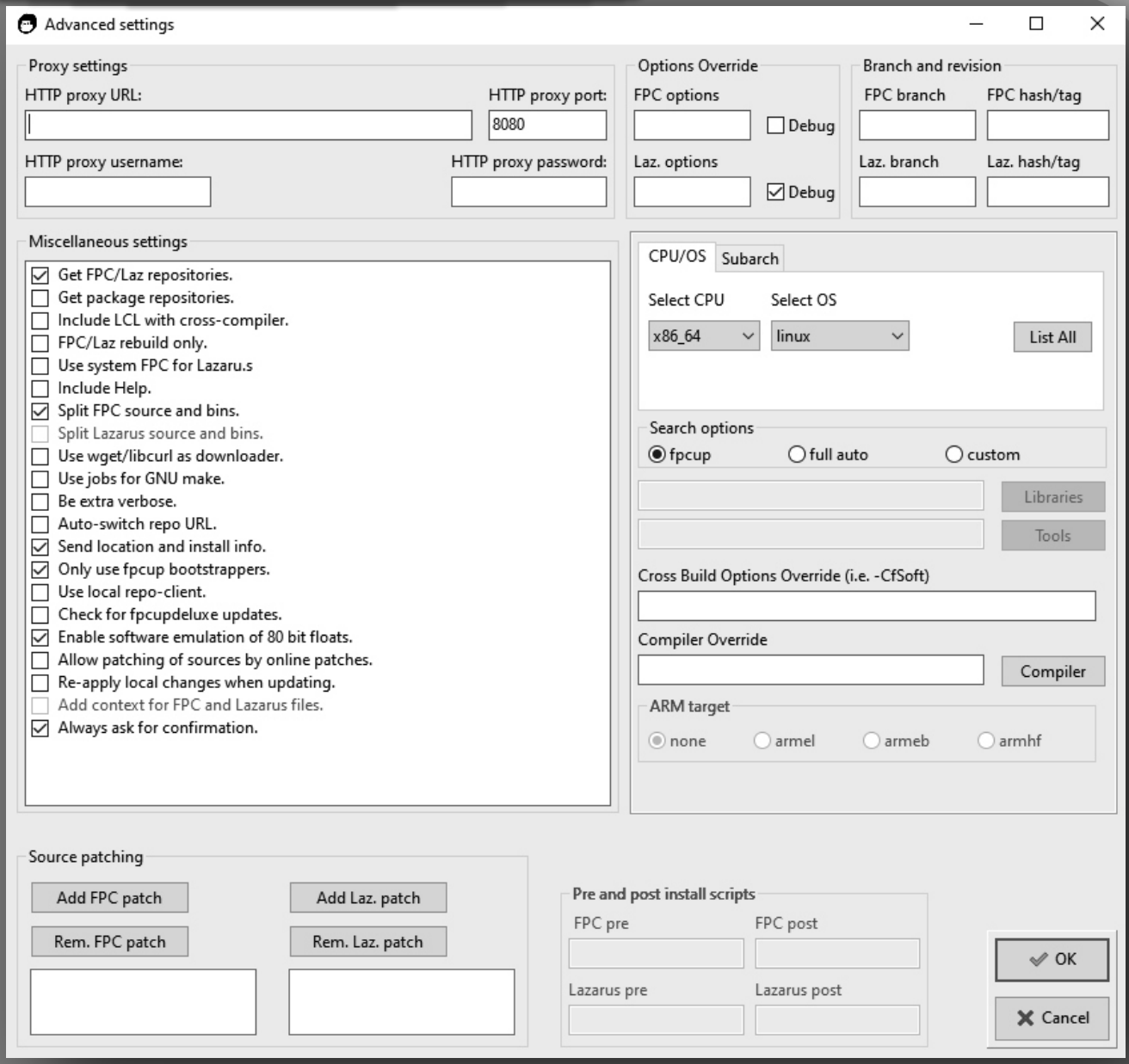


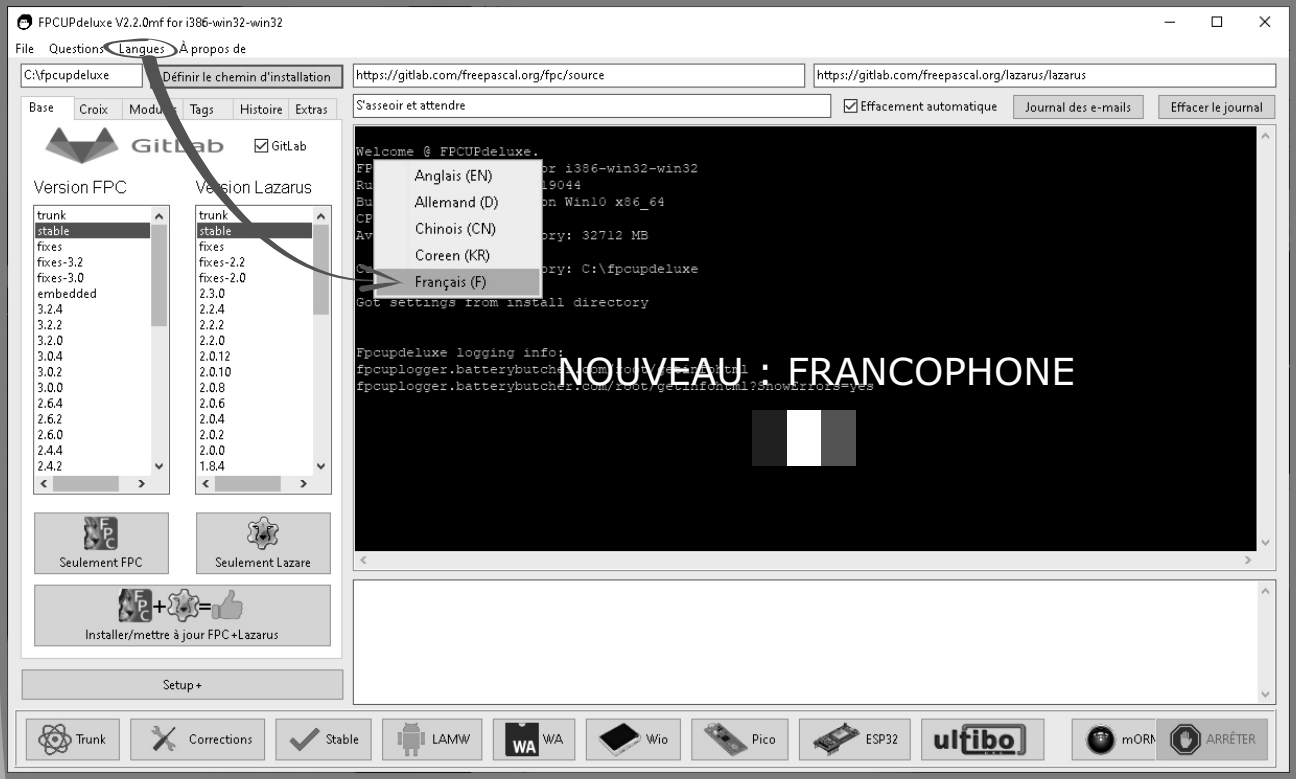
Image 11, Extra Setup

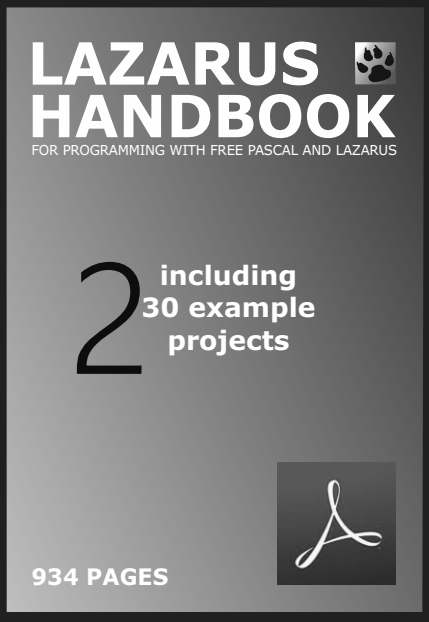
This screen gives the possibility to customize the behavior of **FPCUPDELUXE** in detail.

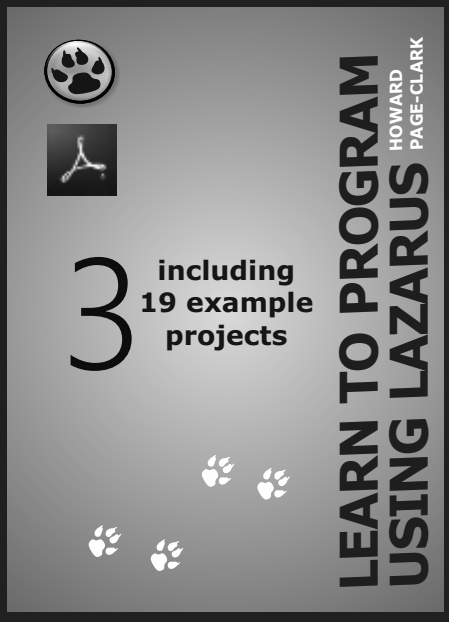
7 CONCLUSION

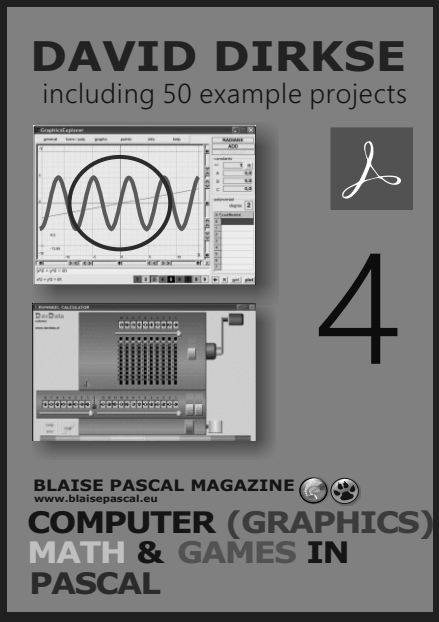
In this article I have shown you how to install **FPC, Lazarus**, cross-compilers and modules by using **FPCUPDELUXE.** The main advantage for me to use this method is that it is possible to install several versions of **FPC** and **Lazarus** without the possibility that the separate versions affect each other. This makes it easy to install a cutting edge trunk compiler and use that to compile my source-code without much trouble. Every installed version is installed in it's own directory, even the configuration and modules are in the installation directory. If I have to remove a version, deleting a directory is sufficient to make this happen. No files needed by a specific version of **FPC, Lazarus** are stored outside the install directory. Use the dedicated topic at the Lazarus forum for extra info and questions.

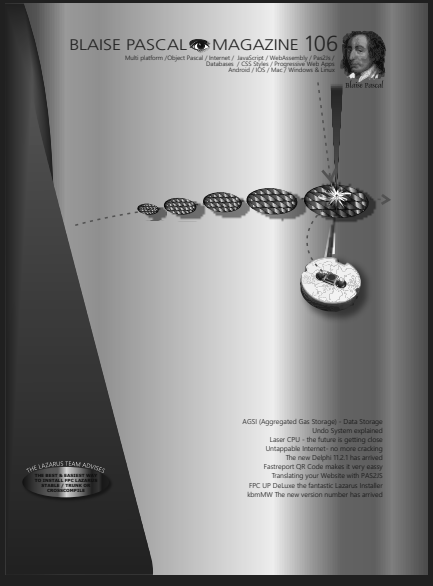

FPCUPdeluxe V2.2.0mf for i386-win32-win32
Datei Ausgaben Sprachen Um
Englisch (EN)
Deutsch (D)
Chinesisch (CN)
Koreanisch (KR)
Französisch (F)
NEU: DEUTSCHSPRACHIG


FPCUPdeluxe V2.2.0mf for i386-win32-win32
File Questions Langues À propos de
Anglais (EN)
Allemand (D)
Chinois (CN)
Coreen (KR)
Français (F)
NOUVEAU : FRANCOPHONE

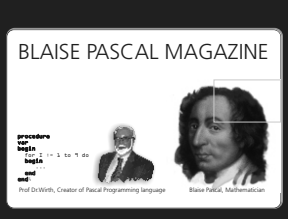








1. One year Subscription
2. The newest LIB Stick - including Credit Card USB stick
3. Lazarus Handbook - Personalized - PDF including Code
4. Book Learn To Program using Lazarus PDF
   including 19 lessons and projects
5. Book Computer Graphics Math & Games
   book + PDF including ±50 projects



BY DETLEF OVERBEEK

starter expert

ABSTRACT

Lasers use gases, solids or liquids as a source of optical amplification, creating coherent laser beams.
These consist of photons - particles representing the smallest discrete amount of a quantum of electromagnetic radiation.
Carbon dioxide (CO2) lasers use gas for optical amplification, while photonic lasers
which are more energy-efficient and compact - use semiconductor crystals to directly amplify the beams.

1 INTRODUCTION

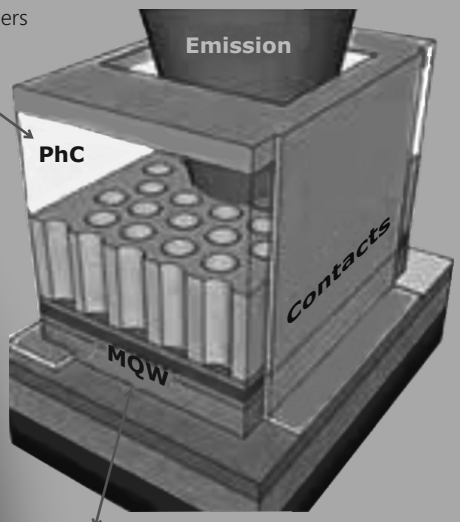
Today, laser manufacturing is largely dominated by **CO2 lasers** and fibre lasers - even for routine tasks such as making tiny holes in circuit boards for smartphones.
It is believed that photonic lasers, which use electricity directly, have the potential to reduce the energy cost of manufacturing.
Halving the costs and emissions of laser cutting should be possible for the future, and perhaps because of their compact size, costs can be reduced even more.

However, before these cost savings can be realised, there are still significant technical hurdles to overcome.
The invention of **PCSELs** (photonic-crystal surface-emitting lasers) at **Kyoto University,** Japan in 1999 improved photon beam quality in the laboratory.
But to use them effectively in production, a trade-off between power and beam quality must be made.

This is a key goal of the "**Photonics and Quantum Technology for Society 5.0**" programme.
(*Photonics is a technology that focuses on detecting, generating, transporting and processing light. The technology is similar to electronics. Only there is one big difference: instead of electrons, photons are used to transmit information. This is both more energy-efficient and faster. Photonics is used, for example, in displays, lighting, lasers, solar cells, sensors and fibre optic networks*).

Although the photonic lasers being produced have not yet completely solved the problems associated with production, the group used mirrors and reformed a PCSEL microstructure to reduce light loss in 2021. With this setup, they performed the first successful machining of a metal surface with a single-chip PCSEL.

Photonic crystal lasers



Multiple Quantum Well Schematic

The multi-quantum well (MQW) solver is a 1D physics-based solver for calculating optical and electronic properties of multi-quantum well stacks.
The solver returns the gain and spontaneous emission coefficients, as well as the electronic band diagram, band structure, and wavefunctions.
The results from the MQW solver are often used as inputs for the TWLM element in INTERCONNECT to model edge-emitting semiconductor lasers or electro-absorption modulators.

The MQW solver simulation object in the finite-element integrated design environment (FE IDE) provides a graphic user interface (GUI) for running the MQW simulations. A set of script-based functions (buildmqwmaterial, mqwgain, mqwindex) can also be used to run the MQW solver.

The next step is to scale up the lasers and develop packaging and cooling systems. Meanwhile, the programme has produced a number of technologies that are in use or about to be used. **PSCEL** sensors will emerge in commercial devices as early as next year.

2 LIGHT WAVES

Many laser manufacturing technologies, which constitute the largest segment of the laser market, entered the commercial market 20-30 years ago.
Japan is in a good position to ride this wave, said Šarunas Vaškelis, CEO of **Direct Machining Control**, a Vilnius-based Lithuanian laser software developer.

Although **Japan** has lagged behind competitors in the **US** and **Germany** in laser technology production since its heyday in the 1990s, it remains a major force in the related optics and photonics industries.

For example, it is a world leader in photonics, which includes technologies such as laser diodes and **LEDs,** fibre optics, optical sensors, displays and solar cells.
Take a look at the country of the manufacturer's headquarters and you'll see **Japan** was in the lead in 2015 with about 30% of the global market, a note from Vaškelis, who prepared a report on the collaboration.

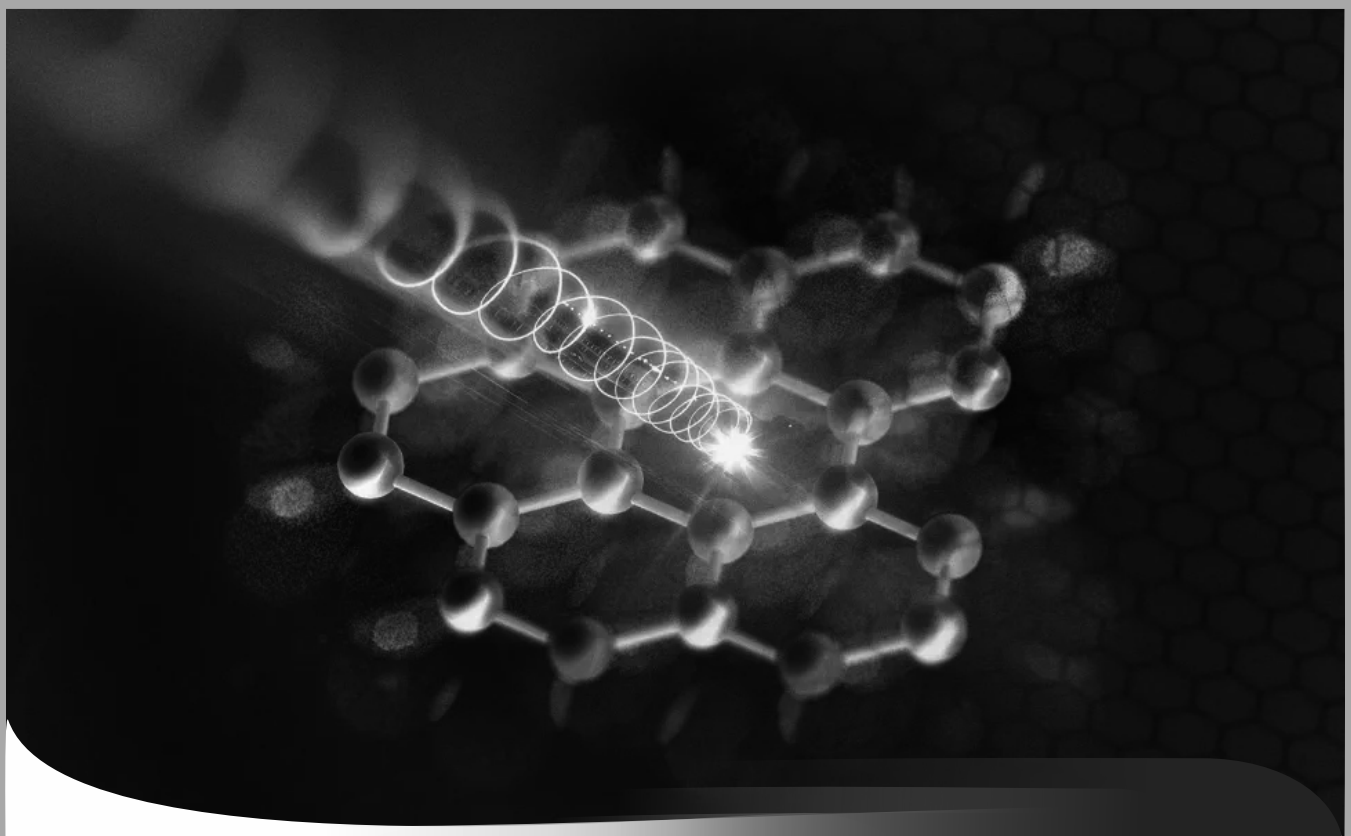
Recent world events are also accelerating the demand for new technologies for efficient, safe and flexible smart factories.
For example, the war in Ukraine - a major global source of oil and gas - has drawn attention to the possibility of energy supply disruptions,
while the COVID-19 pandemic has reinforced the desire to switch to remotely operated facilities.

Uniquely small, efficient PCSELs for **Light Detection and Ranging (LiDAR)** offer new opportunities for remotely guided factory robotics and electric vehicles.

Material-handling AI and photonic communication systems are also reaching realistic use cases and could offer more flexible systems and unprecedented data security for a highly digitised manufacturing paradigm.

A tipping point on sustainability will also affect adoption, according to **John Lincoln**, chief executive of the **UK Photonics Leadership Group**.
There is a global trade-off between long-term government environmental plans and much shorter-term costs to businesses. There is a fairly sudden change to be expected in the need to try to align the two.

(Image credit: Stephen Alvey, Michigan Engineering)



Steadily ongoing advances and continuous progress in electronics and computing have created opportunities to achieve goals that once seemed inconceivable: build autonomous machines, solve complex deep learning problems, and communicate instantaneously across all the places of the planet.

Regrettably all the advances are not at the point where we should be. Our systems-which still rely on these oldfashioned electronic processors are grounded in a very nasty reality: the sheer physics of electrons limits their bandwidth and forces them to produce enormous heat, which means they draw vast amounts of energy. The internet now adays is consuming about 17% of the global energy resources.

As demand for fast and low-energy **ARTIFICIAL INTELLIGENCE (AI)** grows,researchers are exploring ways to push beyond electrons and into the world of **photons.**

They are replacing electronic processors with photonic designs that incorporate lasers and other light components.

While there is skepticism among some observers that the technology can transform analog computing, researchers in the optical space are now building systems demonstrating significant benefits in **AI** and **DEEP LEARNING.**

DONATE FOR UKRAINE AND GET A FREE LICENSE AT:

https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/

(Just click)

COMPONENTS 4 DEVELOPERS

If you are from Ukrainian origin you can get a free Subscription for Blaise Pascal Magazine, we will also give you a free pdf version of the Lazarus Handbook.

You need to send us your Ukrainian Name and Ukrainian email address (that still works for you), so that it proofs you are real Ukrainian.

please send it to editor@blaisepascal.eu and you will receive your book and subscription

BLAISE PASCAL MAGAZINE

DONATE FOR UKRAINE AND GET A FREE LICENSE AT:

https://components4developers.blog/2022/02/26/donate-to-ukraine-humanitarian-aid/
(Just click)

KBMMW PROFESSIONAL AND ENTERPRISE EDITION V. 5.20.01 JUST NOW RELEASED!

This is a significant new release with new high performance transports, **OpenSSL v3 support,** WebSocket support, further improvements to SmartBind, new high performance hashing algorithms, improved RemoteDesktop sample and much more.
This release requires the use of **kbmMemTable** v. 7.96.00 or newer.

- **RAD Studio XE5 to 11 Alexandria 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!

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**

- **NEW: FULL WEBSOCKET SUPPORT.**
  The next release of kbmMW Enterprise Edition will include several new things and improvements.
  One of them is full WebSocket support.
- New I18N context sensitive internationalization framework to make your applications multilingual.
- New ORM LINQ support for Delete and Update.
- Comments support in YAML.
- New StreamSec TLS v4 support (by StreamSec)
- Many other feature improvements and fixes.

Please visit
http://www.components4developers.com
for more information about kbmMW

- 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.

COMPONENTS 4 DEVELOPERS

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