Mein Erfahrungsbericht

Lazarus unter Linux Mint mit fpcupdeluxe installieren und einrichten
und
Einrichten verschiedener Cross Compiler

Quellen:
https://wiki.freepascal.org/fpcupdeluxe/de
https://wiki.freepascal.org/AVR_Embedded_Tutorial_-_Entry_Lazarus_and_Arduino/de
https://www.rompelsoft.de/index.php/download/summary/20-anleitungen/111-cross-compiling-laz-1-9-0-unter-min-18-3
https://forum.lazarus.freepascal.org/index.php?topic=34645.1215
https://wiki.freepascal.org/IDE_Window:_Compiler_Options/de

Wer als Neuling auf Lazarus stößt und dieses mal eben schnell testen möchte der ist mit der Version aus den Repositories gut bedient. Einfach in der Anwedungsverwaltung (Mint) oder im SoftwareCenter (Ubuntu) nach Lazarus suchen, installieren klicken und schon kann es los gehen.

Nach einiger Zeit kann es jedoch vorkommen das man mit dieser Version an Grenzen stößt bzw. das diese Installation Nachteile aufweist. Erstens ist die Version aus den Repositories bei weitem nicht die aktuellste Version. Schwerer wiegt für mich allerdings das die Installation nicht ins home Verzeichnis gemacht wird. Man braucht also für den Zugriff auf die Beispieldateien oder zum Neu-kompilieren immer Rootrechte. Wird die Installation im home Verzeichnis in verschiedene Ordner gemacht ist es auch möglich mehrere Lazarus Versionen gleichzeitig zu betreiben.
Deshalb möchte ich hier, für alle die noch wenig Erfahrung mit der Installation von Lazarus haben, eine mögliche Installation von Lazarus mit dem Installationsprogramm fpcupdeluxe vorstellen. Ebenfalls ist die Integration von Cross-Compilern, mittels fpcupdeluxe, auch von weniger versierten Anwendern zu schaffen.
Alles was ich hier schreibe habe ich mir nicht selbst ausgedacht sondern stammt aus verschiedenen Quellen im Internet. Es kann auch einfachere und bessere Wege geben, ich kann nur sagen bei mir hat es so funktioniert.

Zuerst getestet habe ich dies alles mit fpcupdeluxe1.6.6e und Xubuntu 19.04 64bit in Virtualbox. Es lief alles problemlos durch. Als ich es ein zweites mal in Xubuntu probierte konnte ich leider FPC nicht mit fpcupdeluxe installieren. Auch im Internet stößt man auf etliche Artikel das es zeitweise wohl nicht möglich ist Installationen durch zuführen. Eventuell sollte man es nach einiger Zeit einfach nochmal versuchen.
Danach habe ich das gleiche nochmal unter Linux Mint 19.3 Cinnamon 64bit und mittlerweile fpcupdeluxe1.6.8a durchgeführt.
Ich kann nur sagen wenn es funktioniert ist es genial!

Hier noch das Übliche: Ich gebe keinerlei Garantie für Richtigkeit, ebenso kann ich keinerlei Haftung übernehmen, alles auf eigene Gefahr und trotz sorgfältiger inhaltlicher Kontrolle übernehme ich keine Haftung für die Inhalte externer Links. Für den Inhalt der verlinkten Seiten sind ausschließlich deren Betreiber verantwortlich.

Inhaltsverzeichnis

Installation von fpcupdeluxe:

Zuerst benötigt man folgende Pakete falls noch nicht vorhanden. Im Terminal eingeben:

# Paketliste auf neusten Stand bringen und Updaten.
sudo apt-get update
sudo apt-get upgrade
sudo apt-get autoremove

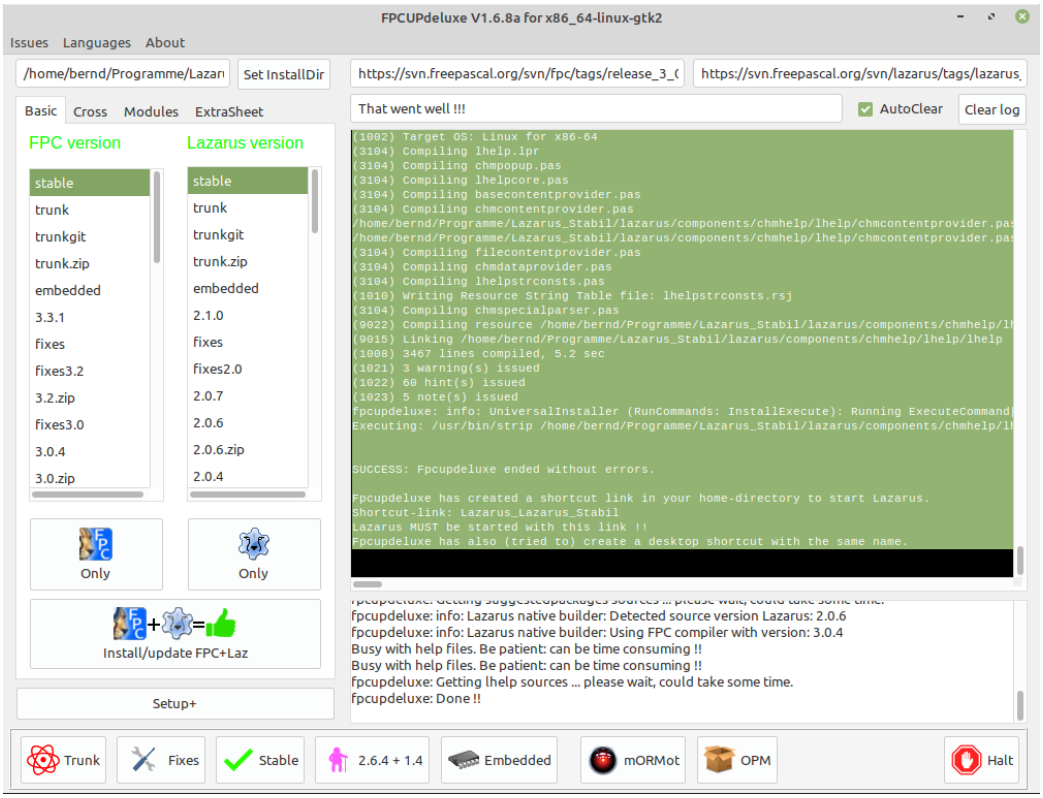
# Die fehlenden Pakete installieren sofern nicht schon vorhanden.
sudo apt-get install libx11-dev
sudo apt-get install libgdk-pixbuf2.0-dev
sudo apt-get install libpango1.0-dev
sudo apt-get install libgtk2.0-dev

# SVN installieren (SVN ist eine freie Software zur zentralen Versionsverwaltung
# von Dateien und Verzeichnissen.)
sudo apt-get install subversion

fpcupdeluxe hier downloaden: https://github.com/newpascal/fpcupdeluxe/releases/latest

In meinem Fall war es: fpcupdeluxe-x86_64-linux

- Immer den neuesten Download verwenden!
- Den Download von FPCUPdeluxe am besten in einen neuen leeren Ordner kopieren.
- Die kopierte Datei am besten im Datei-Manager mit einen Doppelklick starten (eventuell Ausführbar machen).



Ich habe die Datei unter home/meinName/Programme/fpcupdeluxe gespeichert. Nach einem doppelklick startete das Programm.

Installation von Lazarus

Bei set install dir den gewünschten Pfad eingeben wo Lazarus installiert werden soll. Bei mir home/bernd/Programme/Lazarus_Stabil (der Ordner Programme/Lazarus_Stabil muss natürlich angelegt werden).

Wer die Hilfe mit installieren möchte muss auf Setup+ drücken und dort bei Include Help (default= no) einen Haken setzen!

Um die neueste stabile Version zu installieren einfach die obige Einstellung lassen und den Button Install/update FPC+Laz oder auf 

drücken.

Am Ende der Installation sollte so etwas stehen:

SUCCESS: Fpcupdeluxe ended without errors.
Fpcupdeluxe has created a shortcut link in your home-directory to start Lazarus.
Shortcut-link: Lazarus_Lazarus_Stabil
Lazarus MUST be started with this link !!
Fpcupdeluxe has also (tried to) create a desktop shortcut with the same name.

Es ist möglich Lazarus weitere Male zu installieren. Dazu das gleiche wie oben beschrieben einfach in ein anderes Verzeichnis installieren. Wer zum Beispiel eine neue Trunc Version probieren möchte kann dies so machen.

Fpcupdeluxe schließen und Lazarus durch anklicken des neuen Icons auf dem Schreibtisch starten.

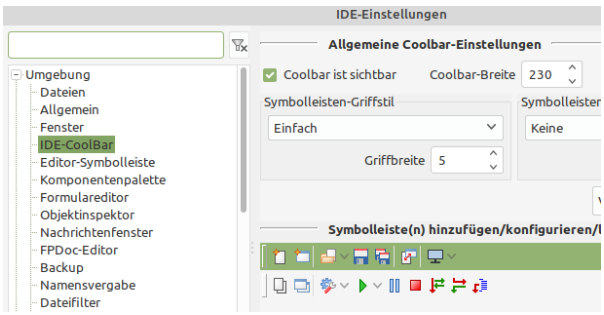
Auf Tools/Options/General gehen und unter Language German(de) auswählen. Mit okay verlassen und Lazarus neu starten. Jetzt ist Lazarus auf deutsch.

Fehlt die Coolbar, dann Werkzeuge/Einstellungen:

Dort einen Haken bei

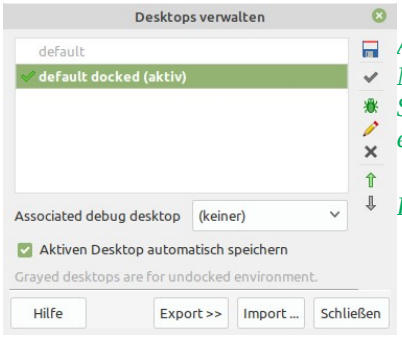
Coolbar ist sichtbar setzen.

In diesem Menü lässt sich die Coolbar auch konfigurieren.



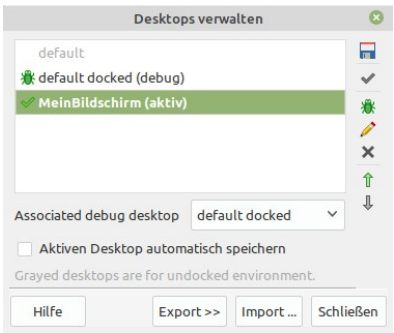
Anchordocking installieren

Jedem der die einzelnen Fenster in Lazarus so wenig mag wie ich, der kann mal Anchordocking testen. Dazu Lazarus schließen fpcupdeluxe starten. Dort unter Modules anchordocking anklicken. Jetzt auf Install module klicken. Das Paket wird installiert und Lazarus neu kompiliert. Jetzt Lazarus neu starten auf Werkzeuge, Desktops dann öffnet sich dieses Fenster:

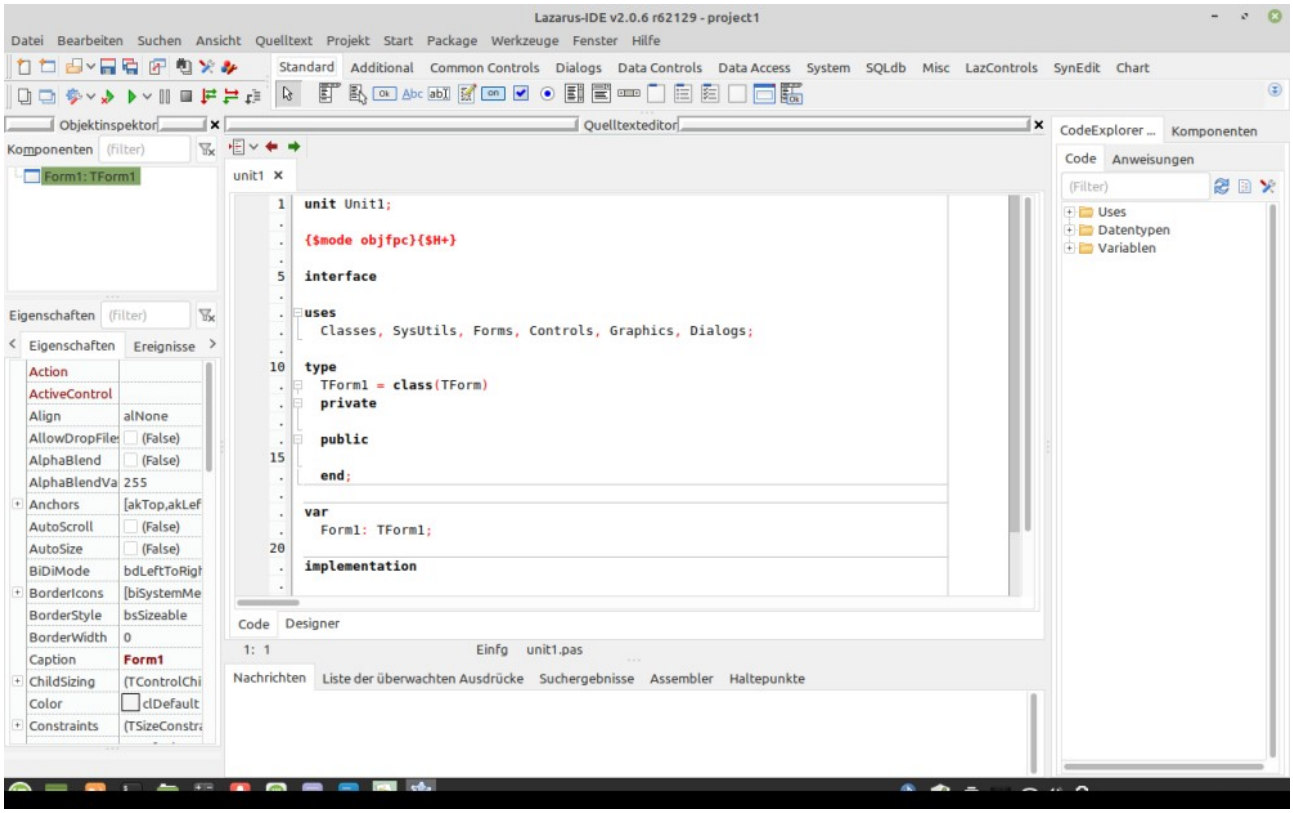


Auf die Diskette klicken und den aktuellen Desktop mit einem eigenen Namen abspeichern. Jetzt kann ohne Risiko herum probiert werden. Sollte man alles verstellen kann man hier das gespeicherte Layout einfach wieder herstellen.

Falls die Diskette nicht erscheint auf Save active desktop as klicken!



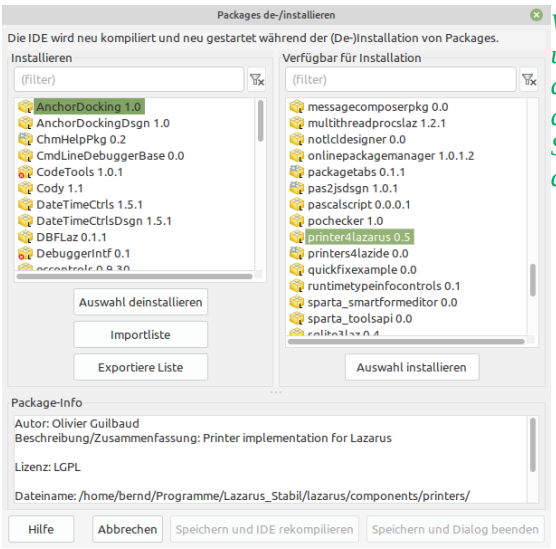
So sieht Lazarus mit der Grundeinstellung von anchordocking aus:



Selbstverständlich können auf die beschriebene Weise alle in fpcupdeluxe enthaltenen Module installiert werden.

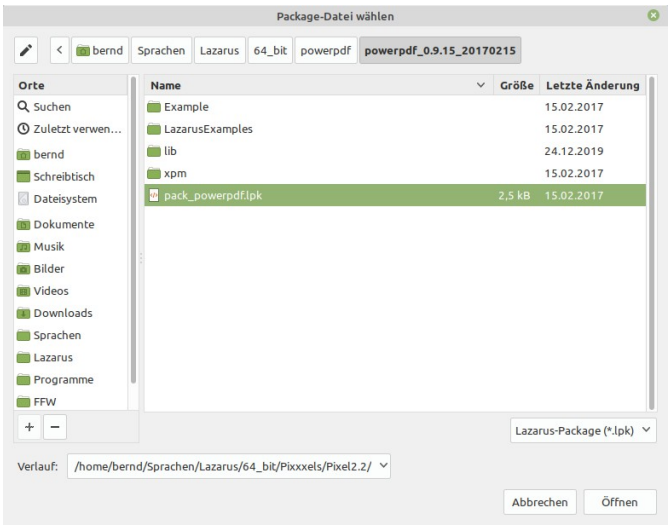
Verfügbare Packages einrichten

Pakete die nicht in fpcupdeluxe enthalten sind müssen auf die herkömmliche Art nach installiert werden. Dazu Lazarus starten, dann Package, Installierte Packages einrichten. Nun öffnet sich folgendes Fenster:

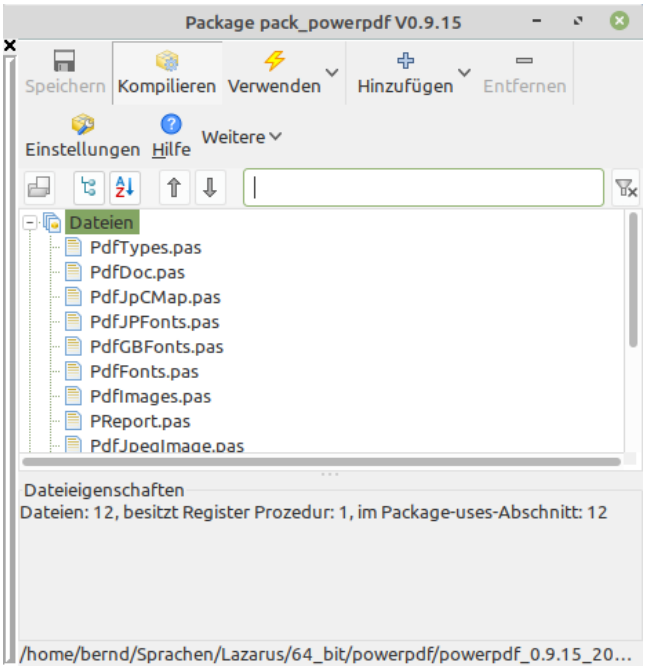


Will man zum Beispiel printer4lazarus nach installieren, dann unter Verfügbar für Installation das gewünschte Paket anklicken, dann Auswahl installieren klicken. Nun erscheint das ausgewählte Paket im linken Fenster unter Installieren. Zuletzt Speichern und IDE rekompilieren anklicken und im darauffolgenden Fenster nochmal bestätigen.

Für Pakete die in Lazarus noch nicht enthalten sind (zum Beispiel powerpdf) besorgt man zuerst das Paket mit der passenden .lpk Datei . Dann Package, Package-Datei(.lpk) öffnen, zur .lpk navigieren und auswählen:



Es erscheint folgendes Fenster hier auf Kompilieren drücken.

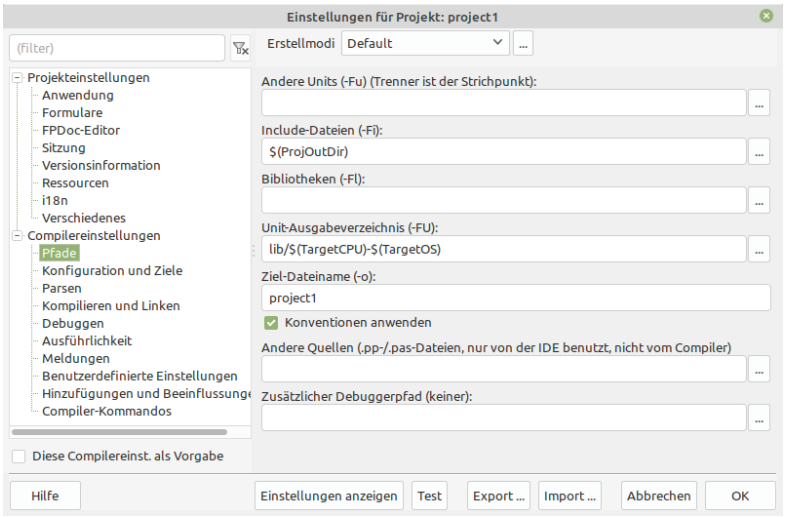


Jetzt befindet sich das Package in der oben beschriebenen Auswahl und kann herkömmlich installiert werden.

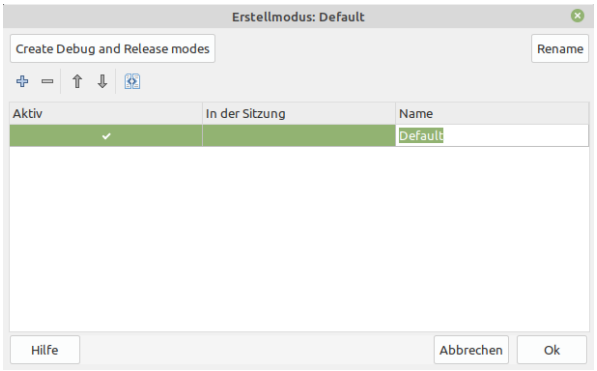
Einrichten der Lazarus IDE für Cross-Kompilierung

Vor dem Hinzufügen weiterer Erstellmodi zum Crosskompilieren kann man den Standartmodi (Default) (bei mir Linux 64bit) umbenennen um später eine bessere Übersicht zu behalten. Dazu folgendes:

Zuerst Projekt, dann Projekteinstellungen anklicken. Dort unter Compilereinstellungen auf Pfade klicken.



Jetzt oben bei Erstellmodi, hinter Default auf die drei … klicken.



Auf Rename doppelklicken und einen neuen Namen eingeben. Ich besitze ein Linux_64, also benenne ich es auch so. Abschließend auf OK.

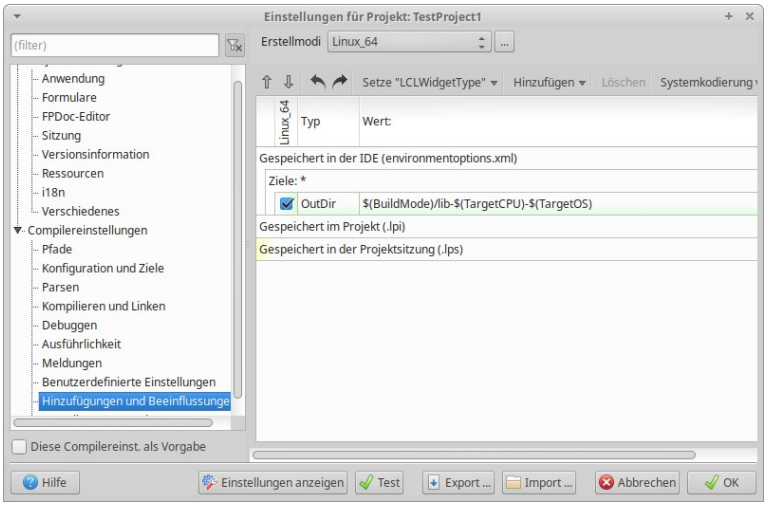
Ohne Änderung werden alle erzeugten Programme im gleichen Verzeichnis abgelegt. Um das Projektverzeichnis übersichtlich zu gestalten sollte man für jeden Erstellmodi jetzt noch auf Hinzufügungen und Beeinflussungen klicken. Dann auf Gespeichert in der IDE. Nun Hinzufügen, Ausgabeverzeichnis ersetzen klicken.

Dann auf Hinzufügen und Ausgabeverzeichnis ersetzen (-FU)

Es kann zum Beispiel folgendes in der Spalte Wert eingetragen werden :

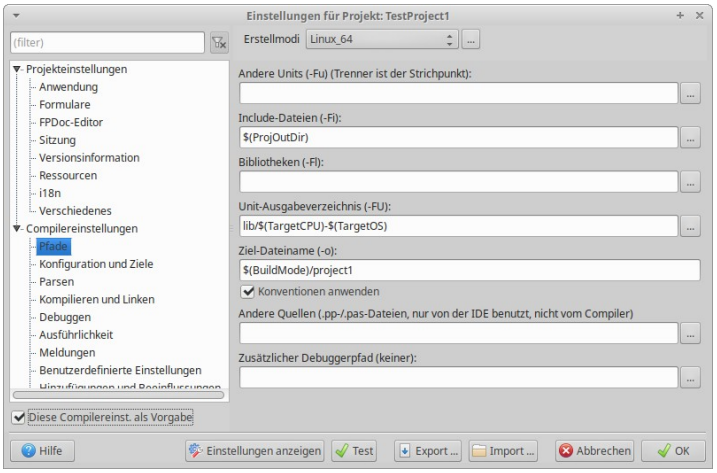
$(BuildMode)/lib-$(TargetCPU)-$(TargetOS)

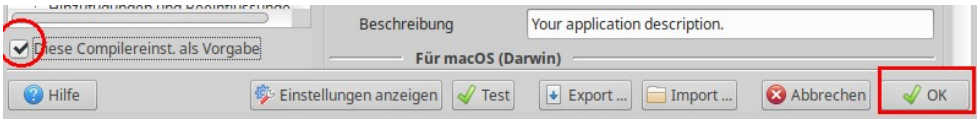
So sollte es jetzt aussehen:



Als nächstes unter Projekt, dann Projekteinstellungen, Compilereinstellungen auf Pfade klicken. In die Zeile „Ziel-Dateiname (-o)“ folgendes eingeben:

$(BuildMode)/project1





Zuletzt den **Haken** bei Diese Compilereinst. Als Vorgabe setzen und mit Ok verlassen.

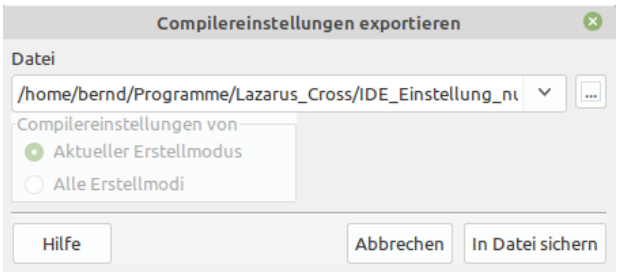
Jetzt werden die erstellten, ausführbaren Dateien in entsprechende Unterverzeichnisse, im Projektverzeichnis, einsortiert. Hier wird also ein Unterverzeichnis mit dem Namen Linux_64 angelegt. Darin befindet sich die ausführbare Datei. In dem Unterverzeichnis lib-x86_64-linux werden die weiteren Dateien abgelegt.

Funktioniert alles wie gewünscht sollte man nach Änderungen die Einstellungen der IDE sichern.

Die Einstellungen der IDE werden nämlich nur bei nachfolgenden Projekten berücksichtigt. Öffnet man ein altes Projekt dann sind dort die damals angelegten Einstellungen wirksam. Über den Import einer Sicherung holt man sich ganz leicht die aktuellen Einstellungen in das geöffnete alte Projekt.

Einstellungen der IDE sichern

Wer seine Einstellung zum Beispiel vor oder nach einer Änderung mal sichern möchte kann dies wie jetzt beschrieben tun:

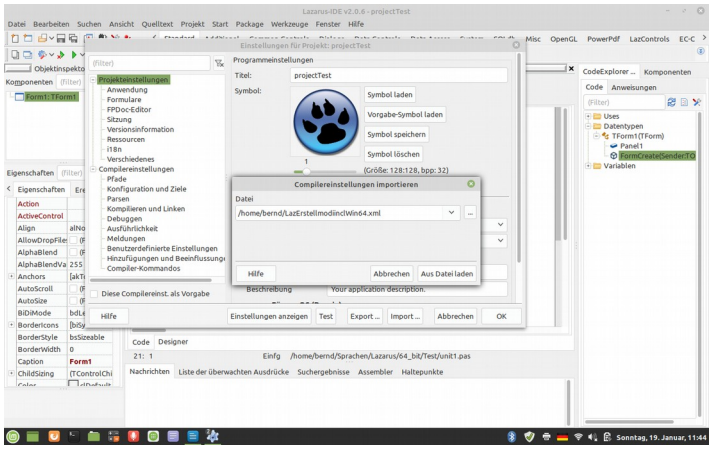


Unter Projekt, dann Projekteinstellungen unten auf den Button Export klicken.

Es erscheint das links stehende Fenster. Hier könnt ihr einen Pfad und einen Dateinamen für eine Sicherung eingeben. Zuletzt auf In Datei sichern klicken. Nun befindet sich in dem angegebenen Pfad eine .xml Datei. Sollte später was schief laufen kann man mit Import die Einstellungen der IDE wieder herstellen.

Achtung: Sind mehrere Modi vorhanden kann man mit dem Radiobutton auswählen ob nur der aktuelle Erstellmodi oder Alle Erstellmodi in der Datei gesichert werden sollen!

Die Einstellungen der IDE werden nur bei nachfolgenden Projekten berücksichtigt. Öffnet man ein altes Projekt dann sind dort die damals angelegten Einstellungen wirksam. Über den Import einer Sicherung holt man sich ganz leicht die aktuellen Einstellungen in das geöffnete alte Projekt. Projekteinstellungen, Import, Aus Datei laden.



Im Anschluss habe ich die Source eines minimalen Programms kopiert welches ich mit allen Cross-Compilern getestet habe:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs,ExtCtrls;
type
  { TForm1 }
  TForm1 = class(TForm)
    Panel1 : TPanel;
    procedure FormCreate(Sender: TObject);
  private
  public
  end;
var
  Form1: TForm1;
implementation
{$R *.lfm}
{ TForm1 }
procedure TForm1.FormCreate(Sender: TObject);
begin
 self.Caption    := 'Test zum Einrichten';
 self.Left       := 200;
 self.Top        := 200;
 self.Width      := 400;
 self.Height     := 200;
 self.Color      := clwhite;
 Panel1          := TPanel.Create(self);
 Panel1.Parent   := self;
 Panel1.Left     := 50;
 Panel1.Top      := 50;
 Panel1.Width    := 300;
 Panel1.Height   := 100;
 Panel1.Color    := cllime;
 Panel1.Caption  := 'Erfolgreich kompiliert';
end;
end.
```

CrossCompiler für Linux 32bit installieren

Um von Linux64 nach Linux32 crosskompilieren zu können müssen als erstes die Multilib Pakete nachinstalliert werden. Sind diese Installiert kann man unter einem Linux64 System auch Linux32bit Anwendungen starten. Wer dies nicht möchte sollte hier aufhören.

Folgende Befehle im Terminal ausführen:

```
# 32-Bit-Pakete abrufen

sudo apt-get install gcc-multilib

sudo apt-get install libx11-dev:i386

sudo apt-get install pixbuf2.0-0:i386

sudo apt-get install libgtk2.0-0:i386

# fehlende Symlinks erstellen

sudo ln -s -f /usr/lib/i386-linux-gnu/libgdk_pixbuf-2.0.so.0 /usr/lib/i386-linux-gnu/libgdk_pixbuf-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0 /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libgdk-x11-2.0.so.0 /usr/lib/i386-linux-gnu/libgdk-x11-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libgobject-2.0.so.0 /usr/lib/i386-linux-gnu/libgobject-2.0.so

sudo ln -s -f /lib/i386-linux-gnu/libglib-2.0.so.0 /lib/i386-linux-gnu/libglib-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libgthread-2.0.so.0 /usr/lib/i386-linux-gnu/libgthread-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libgmodule-2.0.so.0 /usr/lib/i386-linux-gnu/libgmodule-2.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libpango-1.0.so.0 /usr/lib/i386-linux-gnu/libpango-1.0.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libcairo.so.2 /usr/lib/i386-linux-gnu/libcairo.so

sudo ln -s -f /usr/lib/i386-linux-gnu/libatk-1.0.so.0 /usr/lib/i386-linux-gnu/libatk-1.0.so
```

Wichtig:

Leider bekam ich unter Mint Cinnamon beim kompilieren meines ersten kleinen Projektes folgende Fehlermeldung: linker: /usr/bin/ld: -lglib-2.0 kann nicht gefunden werden, projectTest.lpr(21,1) Error: Error while linking

Nachdem ich noch den folgenden zusätzlichen Link erstellt habe hat alles funktioniert:

sudo ln -s -f /usr/lib/i386-linux-gnu/libglib-2.0.so.0 /usr/lib/i386-linux-gnu/libglib-2.0.so

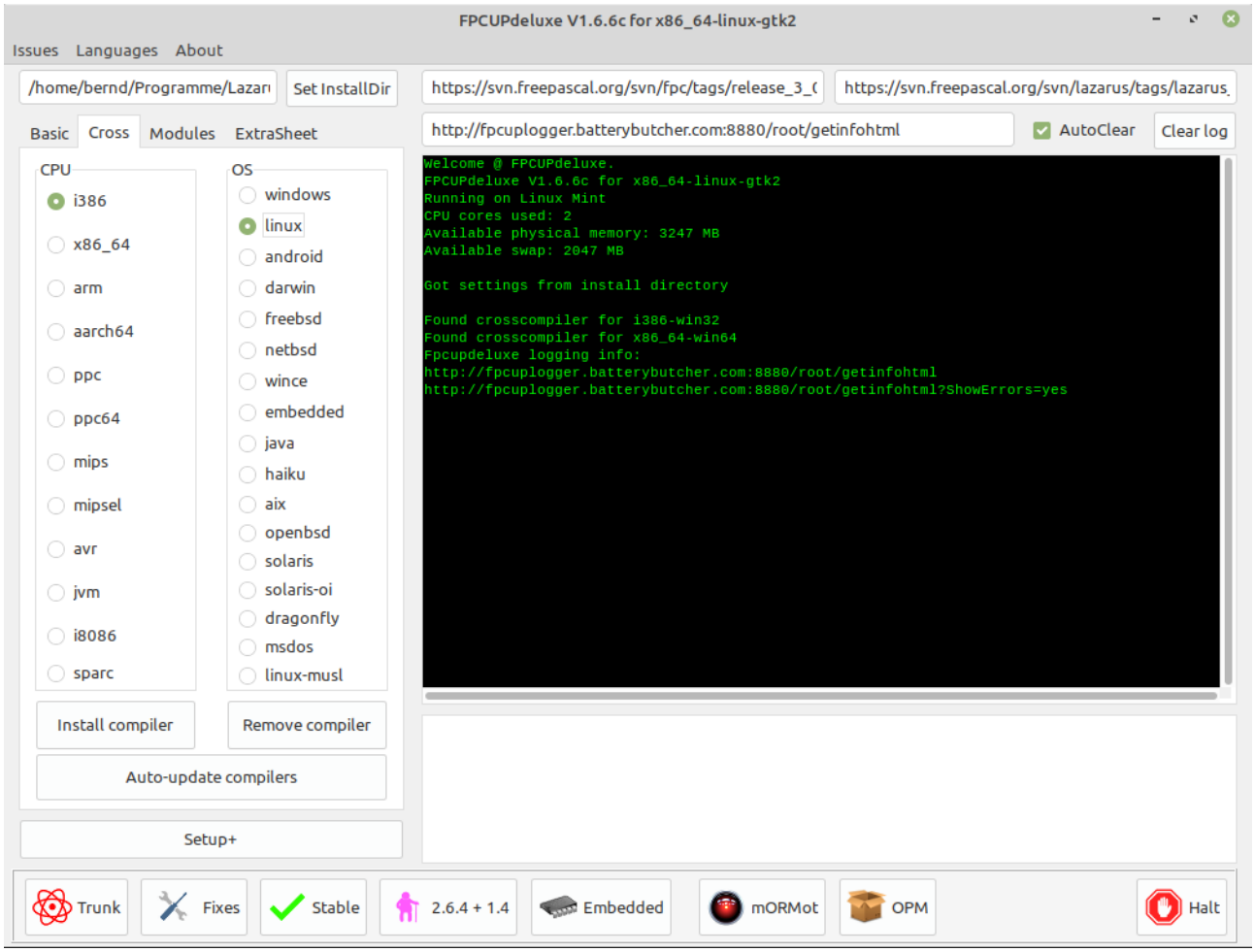
(sollte das Paket komplett fehlen folgendes probieren: sudo apt-get install libglib2.0-0:i386 und dann nochmal den Link esrtellen)

Wer das nicht alles einzeln eingeben möchte kopiert die Befehle einfach in eine Textdatei und stellt noch folgende Zeile an den Anfang:

#!/bin/bash

Die Textdatei dann zum Beispiel als script32bit.sh abspeichern, ausführbar machen, im Terminal ins Verzeichnis wechseln und mit sudo ./script32bit.sh starten.

Jetzt fpcupdeluxe starten.



Obige Einstellung wählen und **Install Compiler** anklicken.

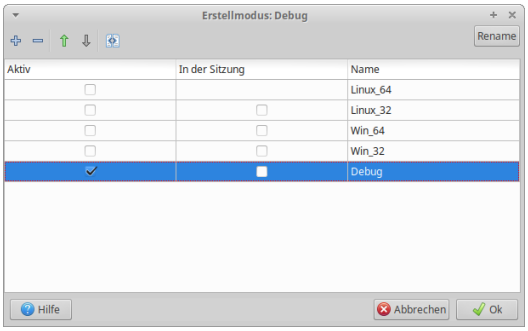
Nach erfolgreicher Installation Fpcupdeluxe schließen und Lazarus öffnen.

Lazarus IDE für Linux32 einrichten

Weiteren Erstellmodi für Linux32 anlegen:

Zuerst Projekt, dann Projekteinstellungen anklicken. Dort unter Compilereinstellungen auf Hinzufügungen und Beeinflussungen klicken. Hinter Erstellmodi auf die 3 … klicken.

Jetzt auf Create Debug and Releas modes klicken. (Nur beim ersten neuen Modi, dann mit + hinzufügen).



Wie neben stehend den Namen für Linux_32bit eingeben und auf Ok klicken.

Info:Mit den Pfeiltasten kann die Anzeige Reihenfolge eingestellt werden.

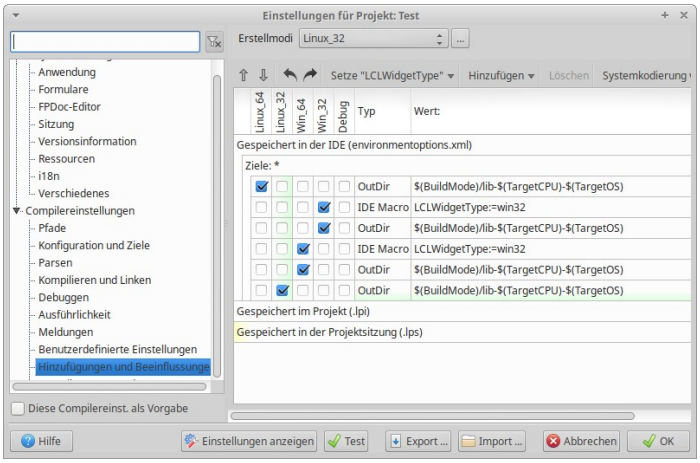
Ohne folgende Änderung werden alle erzeugten Programme im gleichen Verzeichnis abgelegt. Um das Projektverzeichnis übersichtlich zu gestalten sollte man für jeden Erstellmodi jetzt noch auf Hinzufügen, Ausgabeverzeichnis ersetzen klicken. Wichtig: bei Erstellmodi muss Linux_32 ausgewählt sein.

Dann auf Hinzufügen und Ausgabeverzeichnis ersetzen (-FU)

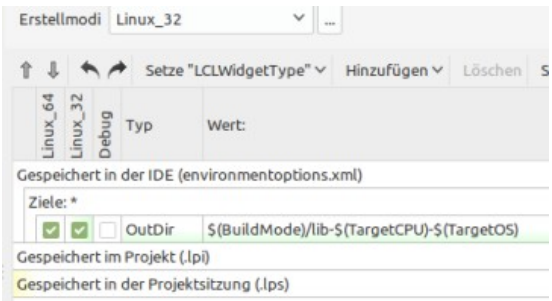
Es muss folgendes in der Spalte Wert eingetragen werden (falls es nicht schon drin steht):

$(BuildMode)/lib-$(TargetCPU)-$(TargetOS)

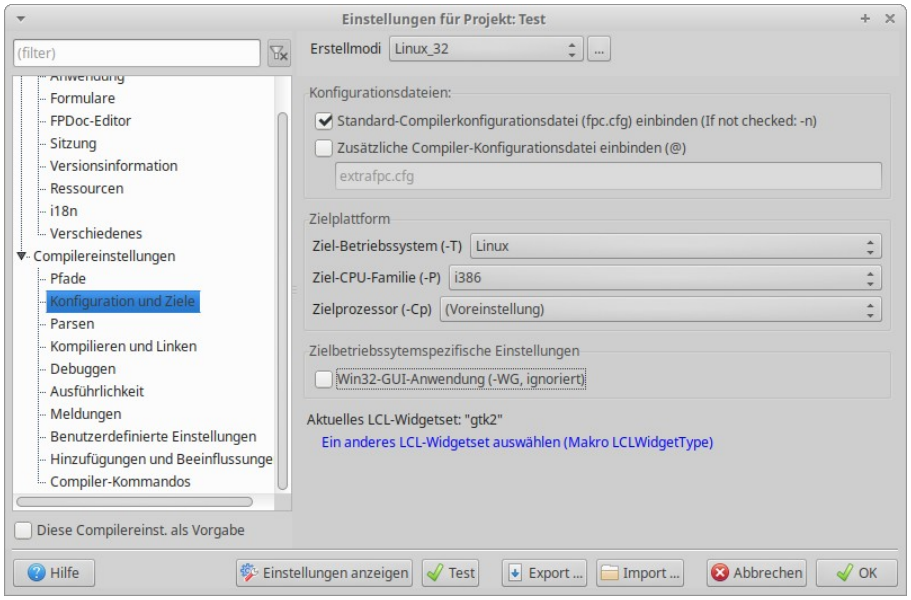
So oder so ähnlich sollte es jetzt aussehen:

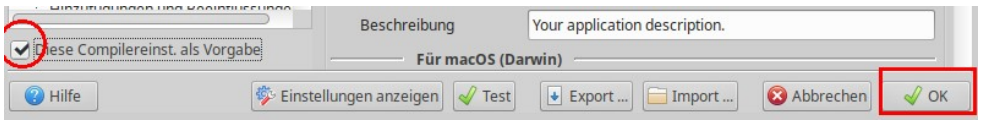


Es geht aber auch so:



Danach unter Projekt, dann Projekteinstellungen auf Konfiguration und Ziele gehen und folgendes einstellen:



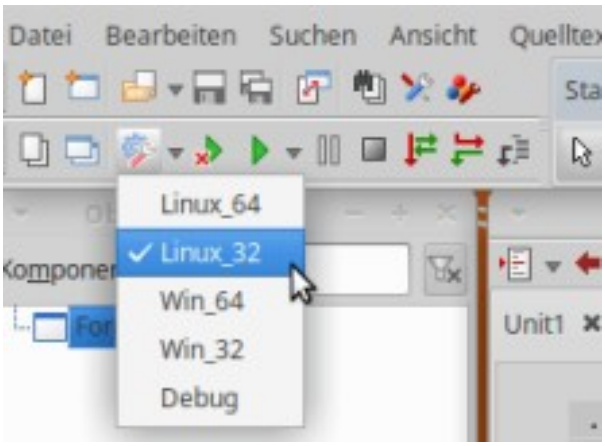


Zuletzt den **Haken** bei Diese Compilereinst. Als Vorgabe setzen und mit Ok verlassen.

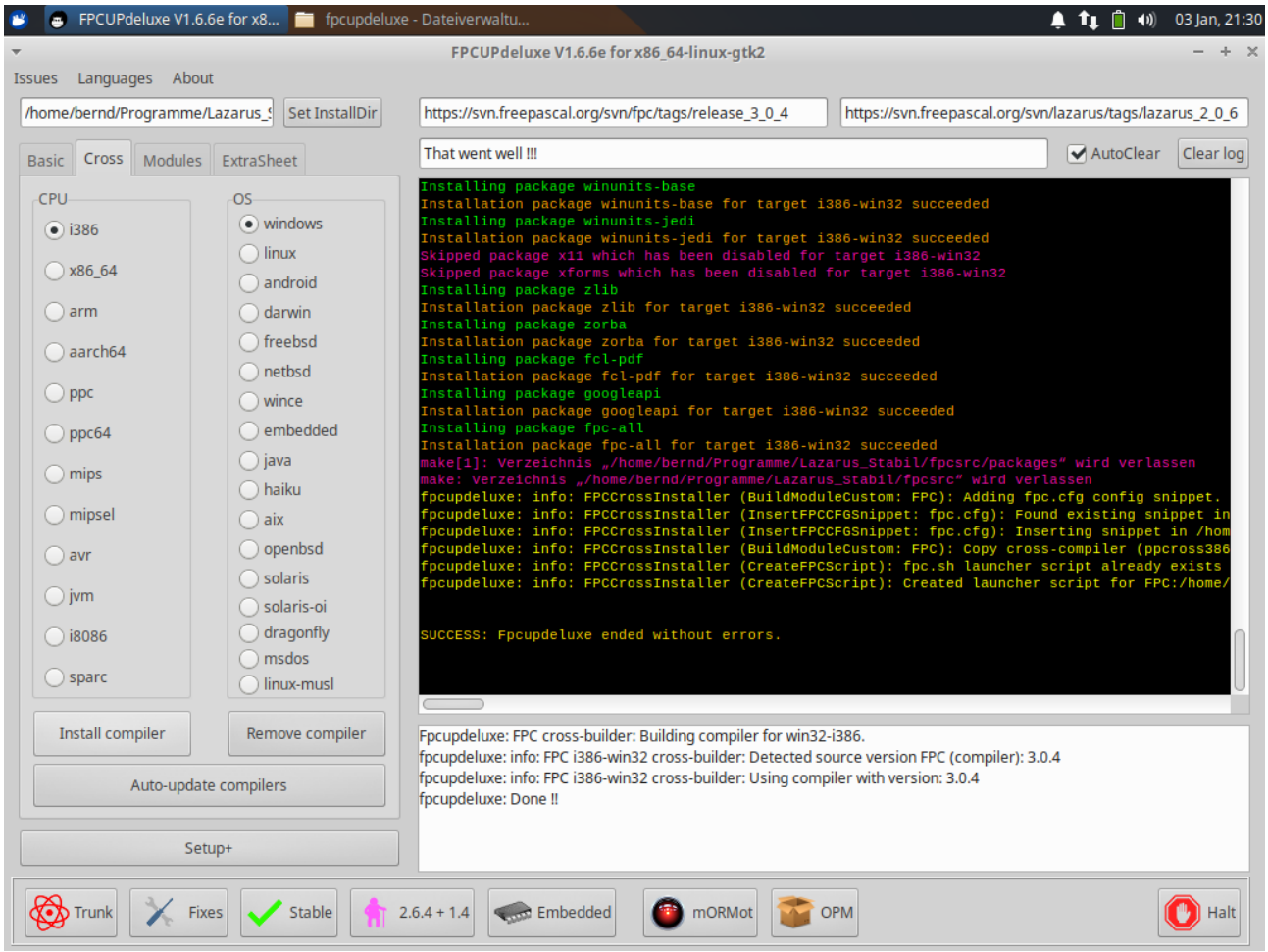
Jetzt können Linux 32bit Anwendungen erstellt werden.

Es kann mit Start F9 kompiliert werden!

Nachdem weitere Erstellmodi zum Cross-Kompilieren eingerichtet wurden kann man mit einem Klick auf den Haken neben dem Zahnrad alle vorhandenen Modi einfach zum Kompilieren auswählen.



CrossCompiler für Windows 32bit installieren



fpcupdeluxe starten.

Obige Einstellung wählen und **Install Compiler** anklicken.

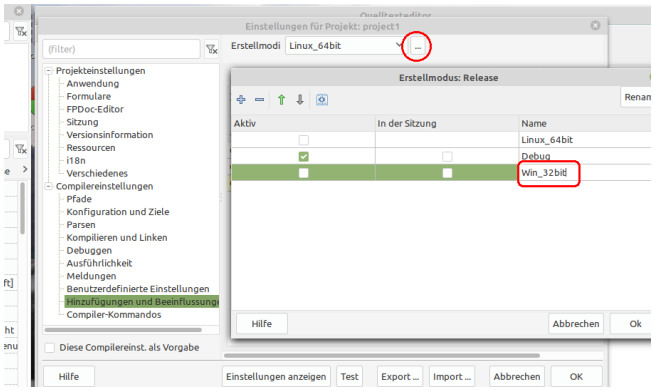
Nach erfolgreicher Installation Fpcupdeluxe schließen und Lazarus öffnen.

Lazarus IDE für Win32 einrichten

Weiteren Erstellmodi für Win32 anlegen:

Zuerst Projekt, dann Projekteinstellungen anklicken. Dort unter Compilereinstellungen auf Hinzufügungen und Beeinflussungen klicken. Hinter Erstellmodi auf den Button mit den . . . klicken.

Jetzt auf Create Debug and Releas modes klicken. (Nur beim ersten neuen Modi, dann mit + hinzufügen).



Wie neben stehend den Namen für

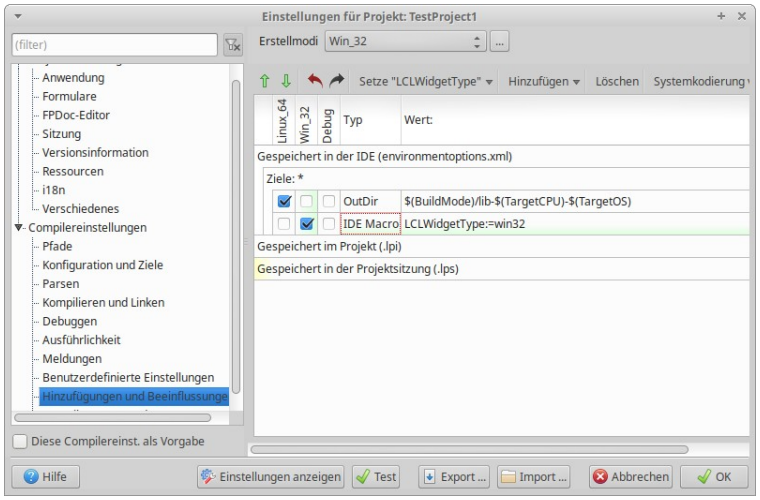
Win_32bit eingeben und auf Ok klicken.

Info:Mit den Pfeiltasten kann die

Anzeige Reihenfolge eingestellt

werden.

Den Erstellmodi Win_32bit auswählen. In die Zeile Gespeichert in der IDE (environmentoptions.xml) klicken. Jetzt auf Setze „LCLWidgetType“ klicken und Wert „win32“ auswählen.

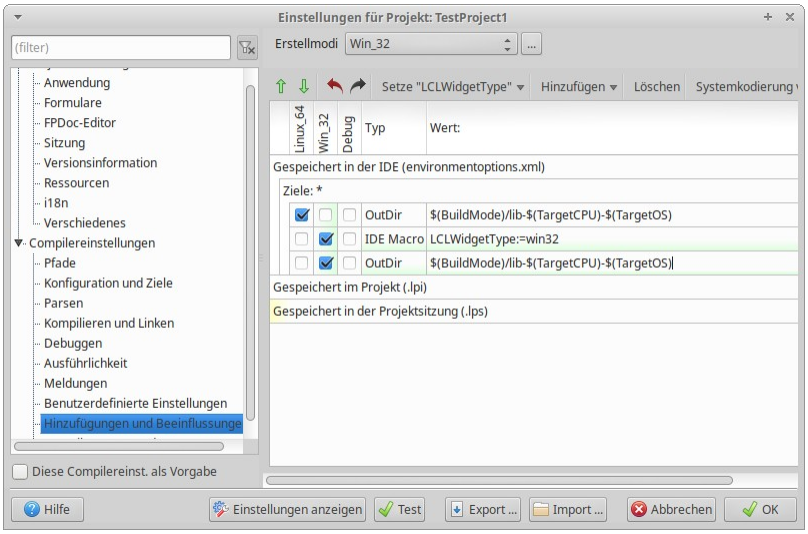


Ohne Änderung werden alle erzeugten Programme im gleichen Verzeichnis abgelegt. Um das Projektverzeichnis übersichtlich zu gestalten sollte man für jeden Erstellmodi jetzt noch in die Zeile Gespeichert in der IDE (environmentoptions.xml) klicken. Dann auf Hinzufügen und Ausgabeverzeichnis ersetzen (-FU)

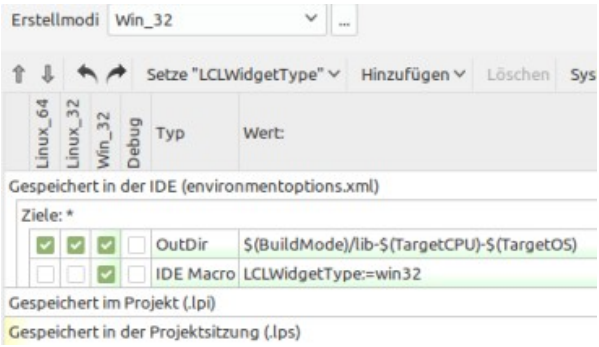
Es kann folgendes in der Spalte Wert eingetragen werden :

$(BuildMode)/lib-$(TargetCPU)-$(TargetOS)

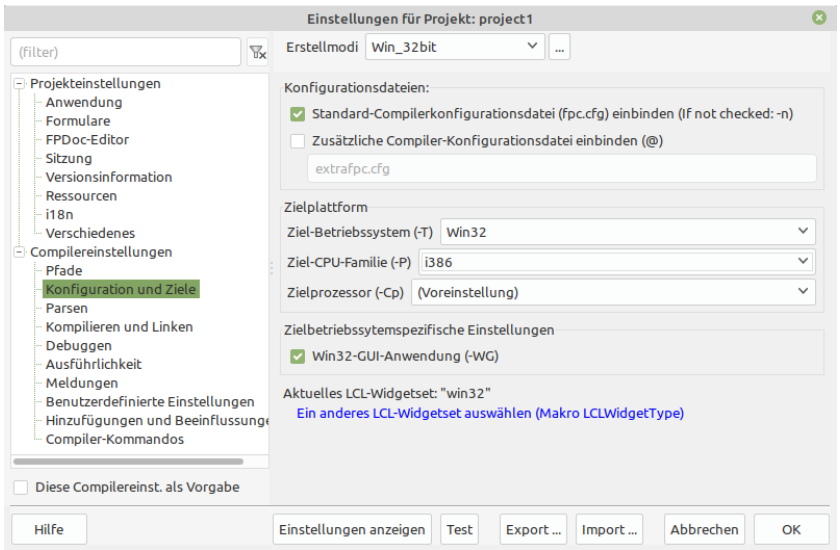
So oder so ähnlich sollte es jetzt aussehen:

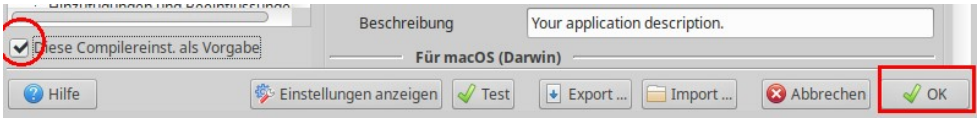


Es geht auch so:



Danach unter Projekt, dann Projekteinstellungen auf Konfiguration und Ziele gehen und folgendes einstellen:



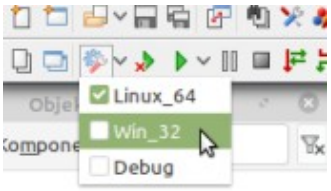


Zuletzt den **Haken** bei Diese Compilereinst. als Vorgabe setzen und mit Ok verlassen.

Jetzt können Windows 32bit Anwendungen erstellt werden.

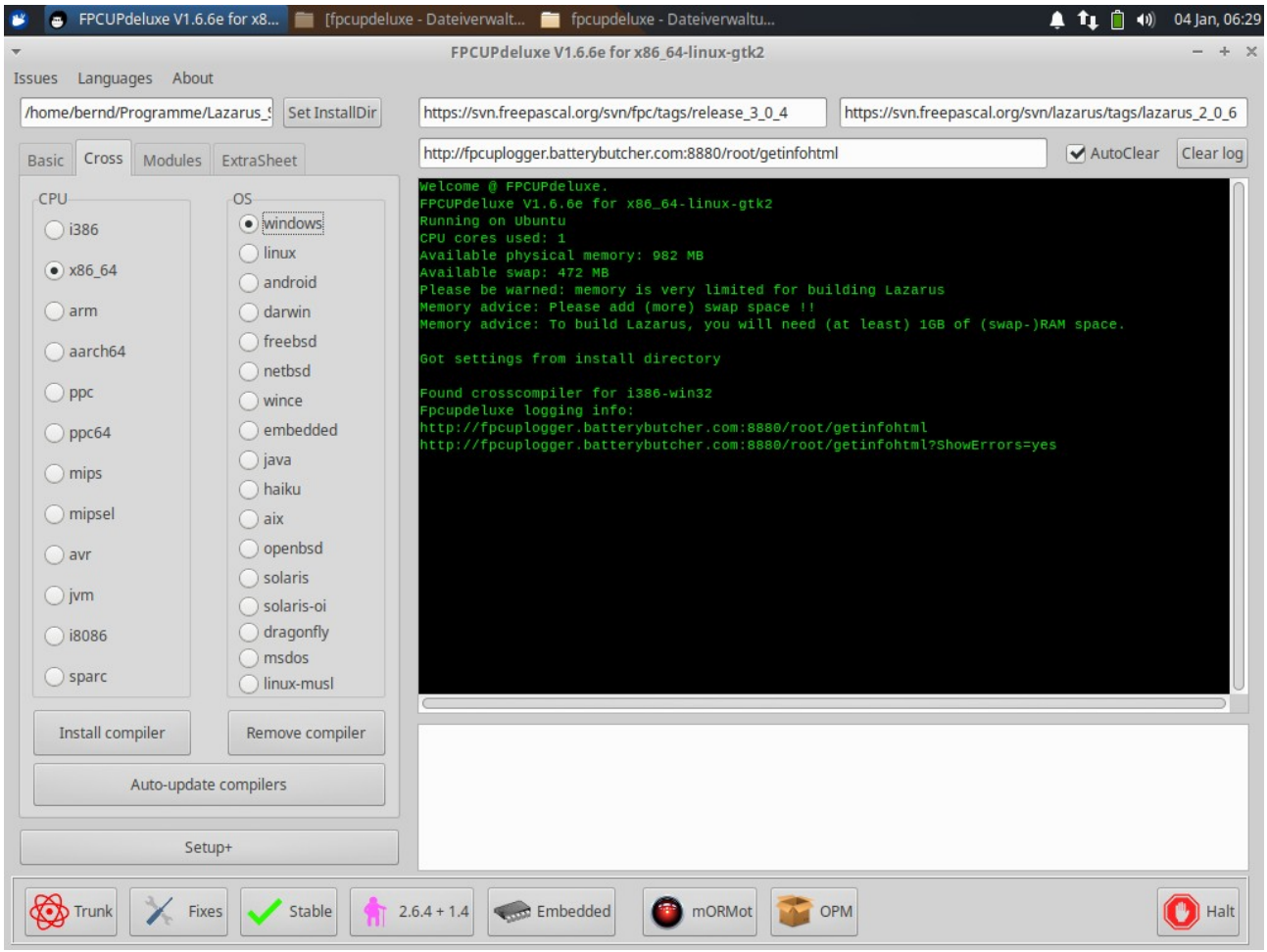
Bitte nur mit strg+F9 kompilieren nicht starten!

Nachdem weitere Erstellmodi zum Cross-Kompilieren eingerichtet wurden kann man mit einem Klick auf den Haken neben dem Zahnrad alle vorhandenen Modi einfach zum Kompilieren auswählen.



CrossCompiler für Windows 64bit installieren

fpcupdeluxe starten.



Obige Einstellung wählen und **Install Compiler** anklicken.

Nach erfolgreicher Installation Fpcupdeluxe schließen und Lazarus öffnen.

Lazarus IDE für Win64 einrichten

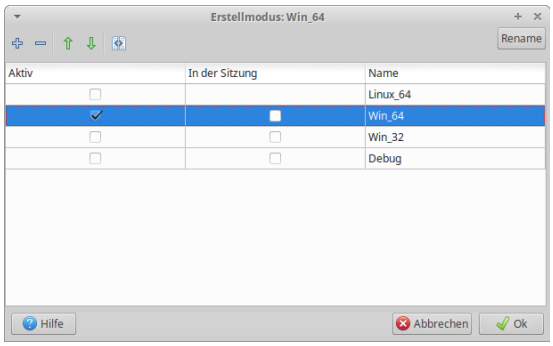
Weiteren Erstellmodi für Win64 anlegen:

Zuerst Projekt, dann Projekteinstellungen anklicken. Dort unter Compilereinstellungen auf Hinzufügungen und Beeinflussungen klicken.

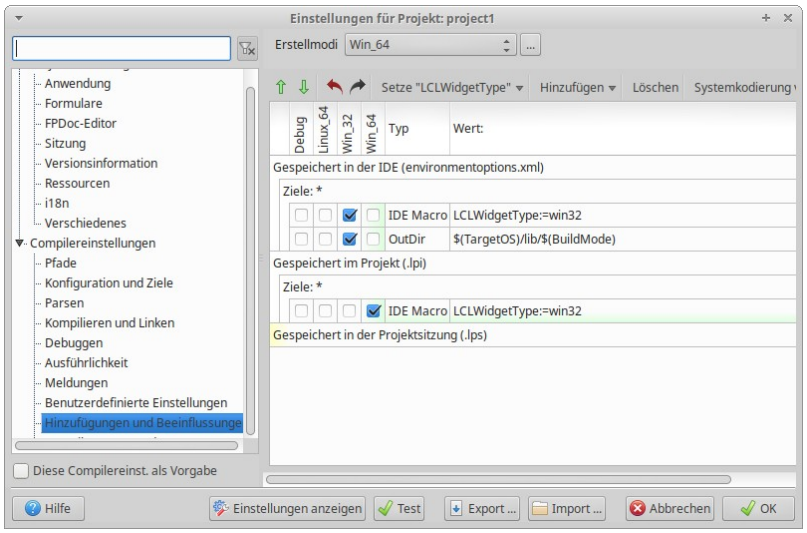
Jetzt auf Create Debug and Releas modes klicken. (Nur beim ersten neuen Modi, dann mit + hinzufügen).

Wie neben stehend den Namen für Win_64bit eingeben und auf Ok klicken.

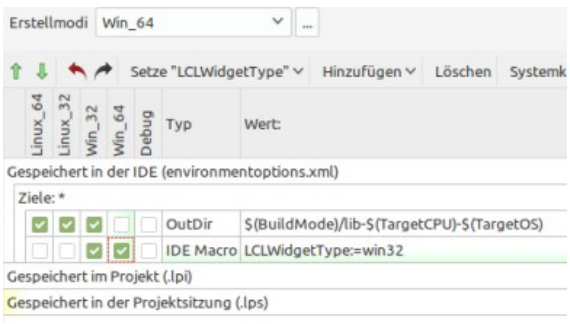
Info:Mit den Pfeiltasten kann die Anzeige Reihenfolge eingestellt werden.



Den Erstellmodi Win_64bit auswählen. In die Zeile Gespeichert in der IDE (environmentoptions.xml) klicken. Jetzt auf Setze „LCLWidgetType“ klicken und **auch hier** den Wert „win32“ auswählen.



Es geht auch so:



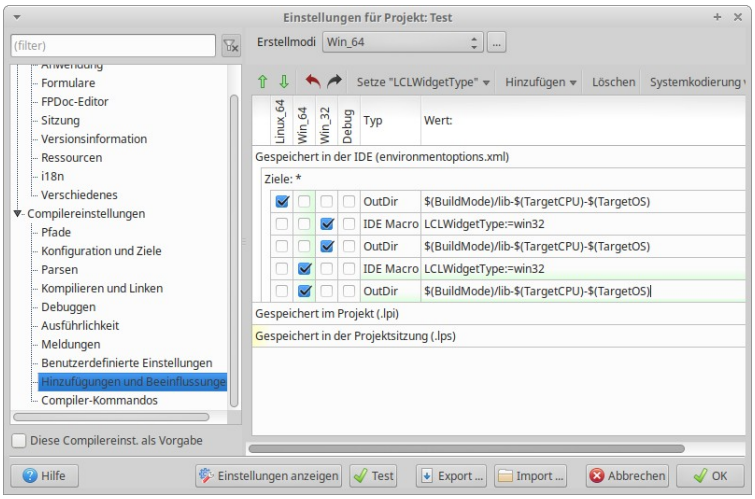
Ohne folgende Änderung werden alle erzeugten Programme im gleichen Verzeichnis abgelegt. Um das Projektverzeichnis übersichtlich zu gestalten sollte man für jeden Erstellmodi jetzt noch auf Hinzufügen, Ausgabeverzeichnis ersetzen klicken.

Dann auf Hinzufügen und Ausgabeverzeichnis ersetzen (-FU)

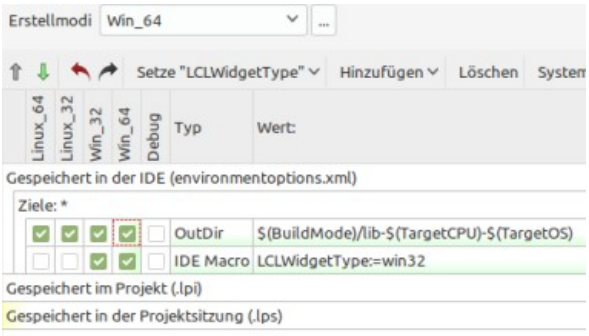
Es muss folgendes in der Spalte Wert eingetragen werden (falls es nicht schon drin steht):

$(BuildMode)/lib-$(TargetCPU)-$(TargetOS)

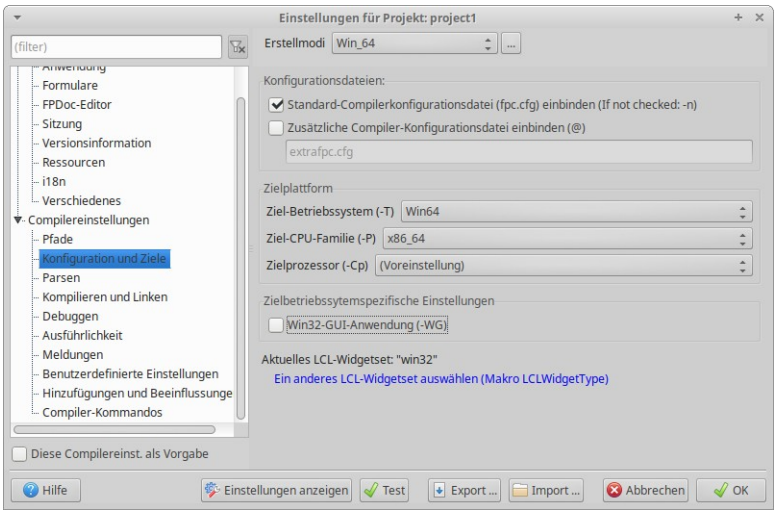
So oder so ähnlich sollte es jetzt aussehen:

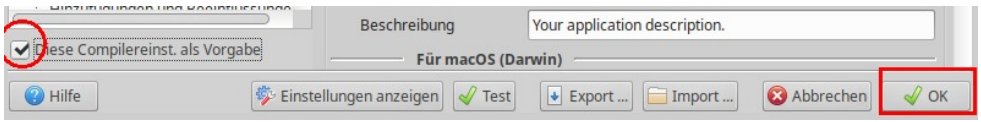


Es geht auch so:



Danach unter Projekt, dann Projekteinstellungen auf Konfiguration und Ziele gehen und folgendes einstellen:



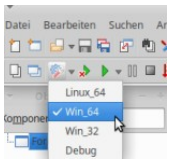


Zuletzt den **Haken** bei Diese Compilereinst. Als Vorgabe setzen und mit Ok verlassen.

Jetzt können Windows 64bit Anwendungen erstellt werden.

Bitte nur mit strg+F9 kompilieren nicht starten!

Nachdem weitere Erstellmodi zum Cross-Kompilieren eingerichtet wurden kann man mit einem Klick auf den Haken neben dem Zahnrad alle vorhandenen Modi einfach zum Kompilieren auswählen.



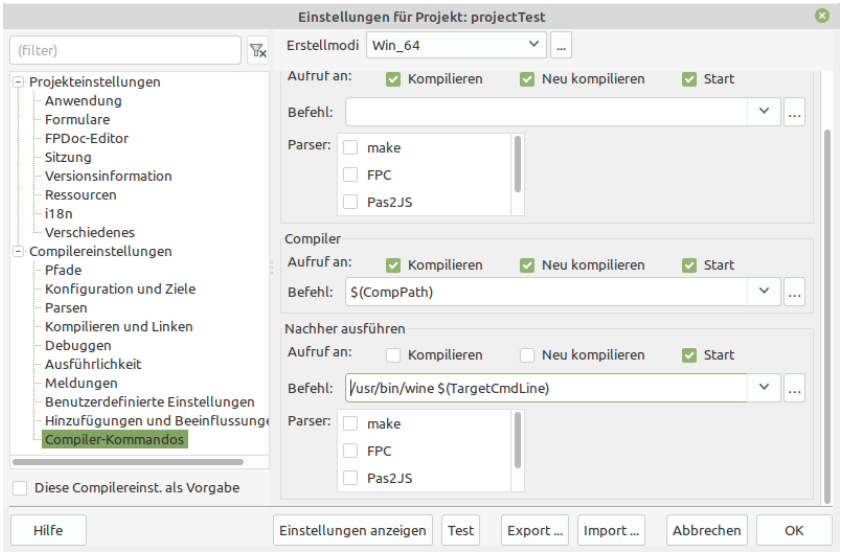
Fehlt in der Coolbar das Symbol für Start ohne Debugger dann auf Werkzeuge/Einstellungen/IDE Coolbar/Konfigurieren/Befehle aus dem Menü Start. Mit Pfeiltasten hinzufügen.

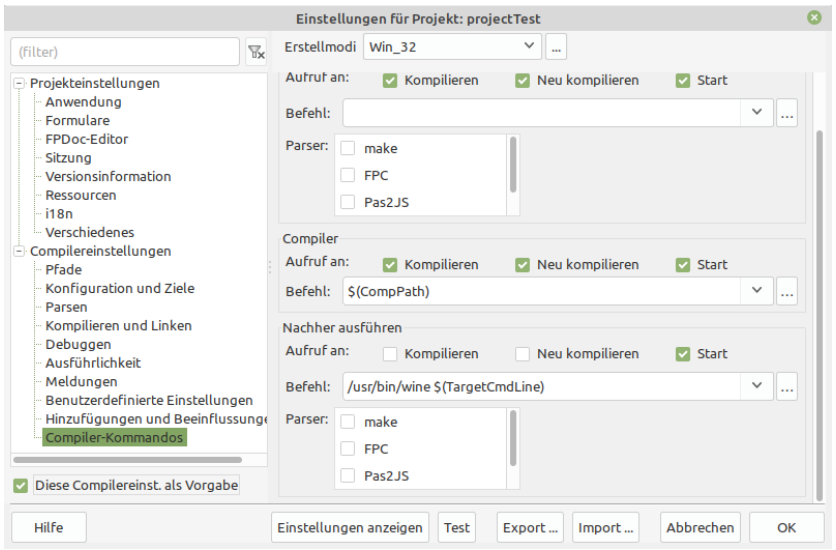
Wine in Lazarus einbinden um Windowsanwendungen zu starten:

Zuerst muss natürlich Wine installiert sein. Ich habe dies einfach über die Anwendungsverwaltung in Mint gemacht.

Dann Lazarus starten. Projekt, Projekteinstellungen, Compiler-Kommandos. Hier **jeweils** für die Erstellmodi Win_32 und Win_64 folgende Einträge vornehmen:

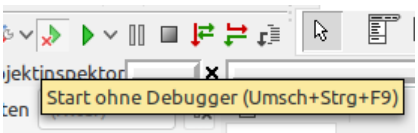
Unter Nachher ausführen in die Zeile Befehl: /usr/bin/wine $(TargetCmdLine) eintragen. Dann die Häkchen bei Kompilieren und Neu kompilieren entfernen.





Zuletzt bei Diese Compilereinst. als Vorgabe ein Haken machen und mit OK verlassen.

Bitte ohne Debugger, also mit Umsch+Strg+F9 starten. Erzeugte Windowsanwendungen werden jetzt an Wine weiter gereicht.



Anmerkung: wird im Quellcode nichts geändert wird beim 2x drücken von Umsch+Strg+F9 nichts gemacht. Abhilfe schafft zum Beispiel irgendwo ein Leerzeichen einzufügen.

Android und Lazarus unter Linux Mint

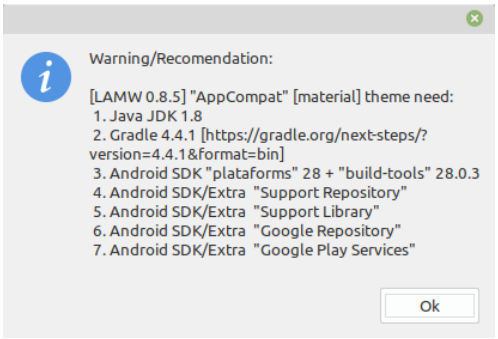
Wer unter Linux mit Lazarus nach Android kompilieren möchte muss einiges an Einstellungen vornehmen. Ich habe es ausschließlich unter Linux Mint 19.3 Cinnamon 64bit getestet. Zuerst in einer Virtualbox Version.

Meine Quelle:
https://forum.lazarus.freepascal.org/index.php/topic,40750.msg281604.html#msg281604
https://forum.lazarus.freepascal.org/index.php/topic,38777.msg264419.html#msg264419
https://wiki.lazarus.freepascal.org/LAMW

Ich habe mich dazu entschieden das Modul LAMW zu verwenden, es läuft auch unter Linux und nimmt einem beim Einrichten der Lazarus IDE sehr viel Arbeit ab!

Liste von Android Versionen: https://de.wikipedia.org/wiki/Liste_von_Android-Versionen

Die in fpcupdeluxe derzeit verwendete LAMW (LazarusAndroidModulWizard) Version benötigt folgende Voraussetzungen:



So gehts los: Bitte folgendes ins Terminal eingeben:

sudo apt-get install -y libx11-dev libgtk2.0-dev libgdk-pixbuf2.0-dev libcairo2-dev libpango1.0-dev libxtst-dev libatk1.0-dev libghc-x11-dev freeglut3 freeglut3-dev

sudo apt-get install -y git subversion make build-essential gdb zip unzip unrar wget

sudo apt-get install -y openjdk-8-jdk android-tools-adb ant

#der nächste Befehl war bei mir nicht nötig:

sudo update-java-alternatives --set /usr/lib/jvm/java-1.8.0-openjdk-amd64

#legt ein Verzeichnis an:

mkdir -p "$HOME/Programme/android"

Folgende Programme downloaden und in den Ordner home/*Benutzername*/Programme/android kopieren und dort extrahieren:

Verschiedene Versionen von Gradle findet man hier (ich nahm 4.4.1, sonst Befehle anpassen):

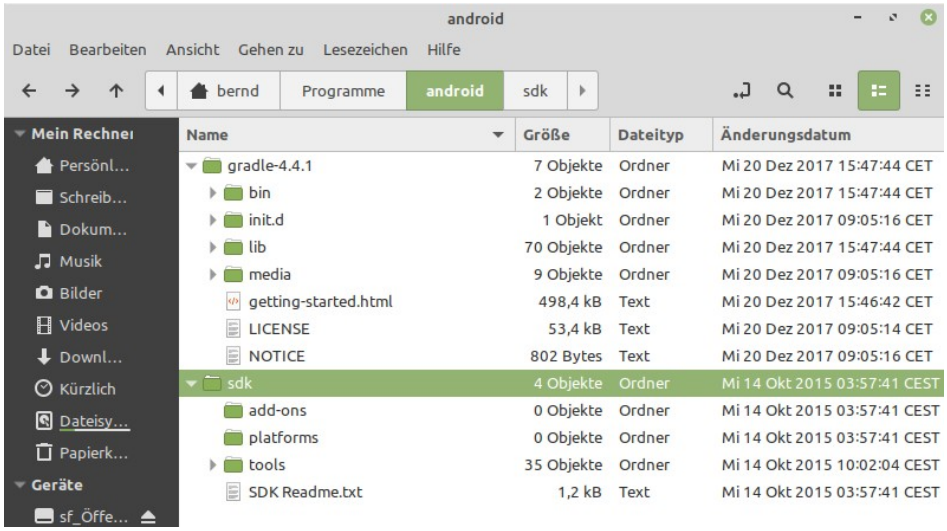
https://gradle.org/releases/ dort auf binary only klicken

Android SDK findet man hier:

http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz

Nachdem die Datei nach android kopiert und dort entpackt wurde das Verzeichnis android-sdk-linux in sdk umbenennen.

Jetzt sollte es so aussehen:



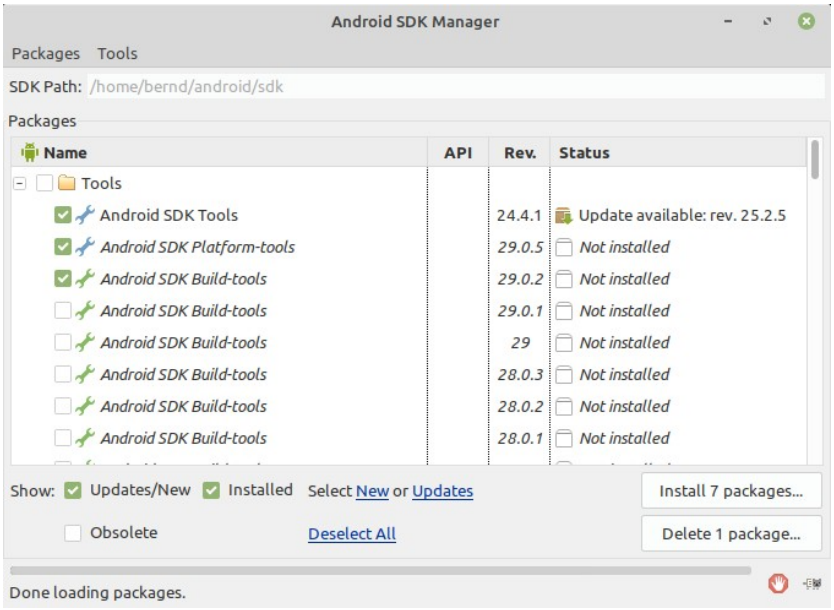
Ins Terminal wechseln und ins Verzeichnis tools wechseln.

cd /home/bernd/Programme/android/sdk/tools dann

./android

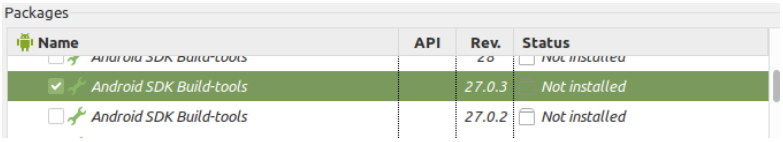
eingeben. Jetzt startet der SDK-Manager.

Es ist die neuste Version Android 10 (API29) vorausgewählt.



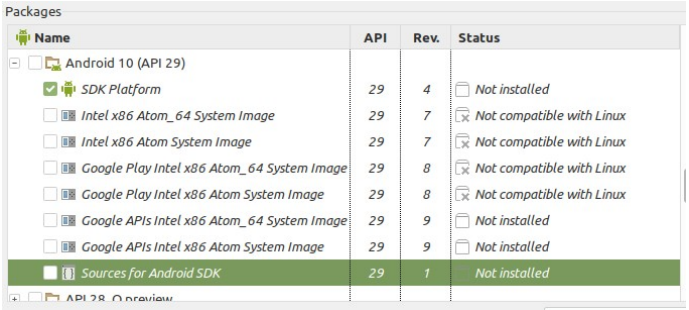
Die 3 Häckchen SDK Tools, SDK Platform-tools und SDK Build-tools drin lassen.

Zusätzlich noch die Build-tools 27.0.3 anwählen (werden für 28 und 29 zusätzlich benötigt)

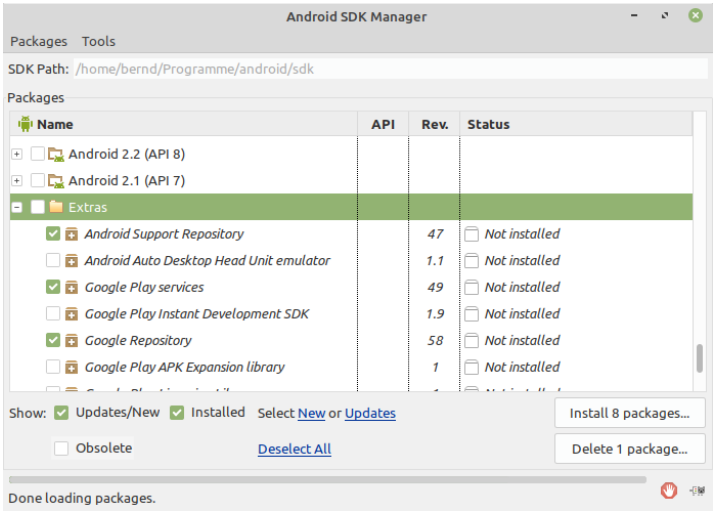


Bei Android 10 (API29) benötigt man nur die SDK Platform. Die anderen Haken entfernen. Die Systemimages werden nur benötigt wenn man zum Testen einen Emulator anlegen möchte. Ich möchte erst mal auf einen alten Handy testen.

[Wer eine ältere Version nehmen möchte muss die dem entsprechende SDK Platform wählen bzw. die passenden Build-tools nehmen. Zum Beispiel SDK Platform von API 25 mit SDK Buil-tools 25.0.3. Es muss dann aber auch die passende NDK Datei ausgesucht werden. Für API 25 funktionierte bei mir android-ndk-r10e-linux-x86_64.]

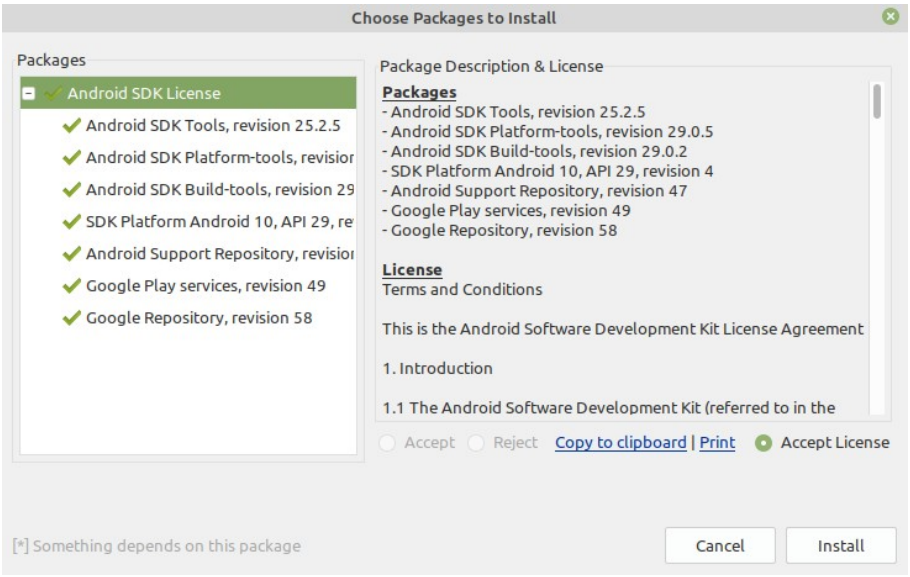


Jetzt noch:



Auf Install 8 packages... klicken.

Dann Accept License anwählen und Install klicken.



Wenn alles fertig ist alle Fenster schließen und den SDK Manager beenden.

Jetzt das aktuelle NDK downloaden:

https://developer.android.com/ndk/downloads

Bei mir war es android-ndk-r21-linux-x86_64. Diese Datei nach Programme/android/sdk kopieren und dort entpacken. Den Ordner anschließend in ndk-bundle umbenennen.

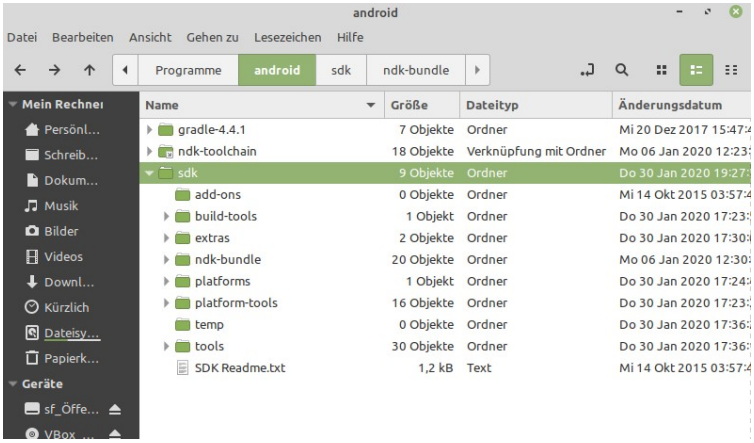
Jetzt im Terminal folgende Links anlegen:

ln -sf "$HOME/Programme/android/sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.9/prebuilt/linux-x86_64/bin" "$HOME/Programme/android/ndk-toolchain"

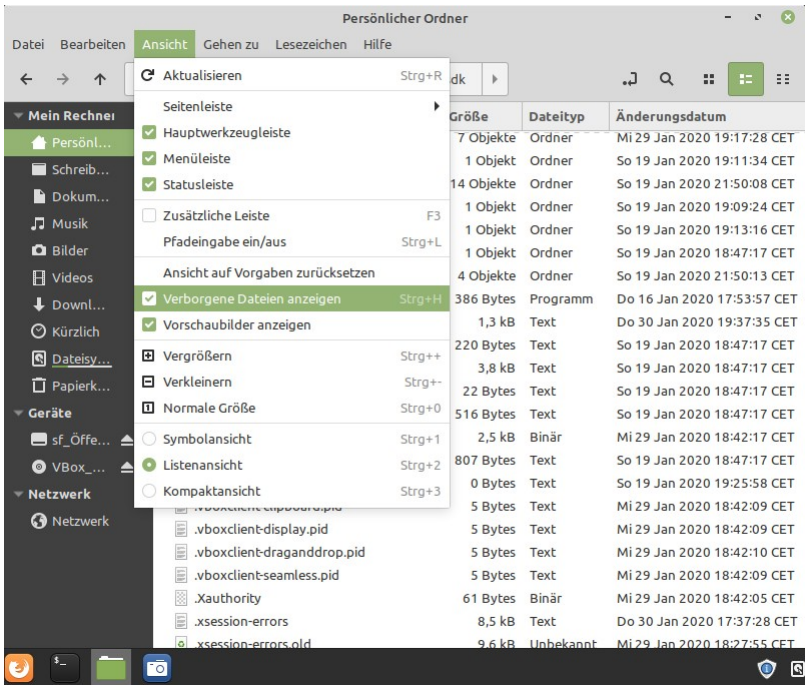
ln -sf "$HOME/Programme/android/sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.9" "$HOME/Programme/android/sdk/ndk-bundle/toolchains/mips64el-linux-android-4.9"

ln -sf "$HOME/Programme/android/sdk/ndk-bundle/toolchains/arm-linux-androideabi-4.9" "$HOME/Programme/android/sdk/ndk-bundle/toolchains/mipsel-linux-android-4.9"

So sollte es jetzt aussehen:



Im Dateimanager (Nemo, Nautilus) auf Ansicht und bei Verborgene Dateien ein Häckchen machen. Im Persönlichen Ordner findet man jetzt die Datei ./profile.



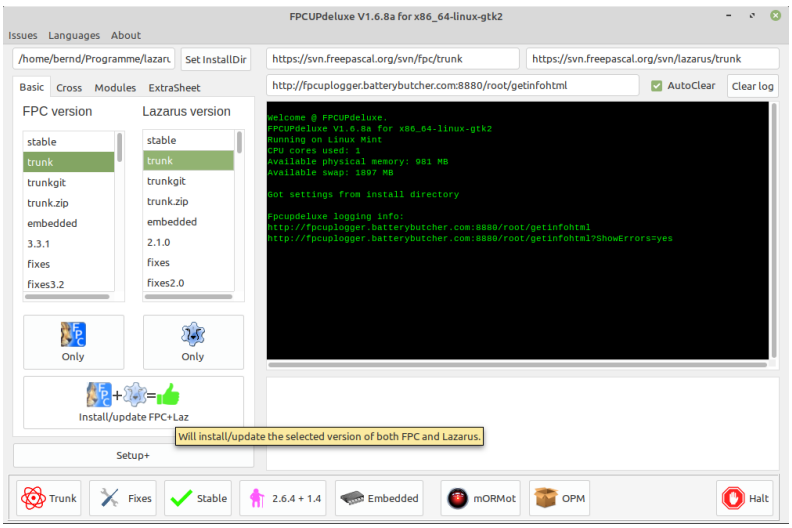
Die Datei ./profile mit rechter Maustaste anklicken und mit Textbearbeitung öffnen wählen. Ganz unten folgendes einfügen und speichern:

```
export ANDROID_SDK_ROOT="${HOME}/Programme/android/sdk"
export GRADLE_HOME="${HOME}/Programme/android/gradle-4.4.1"
export PATH="${PATH}:${HOME}/Programme/android/ndk-toolchain"
export PATH="${PATH}:${GRADLE_HOME}/bin"
```

Jetzt abmelden und wieder anmelden oder den PC neustarten.

Nun fpcupdeluxe starten. Ich empfehle für den Android Crosscompiler zusätzlich ein eigenes Lazarus zu installieren. Nach der Installation von Lamw wurden bei mir die Einstellungen der Lazarus IDE durcheinander gewürfelt.

Die Cross Compiler für arm und i386 lassen sich mit den stable Versionen installieren. Der Cross Compiler für aarch64 nur mit der trunk. Ich habe mich deshalb hier für die trunk Versionen entschieden. Achtung: das neue Lazarus in ein anderes Verzeichnis als das schon vorhandene installieren.



In Lazarus unter Tools, Options, General die Sprache auf deutsch umstellen. Lazarus neu starten.

Folgende Versionen wurden bei mir installiert:

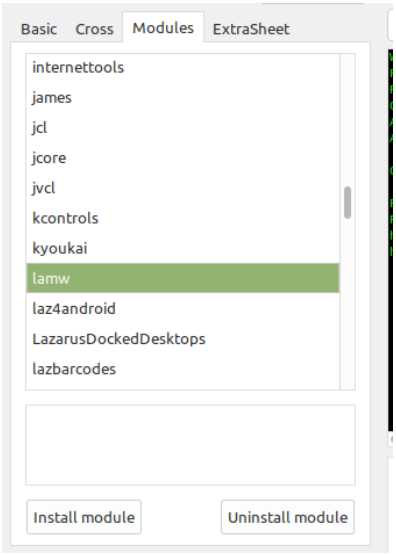
LAMW mit fpcupdeluxe installieren

Als nächstes noch lamw (falls noch nicht vorhanden) installieren:

LAMW Manager :"Ein Installationsprogramm zum Generieren der Lazarus IDE [und aller erforderlichen Komponenten!], angefertigt um für Android zu entwickeln!

zu lamw: https://wiki.lazarus.freepascal.org/LAMW

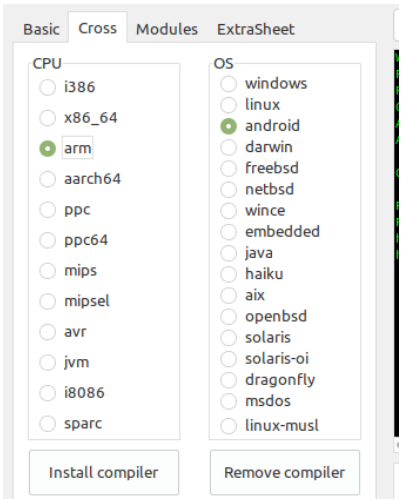
In fpcupdeluxe auf Modules gehen, dort:



Install module anklicken

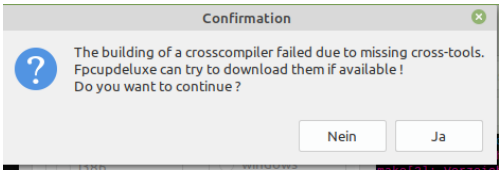
CrossCompiler für Android-arm installieren und einrichten

Jetzt den Cross Compiler installieren, dazu:



Auf Install compiler klicken

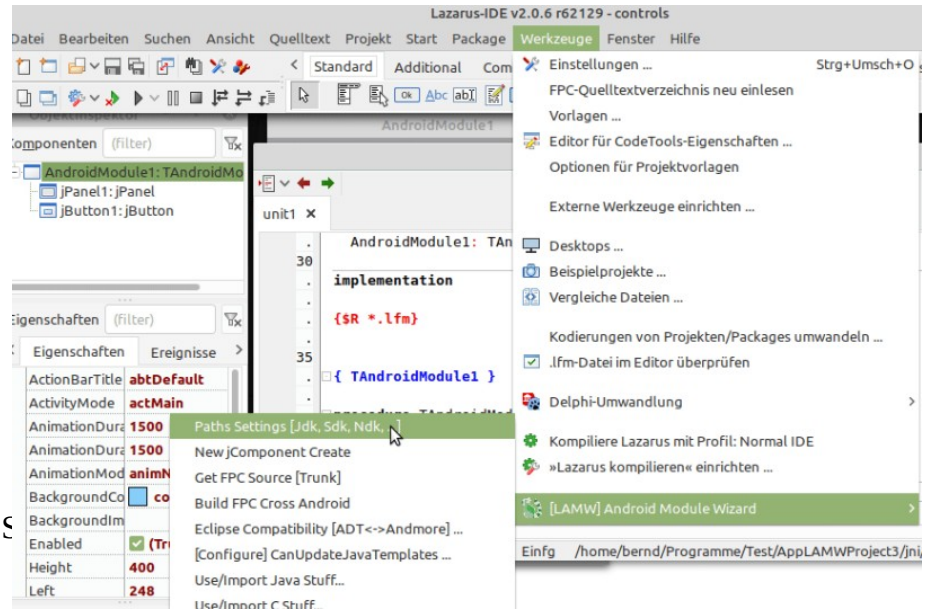
Zwischendurch kommt folgende Meldung:



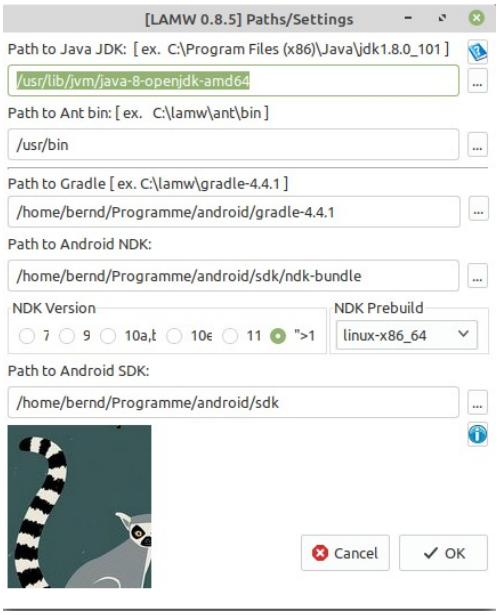
Mit Ja bestätigen. Jetzt werden die fehlenden Pakete geladen.

fpcupdeluxe beenden und Lazarus starten.

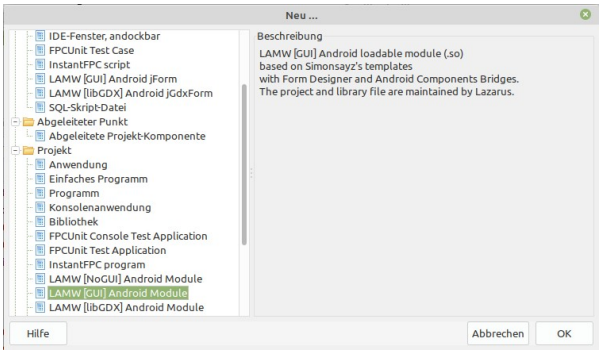
Nun Werkzeuge, [LAMW] Android Module Wizard, Paths Settings [Jdk,Sdk,Ndk]



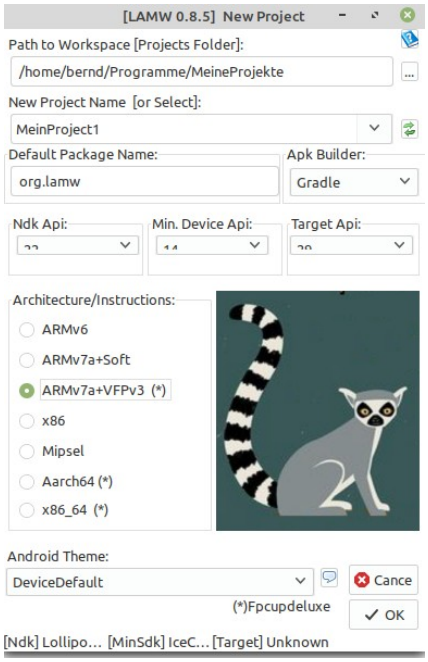
Folgende Einstellungen vornehmen:



Als nächstes Datei, Neu, LAMW[GUI]Android Module[nicht Android jForm], OK



Jetzt erscheint folgendes Fenster:



Zuerst den Pfad zu einem gewünschten Verzeichnis auswählen.

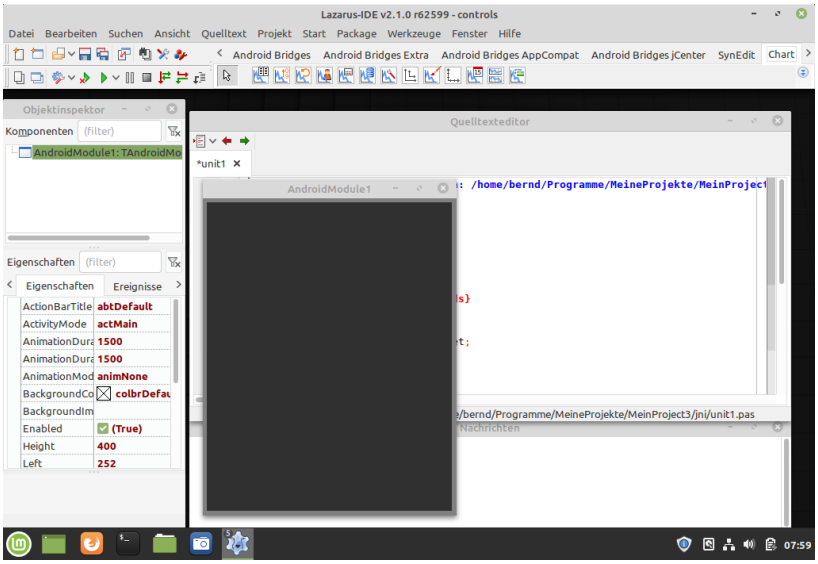
Dann einen Namen für die Anwendung eingeben.

Bei Apk Builder Ant (oder Gradle) auswählen. Ant ist schneller.

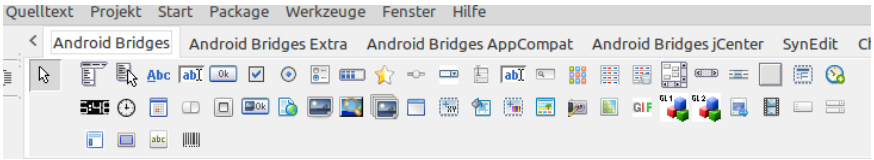
Bei Architecture ARMv7a+VFPv3 wählen. Den Rest einfach lassen.

Dann OK.

Jetzt sollte es etwa so aussehen:



Jetzt kann ein erstes kleines Testprogramm zusammen gestellt werden. Es befindet sich unter Android Bridges reichlich Auswahl für einen ersten Test.



Meine Wahl viel folgendermaßen aus:

Zuerst habe ich mal die Hintergrundfarbe geändert. Dann einen Button drauf gesetzt. Im OI folgendes ändern:

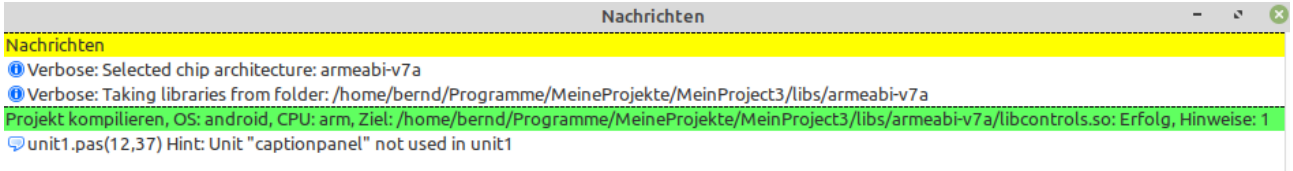
Jetzt noch ein TextView drauf gesetzt. PosRelativeToParent rpCenterin Parent. Text Hallo. Und die Hintergrundfarbe bzw, Textfarbe nach Wunsch angepasst.

Sieht dann so aus:

Mein Quelltext:

```
unit unit1;
{$mode delphi}
interface
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, AndroidWidget,  Laz_And_Controls;
type
  { TAndroidModule1 }
  TAndroidModule1 = class(jForm)
    jButton1: jButton;
    jTextView1: jTextView;
    procedure AndroidModule1Create(Sender: TObject);
    procedure jButton1Click(Sender: TObject);
  private
    {private declarations}
  public
    {public declarations}
  end;
var
  AndroidModule1: TAndroidModule1;
  fl            : byte;
implementation
{$R *.lfm}
{ TAndroidModule1 }
procedure TAndroidModule1.jButton1Click(Sender: TObject);
begin
 if fl=0 then
  begin
   jTextView1.Text:='Hurra!';
   fl:=1;
  end else
  begin
   jTextView1.Text:='Es ist geschafft!';
   fl:=0;
  end;
end;
procedure TAndroidModule1.AndroidModule1Create(Sender: TObject);
begin
 fl:=0;
end;
end.
```

Jetzt auf start, kompilieren (strg+F9). Und warten bis es fertig ist.



Nun muss die kompilierte Anwendung noch in ein Apk Paket verpackt werden damit es unter Android laufen kann.

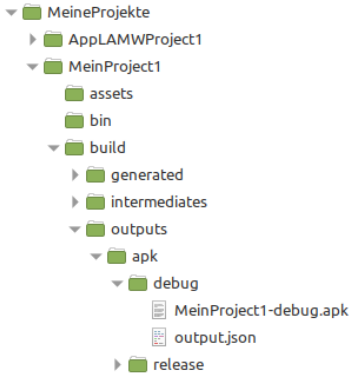
1. Variante mit Gradle:
Im Terminal ins Projektverzeichnis wechseln und dort gradle aufrufen.
Sieht bei mir dann so aus:
cd /home/bernd/Programme/MeineProjekte/MeinProject1

./gradle-local-build.sh

Jetzt startet gradle. Beim ersten Start dauert das ziemlich lang. Es werden noch einige fehlende Programme geladen. Falls es mal aussteigt einfach nochmal starten.
Am Ende sieht es dann so aus:

```
BUILD SUCCESSFUL in 11m 31s
57 actionable tasks: 56 executed, 1 up-to-date
bernd@bernd-VirtualBox:~/Programme/MeineProjekte/MeinProject1$
```

Das benötigte apk Paket findet man nun hier:



Ich musste die -debug Version verwenden. Die unter release hat nicht funktioniert.

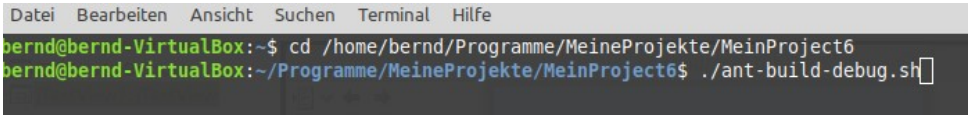
Nun die Datei MeinProject1-debug.apk zum Testen auf ein Androidgerät kopieren und dort installieren. In den Sicherheitseinstellungen muss dies natürlich erlaubt sein.
So sieht es auf einem alten Galaxy S4mini (Android Version 4.4.2)aus:

Auch noch erfolgreich getestet auf einem Tablet mit Android 5.5.1.

Variante2:

Das gleiche nochmal mit Ant anstelle von Gradle:

Im Terminal wieder ins Projektverzeichnis wechseln und dort ant mit dem Befehl ./ant-build-debug.sh aufrufen:



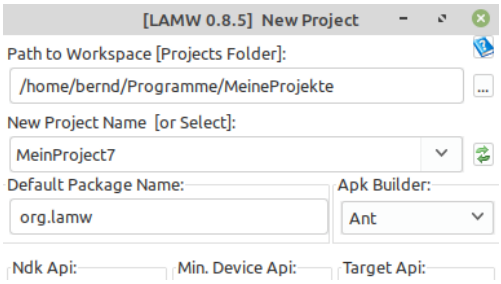
Folgendes Ergebnis:

35 Sekunden sind natürlich deutlich schneller. Ant ist wahrscheinlich deshalb die bessere Methode. Kenne den Unterschied aber leider nicht. Das apk befindet sich wie in der Ausgabe ersichtlich unter: /home/bernd/Programme/MeineProjekte/MeinProject6/bin/MeinProject6-debug.apk

Habe das Paket auf den gleichen Geräten wie oben mit Erfolg getestet.

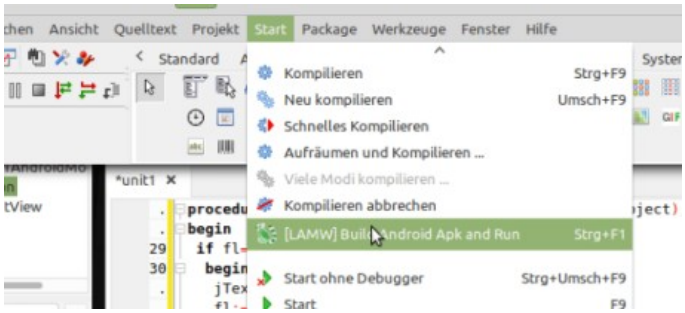
Variante3:

Aus der Lazarus IDE mit Strg+F1

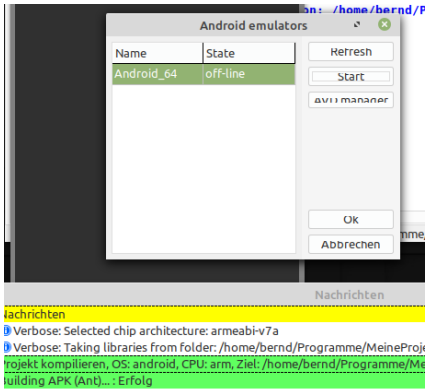


Hier den gewünschten Apk Builder wählen, der dann aufgerufen wird.

Am Ende nicht kompilieren klicken sondern Build Android Apk und Run (bei Lazarus in Virtualbox hängt sich Lazarus bei dieser Variante manchmal auf).



Da noch kein oder kein passender Android Emulator angelegt ist hier auf Abbrechen klicken.



Das Apk Paket befindet sich an den Orten wie unter Variante 1 und 2 beschrieben.

Das Apk Paket mit Adp auf ein Androidgerät automatisch installieren

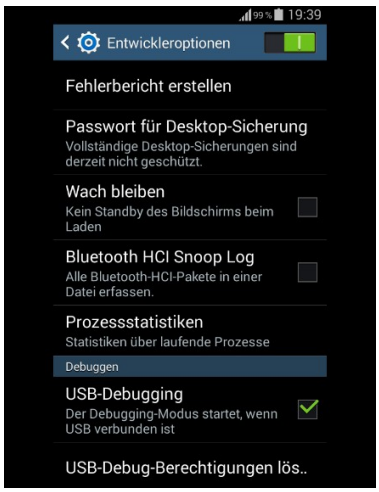
Damit nach „Build Android Apk und Run“ das Apk Paket automatisch auf ein Androidgerät übertragen wird muss natürlich erstens ein passendes Gerät mittels USB an den PC angeschlossen werden. Genutzt wir dann Adb (*Android*-Debug-Bridge), dies ist eine Schnittstelle zu Androidsystemen und in den bereits installierten SDK Platform Tools enthalten. Ich bin folgenden Quellen gefolgt:

`https://wiki.ubuntuusers.de/adb/` und

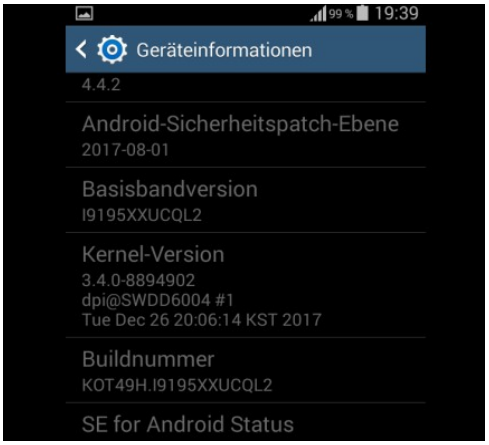
https://www.androidpit.de/adb-treiber-android-windows

Also als erstes muss auf dem Android Gerät USB-Debugging aktiviert sein. Das geschieht in den Entwickleroptionen.

Menü/Einstellungen/Optionen



Sind die Entwickleroptionen nicht sichtbar muss man zuerst zu der Buildnummer navigieren und diese 7mal antippen. Anschließend sind die Entwickleroptionen wie oben beschrieben sichtbar.



Ich musste jetzt noch in der USB-PC Verbindung auf **Kamera (PTP) umstellen** damit ich übertragen konnte!

Man muss mit den Finger

von oben nach unten ziehen,

und dann auf Weitere USB-Optionen.

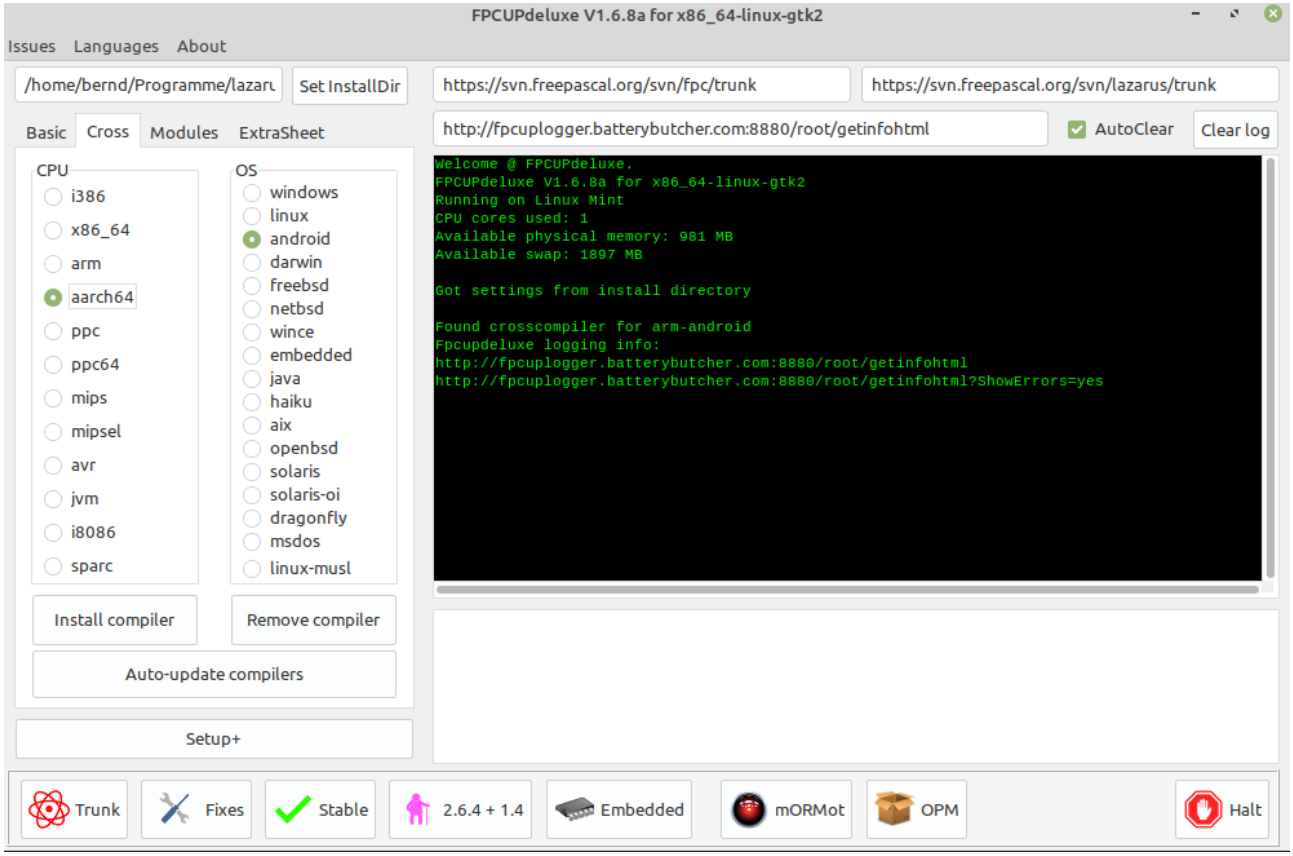
Jetzt das Terminal öffnen und adb devices eingeben. Kommt so was:

List of devices attached
* daemon not running; starting now at tcp:5037
* daemon started successfully
0049c50a device

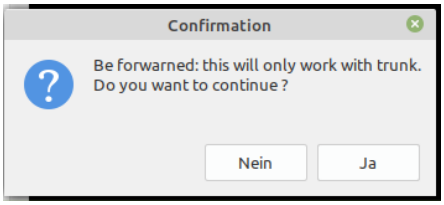
ist alles gut. Kommt irgendwas (beim erstenmal) mit unauthorized dann schaut mal auf das Android Gerät dort müsst ihr die Verbindung zulassen!

Jetzt in Lazarus mit „Build Android Apk und Run“ kompilieren. Das Apk wird übertragen und installiert.

CrossCompiler für Android-aarch64 installieren und einrichten

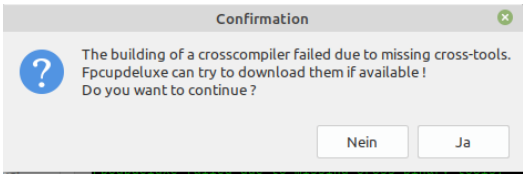


Die folgende Meldung mit ja bestätigen.

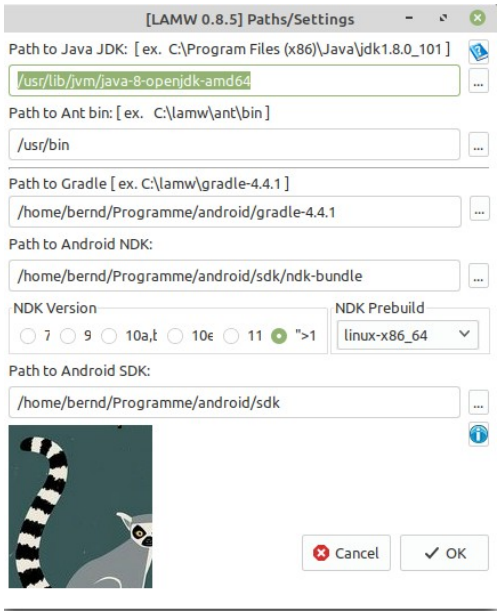


Zwischendurch kommt diese Meldung:

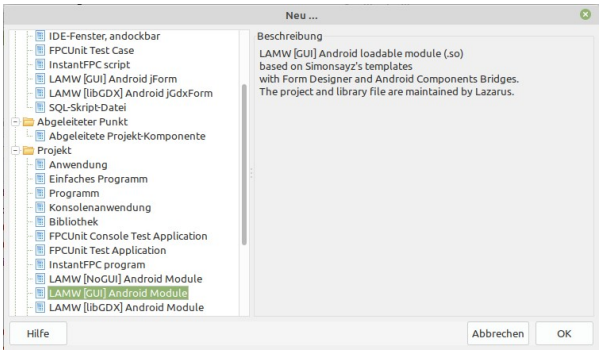
mit ja bestätigen.



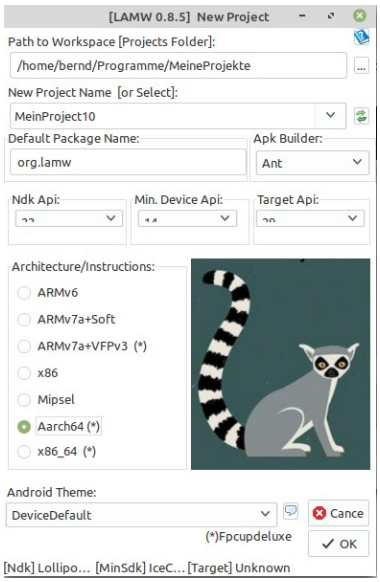
Folgende Einstellungen vornehmen (identisch bei allen Android):



Als nächstes Datei, Neu, LAMW[GUI]Android Module[nicht Android jForm], OK



Jetzt erscheint folgendes Fenster:



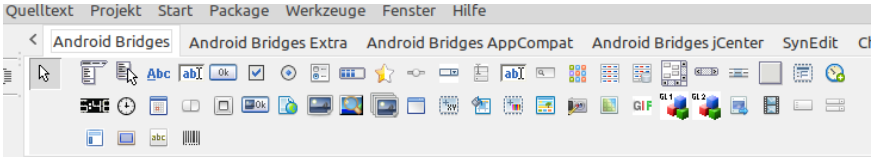
Zuerst den Pfad zu einem gewünschten Verzeichnis auswählen.

Dann einen Namen für die Anwendung eingeben.

Architecture: Aarch64 wählen

Rest lassen, OK

Jetzt kann ein erstes kleines Testprogramm zusammen gestellt werden. Es befindet sich unter Android Bridges reichlich Auswahl für einen ersten Test.



Meine Wahl viel folgendermaßen aus:

Zuerst habe ich mal die Hintergrundfarbe geändert. Dann einen Button drauf gesetzt. Im OI folgendes ändern:

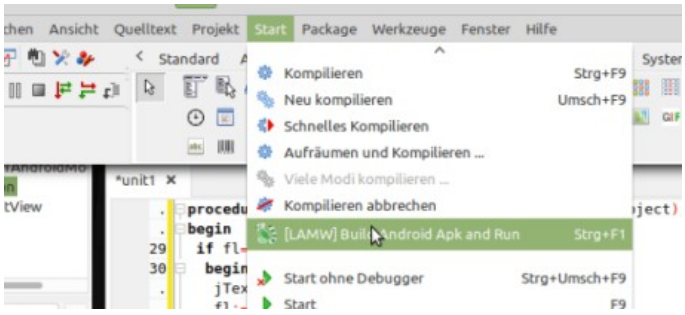
Jetzt noch ein TextView drauf gesetzt. PosRelativeToParent rpCenterin Parent. Text Hallo. Und die Hintergrundfarbe bzw, Textfarbe nach Wunsch angepasst.

Sieht dann so aus:

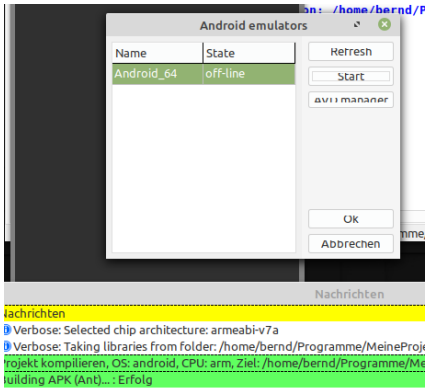
Mein Quelltext:

```
unit unit1;
{$mode delphi}
interface
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, AndroidWidget,  Laz_And_Controls;
type
  { TAndroidModule1 }
  TAndroidModule1 = class(jForm)
    jButton1: jButton;
    jTextView1: jTextView;
    procedure AndroidModule1Create(Sender: TObject);
    procedure jButton1Click(Sender: TObject);
  private
    {private declarations}
  public
    {public declarations}
  end;
var
  AndroidModule1: TAndroidModule1;
  fl            : byte;
implementation
{$R *.lfm}
{ TAndroidModule1 }
procedure TAndroidModule1.jButton1Click(Sender: TObject);
begin
 if fl=0 then
  begin
   jTextView1.Text:='Hurra!';
   fl:=1;
  end else
  begin
   jTextView1.Text:='Es ist geschafft!';
   fl:=0;
  end;
end;
procedure TAndroidModule1.AndroidModule1Create(Sender: TObject);
begin
 fl:=0;
end;
end.
```

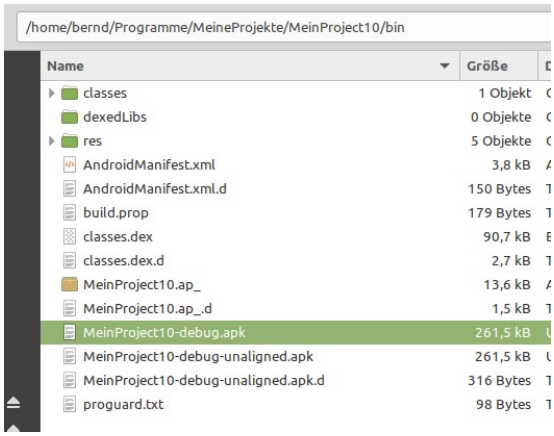
Am Ende nicht kompilieren klicken sondern Build Android Apk und Run.



Da noch kein oder kein passender Android Emulator angelegt ist hier auf Abbrechen klicken.



Dann befindet sich hier das apk Paket:



Leider besitze ich kein Androidgerät mit Aarch64. Deshalb benötige ich zum Testen ein virtuelles Gerät.

Einrichten eines virtuellen Android Gerätes für Aarch64

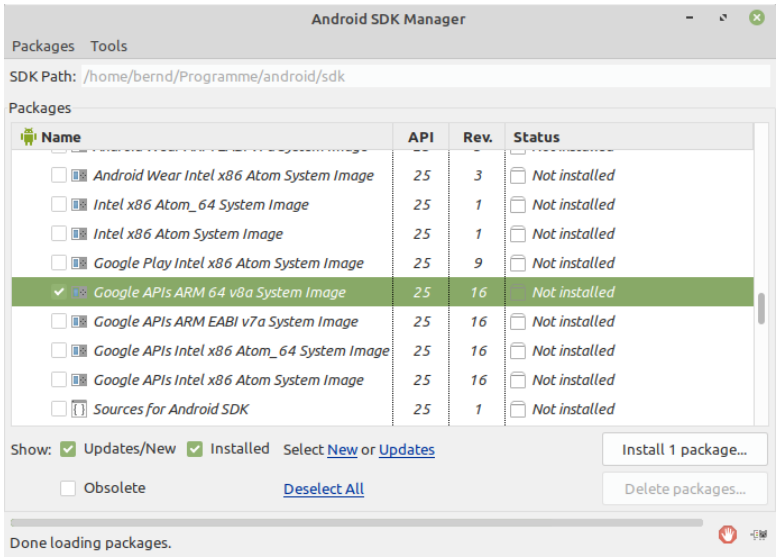
Zuerst benötigt man ein passendes System Image. Nach Recherche im Internet denke ich das Aarch64 und arm64 im Prinzip das gleiche ist. Im SDK Manager ist die letzte Version mit arm64 die Version 25.

Ins Terminal wechseln und ins Verzeichnis tools wechseln.

cd /home/bernd/Programme/android/sdk/tools dann

./android

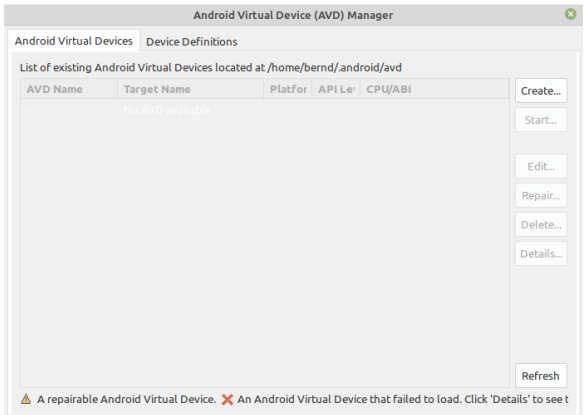
eingeben. Jetzt startet der SDK-Manager. Die vorgewählten Häkchen bei Android10 entfernen. Dann zur Version Android 7.7.1 (API25) gehen und dort das ARM64 Image auswählen.



Install 1 package… klicken, dann accept License und install

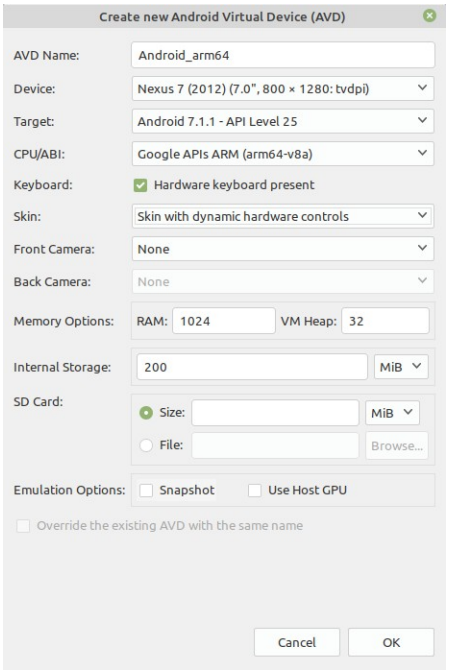
Es gibt wohl mehrere möglichkeiten virtuelle Geräte anzulegen. Ich beschreibe hier den Weg mit dem AVD Manager von Google. Leider läuft diese Variante nicht in Virtualbox!

Wenn das Image fertig installiert ist im Android SDK Manager oben auf Tools klicken und Manage AVDs… auswählen. Jetzt startet der Android Virtual Device Manager. In diesem können virtuelle Geräte wie zum Beispiel Handys konfiguriert werden.



Nun auf Create…

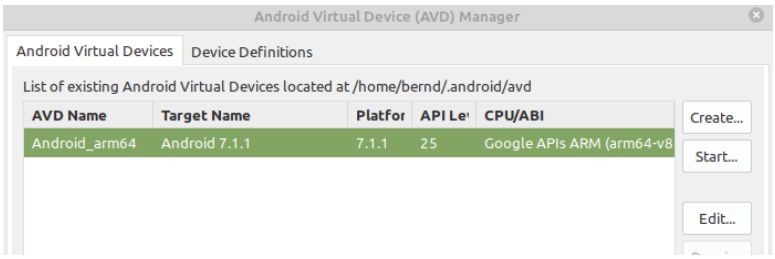
Ich habe mich für diese Einstellungen entschieden:



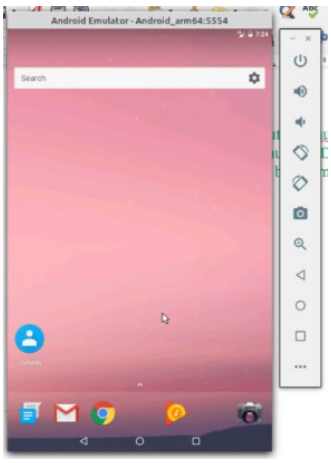
Wer kann hier mehr RAM eintragen.

Dann OK und beim nächsten Fenster wieder OK.

Jetzt wird im AVD Manager das neue Gerät Android_64 angezeigt. Einmal drauf klicken, jetzt wird die Seitenleiste aktiv und dann auf Start…



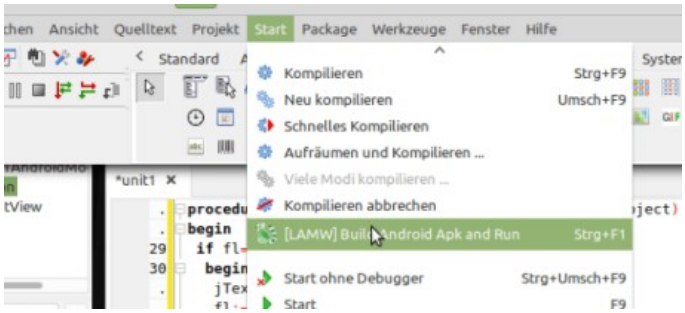
Das Fenster LaunchOptions mit den Button Launch bestätigen. Jetzt startet der Android Emulator. Dauert leider ziemlich lange, bei mir waren es bestimmt 40 Minuten!



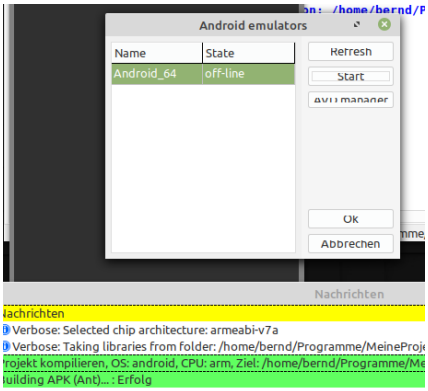
Den Emulator durch anklicken des kleinen x schließen.

Damit in Lazarus ein Paket übertragen wird muss man den Emulator aus der Lazarus IDE starten.

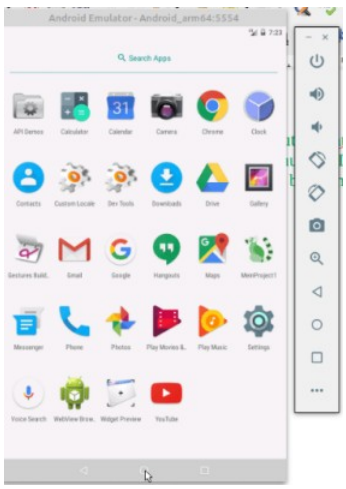
Dazu nicht kompilieren klicken sondern Build Android Apk und Run.

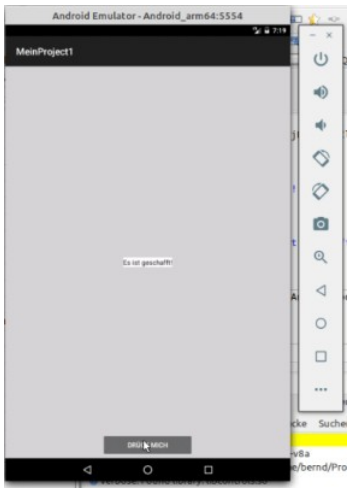


Jetzt falls mehrere Geräte angelegt sind das Richtige anwählen und Start klicken. In diesem Fenster kann auch der AVD Manager gestartet werden um (weitere) Geräte anzulegen.



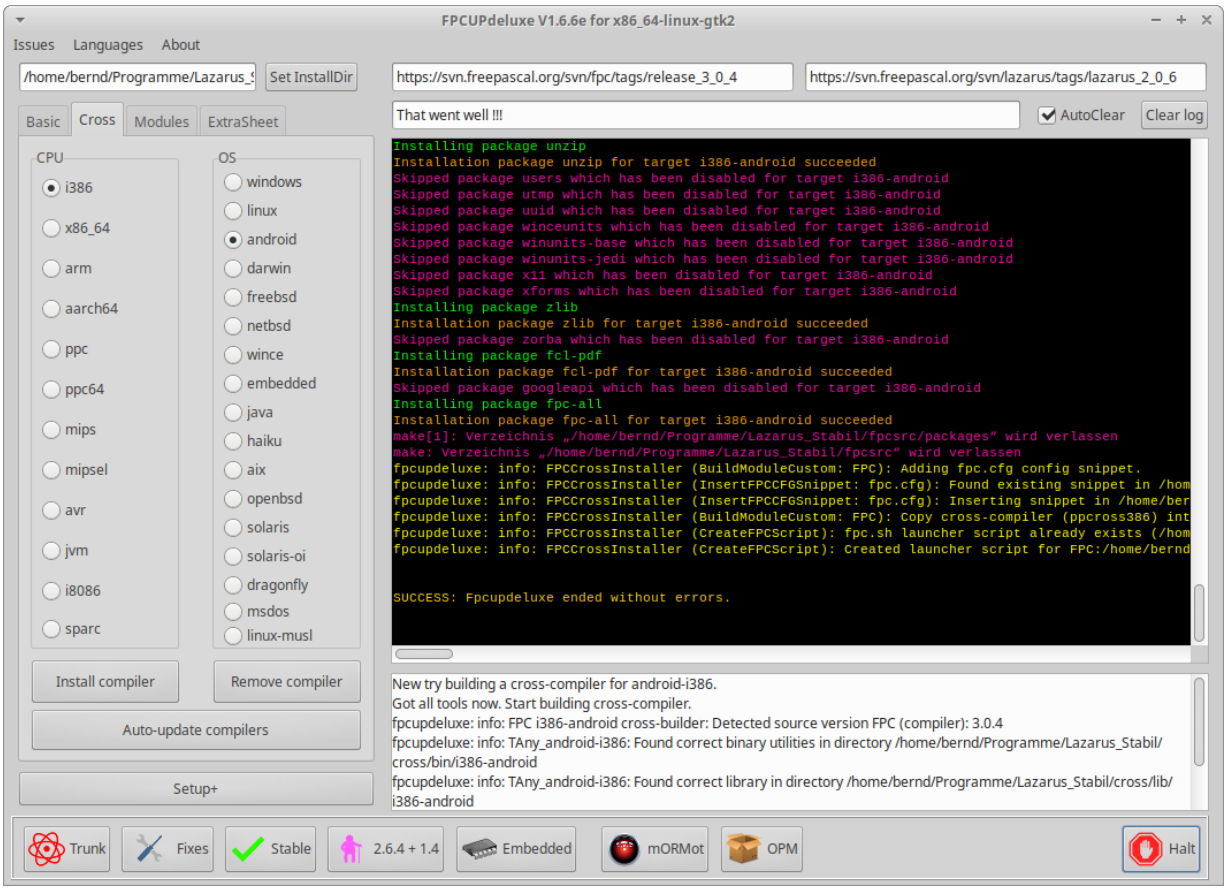
Nun dauerte es bei mir wieder ca. 40 Minuten bis das Apk erfolgreich übertragen und installiert war.



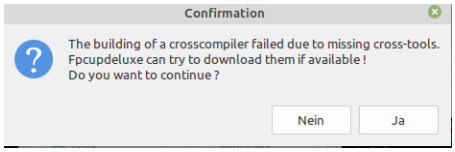


Fazit: Es funktioniert, ist aber nicht praktikabel.

CrossCompiler für Android-x86 installieren und einrichten



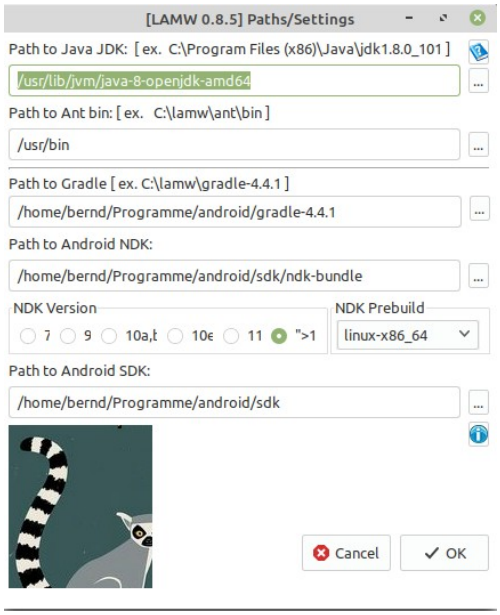
Obige Einstellung wählen und **Install Compiler** anklicken. Zwischendurch kam bei mir eine Abfrage ob fehlende Pakete vom Internet geladen werden sollen. Dies habe ich mit ja bestätigt.



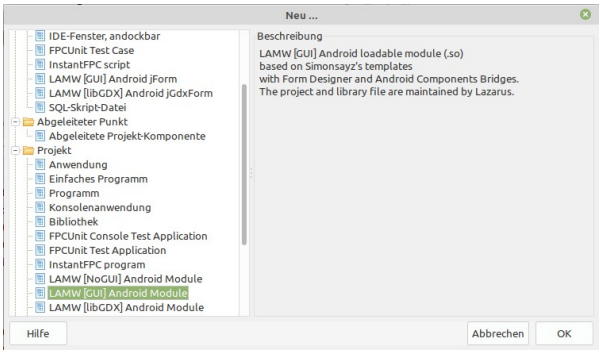
Die fehlenden Pakete werden nun geladen und dann lief alles ohne Probleme durch.

fpcupdeluxe beenden und Lazarus starten.

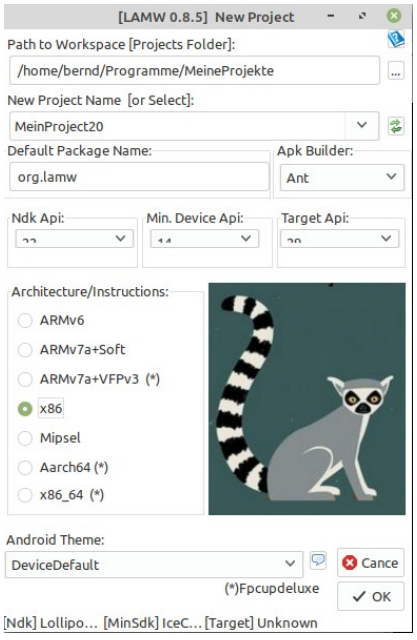
Folgende Einstellungen vornehmen (identisch bei allen Android):



Als nächstes Datei, Neu, LAMW[GUI]Android Module[nicht Android jForm], OK



Jetzt erscheint folgendes Fenster:



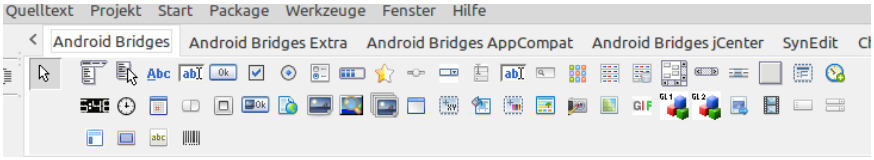
Zuerst den Pfad zu einem gewünschten Verzeichnis auswählen.

Dann einen Namen für die Anwendung eingeben.

Architecture: X86 wählen

Rest lassen, OK

Jetzt kann ein erstes kleines Testprogramm zusammen gestellt werden. Es befindet sich unter Android Bridges reichlich Auswahl für einen ersten Test.



Meine Wahl viel folgendermaßen aus:

Zuerst habe ich mal die Hintergrundfarbe geändert. Dann einen Button drauf gesetzt. Im OI folgendes ändern:

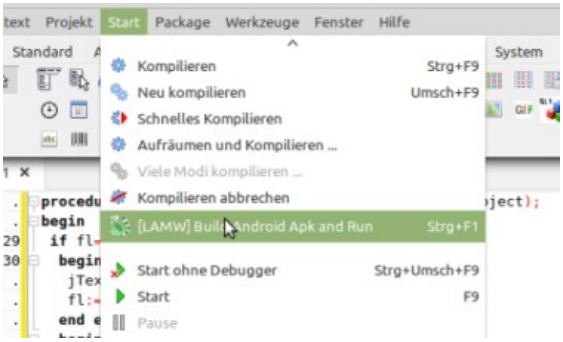
Jetzt noch ein TextView drauf gesetzt. PosRelativeToParent rpCenterin Parent. Text Hallo. Und die Hintergrundfarbe bzw, Textfarbe nach Wunsch angepasst.

Sieht dann so aus:

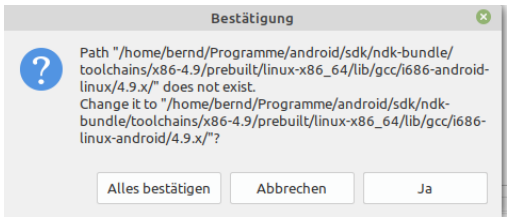
Mein Quelltext:

```
unit unit1;
{$mode delphi}
interface
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, AndroidWidget,  Laz_And_Controls;
type
  { TAndroidModule1 }
  TAndroidModule1 = class(jForm)
    jButton1: jButton;
    jTextView1: jTextView;
    procedure AndroidModule1Create(Sender: TObject);
    procedure jButton1Click(Sender: TObject);
  private
    {private declarations}
  public
    {public declarations}
  end;
var
  AndroidModule1: TAndroidModule1;
  fl            : byte;
implementation
{$R *.lfm}
{ TAndroidModule1 }
procedure TAndroidModule1.jButton1Click(Sender: TObject);
begin
 if fl=0 then
  begin
   jTextView1.Text:='Hurra!';
   fl:=1;
  end else
  begin
   jTextView1.Text:='Es ist geschafft!';
   fl:=0;
  end;
end;
procedure TAndroidModule1.AndroidModule1Create(Sender: TObject);
begin
 fl:=0;
end;
end.
```

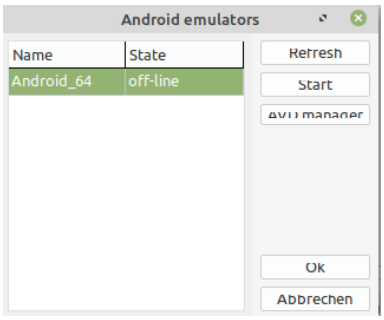
Jetzt auf:



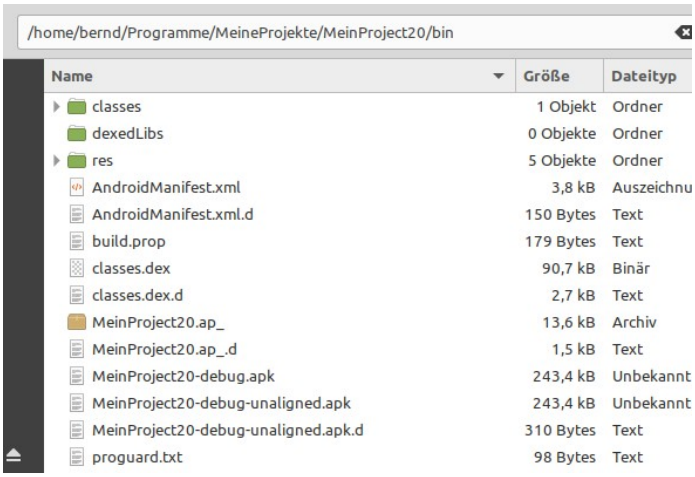
dann



mit Ja bestätigen. Es startet folgendes Fenster:



Ist kein Gerät oder kein passendes Gerät vorhanden kann man jetzt auf Abbrechen klicken. Das apk Paket befindet sich dann hier:

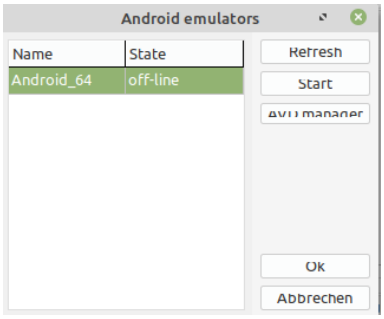


Die Datei MeinProjekt20-debug.apk auf ein Android Gerät mit x86 kopieren und installieren.

Möchte man ein Virtuelles Gerät anlegen dann sollte man so weiter machen:

Anlegen eines Android Emulators für Android x86

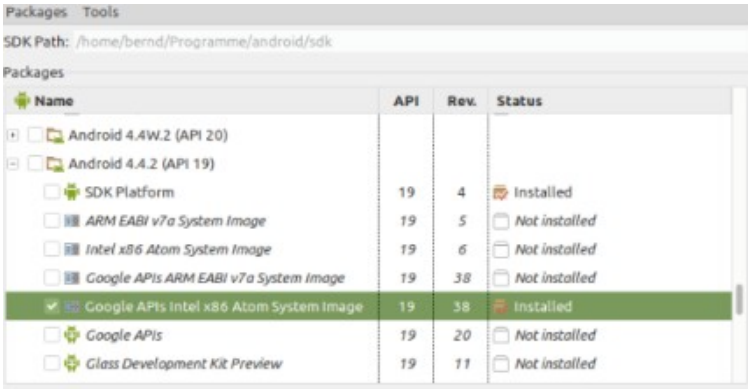
Es startet folgendes Fenster:



Da noch kein passender Android Emulator vorhanden auf AVD Manager klicken. Jetzt benötigt man zu erst ein passendes System Image. Deshalb auf Tool und Manage SDK.

Im SDK Manager hab ich mich für das Google_x86 Image der Android 4.4.2 Version entschieden.

Auf Install 1 package… klicken



Wenn alles fertig ist SDK Manager schließen und im AVD Manager auf Create klicken.

Ich habe folgendes Gerät angelegt:

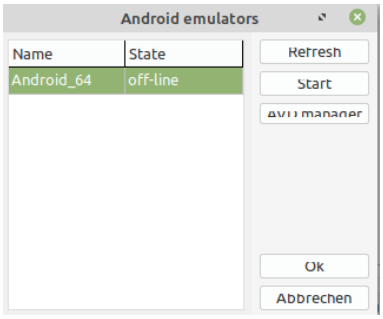
Zuletzt auf Ok und noch mal Ok

Nun ist das neue Gerät im AVD Manager sichtbar, dort anklicken und auf Start klicken.

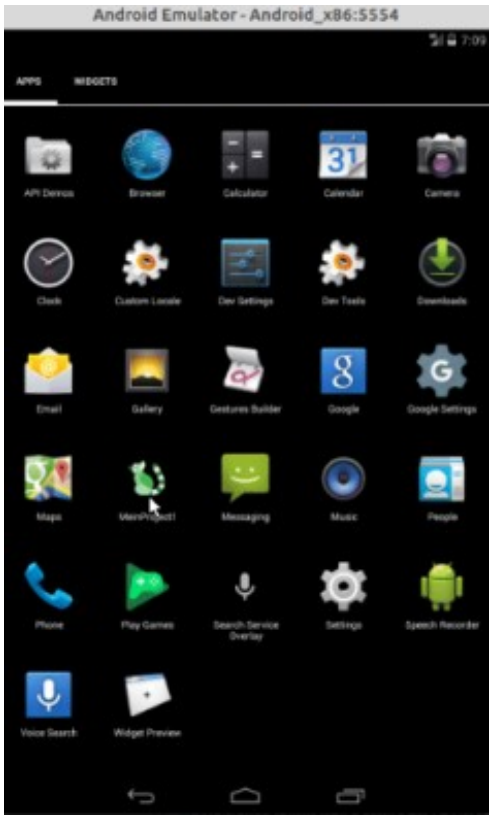
Das Launchfenster mit Launch bestätigen.

Warten bis der Emulator komplett gestartet ist (dauert bei mir ca. 3 Minuten):

Dann hier auf OK drücken:



Jetzt wird das Apk Paket übertragen und installiert!



Damit ein neues oder geändertes Projekt übertragen werden kann muss der Emulator geschlossen werden (kleines x). Ab den zweiten Mal verkürzte sich die Zeit auf eine gute Minute.

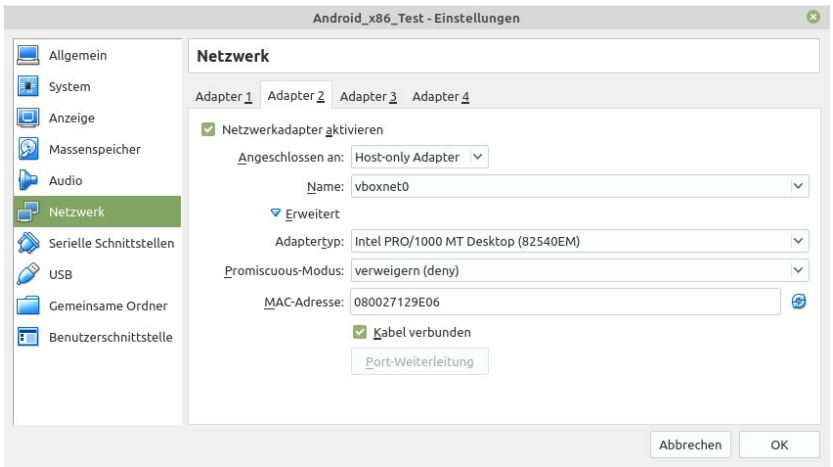
Programme für Android-x86 in Virtualbox testen

Zuerst benötigt man ein ISO Systemimage von einer Android x86 Version.

Hier wird man fündig: https://www.osboxes.org/android-x86/ oder auch hier: https://code.google.com/archive/p/android-x86/downloads
Bei mir lief diese Version: Android-x86 6.0 Marshmallow (R3) (aber nicht sehr schnell) deshalb habe ich mich für diese Version entschieden: android-x86-4.0-r1-thinkpad.iso (lief richtig flüssig)

Wie Android in einer Virtualbox installiert wird steht zum Beispiel hier: https://www.giga.de/apps/android/tipps/android-in-virtualbox-installieren-anleitung-schritt-fuer-schritt/ Das Image von obiger Adresse muss nicht installiert werden sondern wird einfach angewählt.

Nachdem Android in der Virtualbox installiert ist und läuft, Android herunter fahren (nicht ausschalten). In den Einstellungen unter Netzwerk Host-only Adapter einstellen.



Nun kann man mit adb Apk-Pakete in die Virtualbox schicken. Dies funktioniert auch wenn die Lazarusversion ebenfalls in einer Virtualbox läuft. Adp ist bereits in der Android SDK enthalten.

Quelle: https://wiki.ubuntuusers.de/adb/ und

https://ketnoinhau.wordpress.com/2014/10/26/copy-and-install-files-apk-from-pc-to-virtualbox-android-machine/ und

https://www.androidpit.de/adb-treiber-android-windows

Folgende Vorgehensweise hat bei mir funktioniert (kann man natürlich machen wie man möchte). Das erzeugte Apk-Paket direkt in den Persönlichen Ordner kopieren (Pfade werden dadurch kürzer). Das Apk Paket umbenennen, damit der Name nicht so lange ist, zum Beispiel in Test.apk. (Die entpackte Anwendung in Android besitzt aber trotzdem den ursprünglichen Namen!)

Jetzt Android in der Virtualbox starten und warten bis es komplett hochgefahren ist. Nun im Linux das Terminal öffnen und folgendes eingeben:

adb connect 192.168.56.101

Kommt eine Verbindung zustande kommt folgende Meldung:

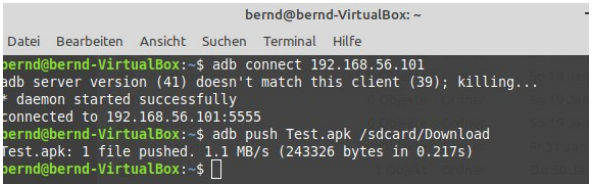
connected to 192.168.56.101:5555

War die Verbindung erfolgreich dann folgendes eingeben:

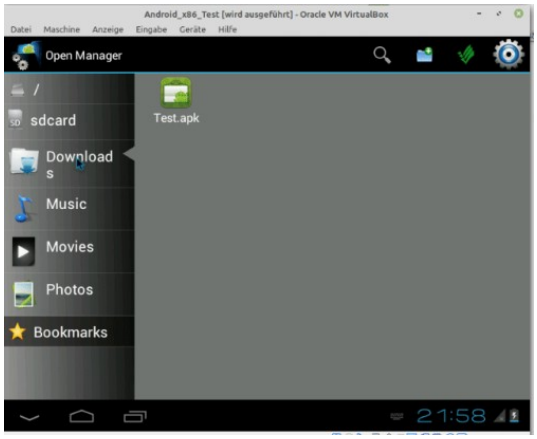
adb push Test.apk */sdcard/*Download

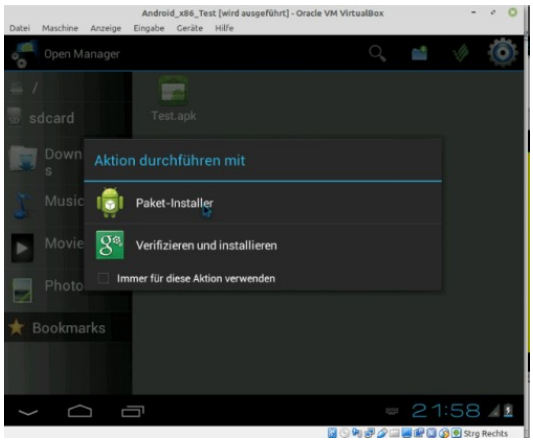
Wurde das Paket erfolgreich übertragen kommt folgende Meldung:

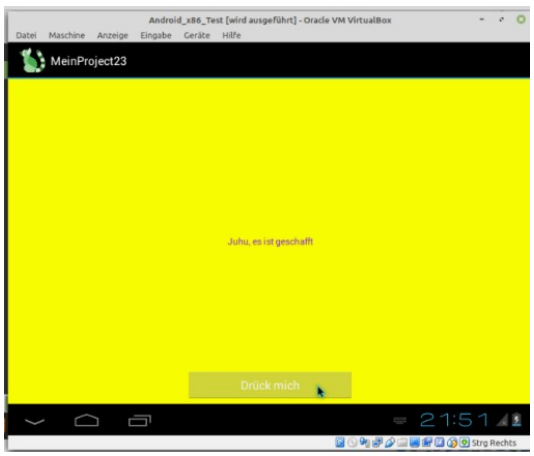
Test.apk: 1 file pushed. 1.4 MB/s (243403 bytes in 0.164s)



Die Datei befindet sich nun im Ordner Downloads des Android-x86. Apk installieren und öffnen.



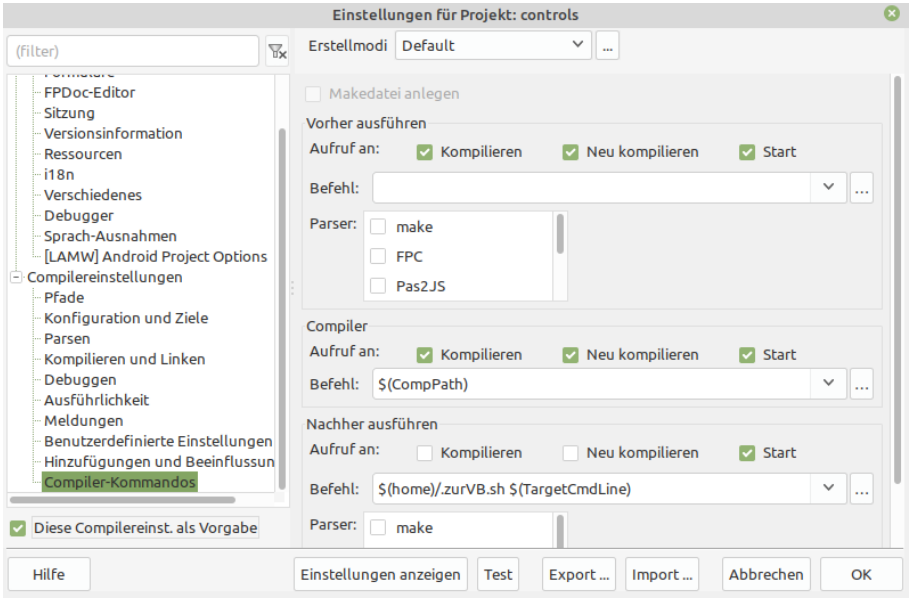




Wer **nur das Senden** (wer gleich installieren möchte weiter unten lesen) der Apk in die Virtualbox automatisieren möchte kann das **zum Beispiel** so machen:
Zuerst einen Texteditor öffnen und das folgende Bashscript einfügen:

```
#!/bin/bash
cd ..
./ant-build-debug.sh
adb connect 192.168.56.101
cd bin
adb push $(find -name "*-debug.apk") sdcard/Download
```

Jetzt das Script mit folgenden Dateinamen im Persönlichen Ordner speichern: .zurVB.sh
Der . vor dem Dateinamen macht die Datei unsichtbar. Die Datei muss eventuell ausführbar gemacht werden! Nun in der Lazarus IDE unter Projekt, Projekt Einstellungen folgendes einstellen:



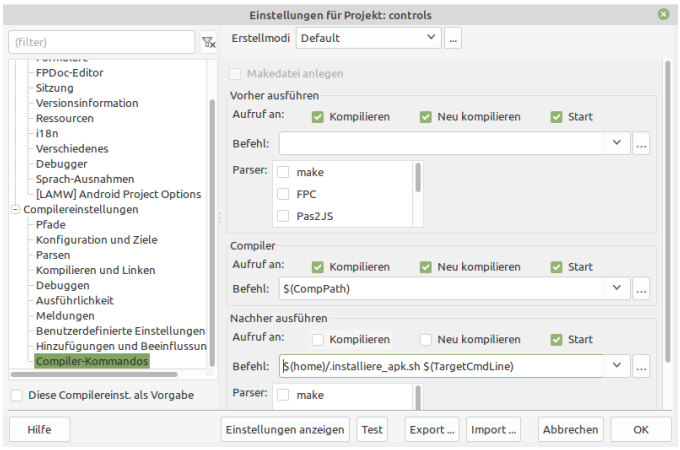
Bei Nachher ausführen nur ein Haken bei Start, bei Befehl
$(home)/.zurVB.sh $(TargetCmdLine)

Zuerst das Android in der Virtualbox starten! Kompiliert man nun mit Start ohne Debugger (Umsch+Strg+F9) wird kompiliert, das Apk mit Ant erzeugt und das Apk in die Virtualbox gesendet. Es befindet sich im Ordner sdcard/Download. Das Installieren im Android muss man natürlich dann noch machen.

Wer die Apk in die Virtualbox senden und gleich installieren möchte kann dies **zum Beispiel** so machen:
Zuerst einen Texteditor öffnen und das folgende Bashscript einfügen:

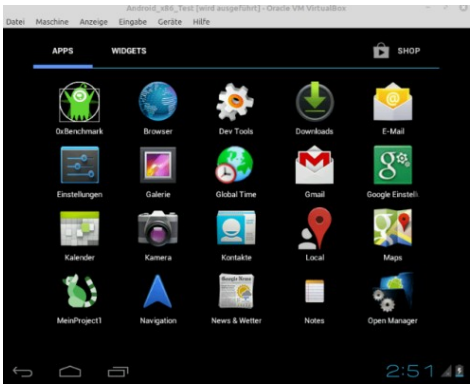
```
#!/bin/bash
cd ..
./ant-build-debug.sh
adb connect 192.168.56.101
cd bin
adb install $(find -name "*-debug.apk")
```

Jetzt das Script mit folgenden Dateinamen im Persönlichen Ordner speichern: .installiere_apk.sh
Der . vor dem Dateinamen macht die Datei unsichtbar. Die Datei muss eventuell ausführbar gemacht werden! Nun in der Lazarus IDE unter Projekt, Projekt Einstellungen folgendes einstellen:



Bei Nachher ausführen nur ein Haken bei Start, bei Befehl
$(home)/.installiere_apk.sh $(TargetCmdLine)

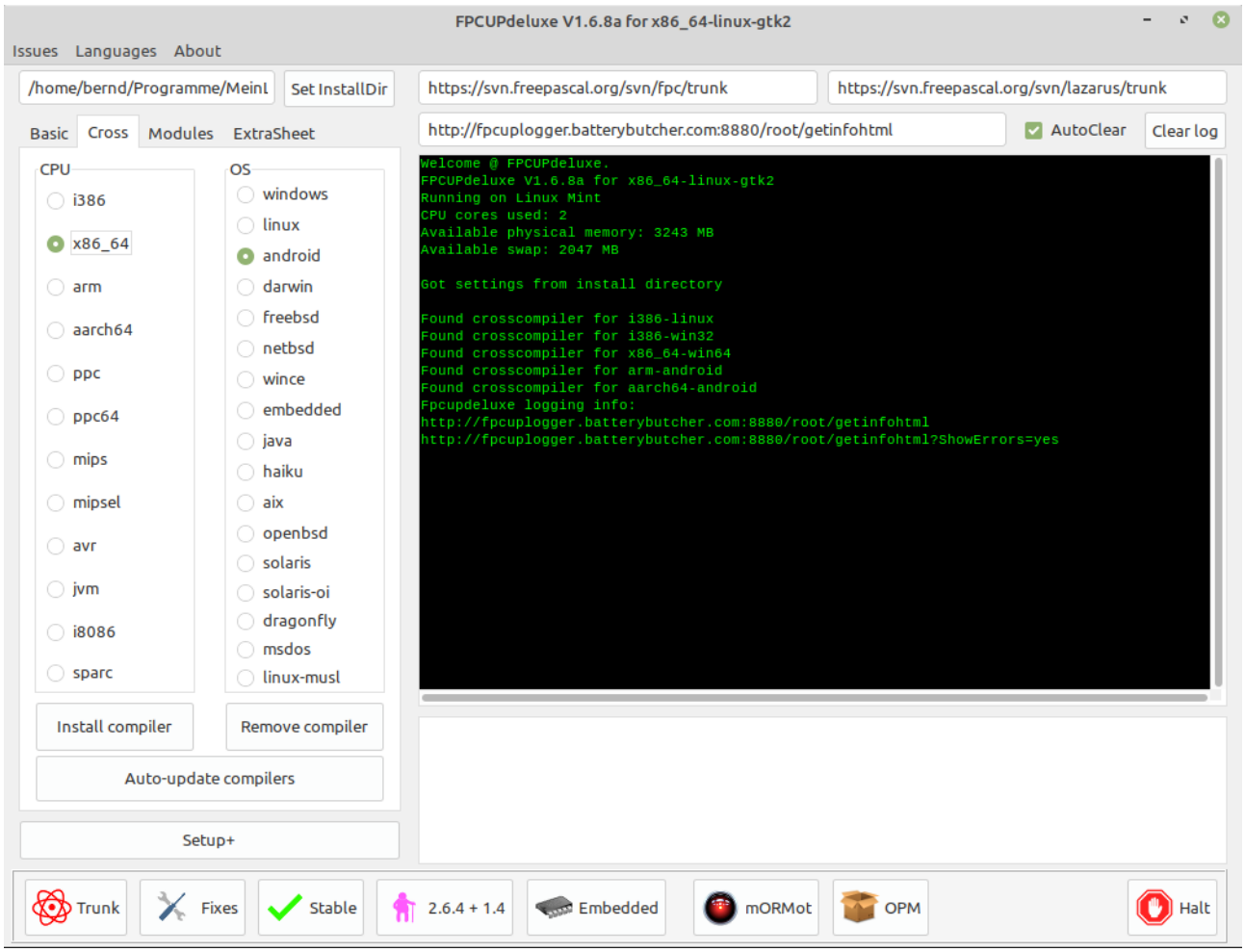
Zuerst das Android in der Virtualbox starten! Kompiliert man nun mit Start ohne Debugger (Umsch+Strg+F9) wird kompiliert, das Apk mit Ant erzeugt und das Apk in die Virtualbox gesendet und gleich installiert. Es kann durch anklicken gestartet werden.



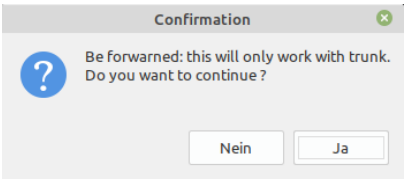
Funktioniert sehr schnell!!!!

Leider habe ich es nicht geschafft die IDE Einstellungen zu speichern. Beim Anlegen eines neuen LAMW-Modules sind die Einstellungen wieder weg. Meine Lösung ist das Exportieren der IDE Einstellung in eine Datei (siehe: Einstellungen der IDE sichern). Beim Erstellen einer neuen Anwendung müssen dann nur die Einstellungen importiert werden.

CrossCompiler für Android-x86_64 installieren und einrichten

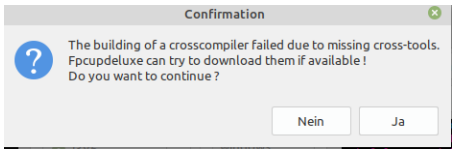


Obige Einstellung wählen und **Install Compiler** anklicken.



Mit Ja bestätigen.

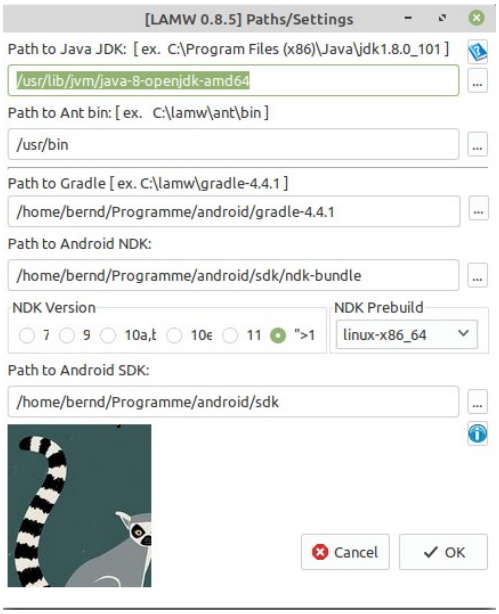
Zwischendurch kam bei mir eine Abfrage ob fehlende Pakete vom Internet geladen werden sollen. Dies habe ich mit ja bestätigt.



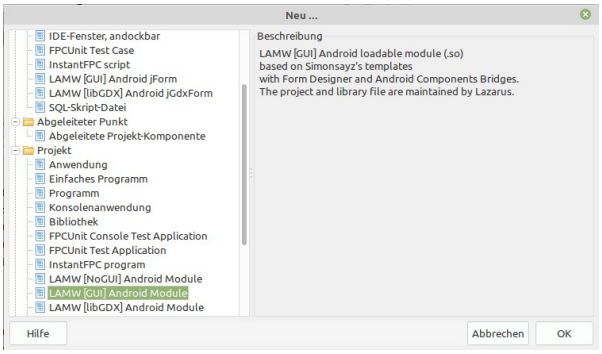
Die fehlenden Pakete werden nun geladen und dann lief alles ohne Probleme durch.

fpcupdeluxe beenden und Lazarus starten.

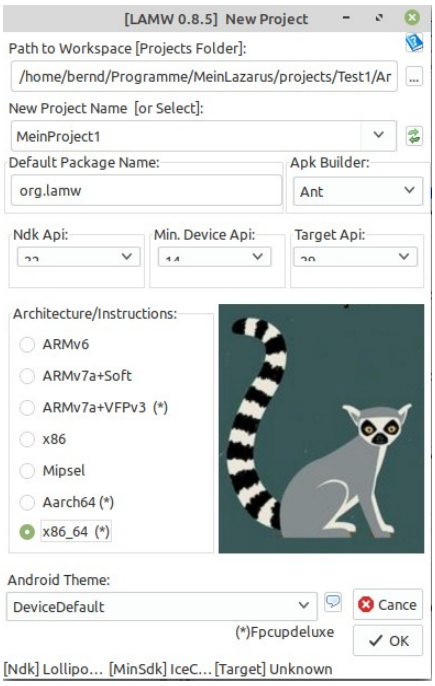
Folgende Einstellungen vornehmen (ist bei allen Android gleich):



Als nächstes Datei, Neu, LAMW[GUI]Android Module [nicht Android jForm], OK



Jetzt erscheint folgendes Fenster:



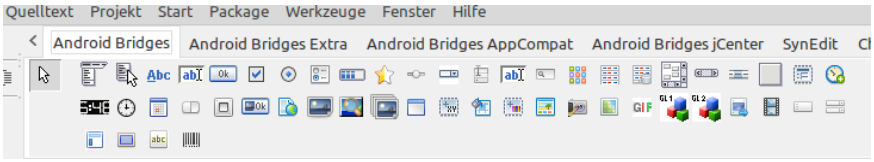
Zuerst den Pfad zu einem gewünschten Verzeichnis auswählen.

Dann einen Namen für die Anwendung eingeben.

Architecture: X86-64 wählen

Rest lassen, OK

Jetzt kann ein erstes kleines Testprogramm zusammen gestellt werden. Es befindet sich unter Android Bridges reichlich Auswahl für einen ersten Test.



Meine Wahl viel folgendermaßen aus:

Zuerst habe ich mal die Hintergrundfarbe geändert. Dann einen Button drauf gesetzt. Im OI folgendes ändern:

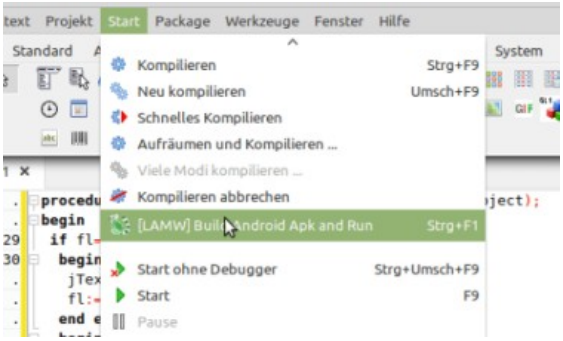
Jetzt noch ein TextView drauf gesetzt. PosRelativeToParent rpCenterin Parent. Text Hallo. Und die Hintergrundfarbe bzw, Textfarbe nach Wunsch angepasst.

Sieht dann so aus:

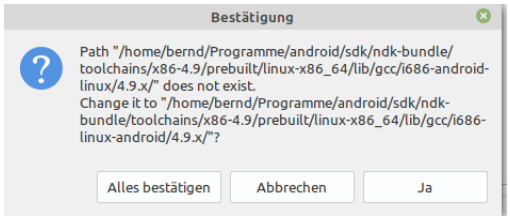
Mein Quelltext:

```
unit unit1;
{$mode delphi}
interface
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, AndroidWidget,  Laz_And_Controls;
type
  { TAndroidModule1 }
  TAndroidModule1 = class(jForm)
    jButton1: jButton;
    jTextView1: jTextView;
    procedure AndroidModule1Create(Sender: TObject);
    procedure jButton1Click(Sender: TObject);
  private
    {private declarations}
  public
    {public declarations}
  end;
var
  AndroidModule1: TAndroidModule1;
  fl            : byte;
implementation
{$R *.lfm}
{ TAndroidModule1 }
procedure TAndroidModule1.jButton1Click(Sender: TObject);
begin
 if fl=0 then
  begin
   jTextView1.Text:='Hurra!';
   fl:=1;
  end else
  begin
   jTextView1.Text:='Es ist geschafft!';
   fl:=0;
  end;
end;
procedure TAndroidModule1.AndroidModule1Create(Sender: TObject);
begin
 fl:=0;
end;
end.
```

Jetzt auf:



dann



mit Ja bestätigen. (Oder das Verzeichnis i686-linux-android in i686-android-linux umbenennen!)

Es hat bei mir ohne Fehler kompiliert.

```
Projekt kompilieren, OS: android, Ziel: /home/bernd/Programme/MeinLazarus/projects/Test1/Android_x86_64/MeinProject1/libs/x
Building APK (Ant)... : Erfolg
```

Leider habe ich es nicht geschafft einen Emulator für 64bit auf meinem alten System zum Laufen zu bringen (Intel Core2 Duo mit 4GB Ram).