LAZ 8. INIFILES

Um INIFILES in Lazarus verwenden zu können, benötigen wir dafür den Einsatz einer spezielle Library: inifiles

Starte ein neues Lazarus Projekt (...das kannst du jetzt sicherlich schon ohne Anleitung)

Wir schauen uns jetzt den Quellcode im Editor mal etwas näher an:

LAZARUS-Quellcode

```
1. unit Unit1;
2. {$mode objfpc}{$H+}
3. interface
4. uses
5. Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, inifiles;
6. type
7. TForm1 = class(TForm)
8. ...
```

Alles anzeigen

Hier sehen wir einen Abschnitt **uses**
Testweise kannst du jetzt eine FTDI Komponente auf die Form legen und dir danach nochmal den Abschnitt "uses" anschauen...
Hast du gesehen, was passiert ist?

Spoiler anzeigen

Bei Erstellen eines neuen Projektes, fügt Lazarus alle nötigen Klassen in den "uses" Abschnitt hinzu, damit wir das Programm kompilieren können.
Wenn wir Komponenten aus der Komponentenleiste verwenden, wird der "uses" Abschnitt automatisch erweitert, falls das nötig ist.

Da wir jetzt INIFILES verwenden möchten und dies NICHT als fertige Komponente verwenden, müssen wir den "uses" Abschnitt selbst erweitern.
Füge dazu einfach inifiles mit Komma getrennt dem Abschnitt hinzu.

Projekt speichern.

1)
Wir fügen der Form nun ein paar sichtbare Komponenten hinzu, deren Zustand wir in der INI-Datei speichern werden.
Nach dem Neustart des Programms werden diese Eigenschaften wieder sichtbar sein.
TEdit, TCheckbox, TSpinEdit (Misc) der Form hinzufügen.
Ordne diese Komponenten auf der Form an, wie du das magst.
Speichere dein Projekt.

2) wir realisieren das Laden und Speichern unseres Textes auf Edit1, des Zustandes unserer Checkbox1 und die eingestellte Zahl aus dem Spinedit1.
Dazu benötigen wir die Ereignisse onShow und onClose von Form1. Erzeuge diese Proceduren.

Bis hierher haben wir immer mit den grafischen Komponenten gearbeitet. Diese Komponenten wurden erzeugt, weil wir das Icon in der Komponentenpalette ausgewählt und Form1 angeklickt haben.
Dann waren sie "plötzlich" vorhanden. Lazarus hat uns die ganze Arbeit abgenommen und hat die Komponenten jeweils erzeugt und registriert.
Was wir jetzt vorhaben, ist eine Komponente selbst zu erstellen, ohne Hilfe von Lazarus.

...nein, es wird nicht schlimm. Ich zeige dir gleich, wie einfach es ist, eine Komponente selbst zu erzeugen.
Hier zunächst der Quellcode für die onShow Procedure der Form1:

LAZARUS-Quellcode

```
1. procedure TForm1.FormShow(Sender: TObject);
2. var
3. MEINE_ERSTE_INI : TINIFile;
4. begin
5. MEINE_ERSTE_INI := TINIFile.Create( extractfilepath(Application.exename) + 'Meine.ini');
6. Edit1.text := MEINE_ERSTE_INI.ReadString('Form1', 'Edit1', '');
7. Checkbox1.checked := MEINE_ERSTE_INI.ReadBool('Form1', 'Checkbox1', false);
8. Spinedit1.value := MEINE_ERSTE_INI.ReadInteger('Form1','Spinedit1',0);
9. MEINE_ERSTE_INI.free;
10. end;
```

Was passiert im Einzelnen:

var wir haben die Procedure um den Abschnitt "var" erweitert. Sicher kennt ihr so etwas bereits von BASCOM, dort heißt es local
In diesem "var" Abschnitt werden lokale Variablen definiert, welche nur in dieser FUNCTION oder PROCEDURE gültig und sichtbar sind! Dem restlichen Programm sind die völlig unbekannt.

In diesem Abschnitt sehen wir die Definition von "MEINE_ERSTE_INI". Es ist vom Typ "TINIFile". TINIFile ist ein Klasse, welche in INIFiles.pas definiert ist. (du erinnerst dich? Das haben wir in "uses" eingebunden!)
Der Compiler weiß nun also, dass wir "MEINE_ERSTE_INI" benutzen möchten und das dies die Eigenschaften von "TINIFile" haben soll.
Mehr ist es bis dahin nicht und du musst an dieser Stelle nicht weiter darüber nachdenken!
Das ist so, wenn ich zu dir sage: "Auto". Sofort hast du eine Vorstellung davon, wovon ich spreche. In unserem Beispiel weiß der Compiler jetzt, was wir wollen.

Kommen wir zur folgenden Zeile im Quellcode:
MEINE_ERSTE_INI := TINIFile.Create(extractfilepath(Application.exename) + 'Meine.ini');

Links steht "MEINE_ERSTE_INI :=".
Von MEINE_ERSTE_INI hat der Compiler ja jetzt schon eine Vorstellung: Ist ein Objekt vom Typ TINIFILE.
Das ":=" weist etwas zu, das haben wir schon gelernt.

Spannend wird es jetzt: TINIFile.Create(

Hier wird unser Objekt REALITÄT! Wir erschaffen ein Objekt vom Typ TINIFile im Speicher unseres Rechners. In Klammern wird der Ort angegeben, an welchem sich die INI-Datei nachher befindet.
Es wird beim Create eine Textdatei angelegt, welche all unsere Daten abspeichern soll!

Wir übergeben den Dateinamen unserer INI Datei mit folgendem:
extractfilepath(Application.exename) + 'Meine.ini'

extractfilepath kann man sich leicht übersetzen. Es ist eine Anweisung, welche in Lazarus bereits zur Verfügung steht, ohne dass wir hierfür etwas in "uses" einbinden müssen 👍😁👍
Die Funktion extrahiert aus einer kompletten Pfadangabe incl. Dateinamen den eigentlichen Pfad. Also z.b. extractfilepath ('c:\windows\system32\regedit.exe') liefert als Ergebnis: 'c:\windows\system32\'

Nur haben wir den Pfad nicht als Konstante übergeben, sondern "application.exename". Dies enthält den Pfad und Namen unseres eigenen Programms!
Demnach erzeugen wir jetzt eine INI Datei namens 'Meine.ini' im gleichen Verzeichnis, wo unser Programm gespeichert ist, bzw erzeugt wurde (Projektverzeichnis)

Ich denke, damit dürfte die erste Zeile klar sein...

Nun können wir mit den Eigenschaften, welche eine INI Datei hat, arbeiten:
Wir lesen jeweils aus der Sektion "Form1" die entsprechenden Eigenschaften

```
Edit1.text := MEINE_ERSTE_INI.ReadString('Form1', 'Edit1', '');
Checkbox1.checked := MEINE_ERSTE_INI.ReadBool('Form1', 'Checkbox1', false);
Spinedit1.value := MEINE_ERSTE_INI.ReadInteger('Form1','Spinedit1',0);
```

Dabei unterstützt uns die Komponente INIFile! Wir können logische Werte (boolean), Text (String) oder auch Zahlen (integer, float) lesen und schreiben.
Der jeweils dritte Parameter ist beim Lesen von Eigenschaften wie bei ReadString der "default Value". Kann also nicht gelesen werden, weil die INI Datei z.B. noch gar nicht existent ist, so wird als Wert dieser dritte Parameter verwendet.

Erforsche:
Schreibe in eine Zeile des Editors MEINE_ERSTE_INI gefolgt von einem Punkt. Direkt nach Eingabe des Punktes erscheint ein Fenster mit allen Eigenschaften des Typ TINIFile
Sicher findest du hier Funktionen, welche dir neue Ideen geben... erweitere das Projekt um weitere sichtbare Komponenten und lese oder speichere Eigenschaften von ihnen.

Nun haben wir beim Erscheinen unseres Programmfenster Werte aus der INI gelesen. Wir wollen aber auch Änderungen beim Beenden das Programms in der INI anbspeichern.
Hierfür hattest du bereits die Procedure onClose erzeugt.
Der Quellcode dürfte jetzt keine bis wenig Geheimnisse für dich bereit halten...

LAZARUS-Quellcode

```
1.  procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
2.  var
3.  MEINE_ERSTE_INI : TINIFile;
4.  begin
5.  MEINE_ERSTE_INI := TINIFile.Create( extractfilepath(Application.exename) + 'Meine.ini');
6.  MEINE_ERSTE_INI.WriteString('Form1', 'Edit1', Edit1.text);
7.  MEINE_ERSTE_INI.WriteBool('Form1', 'Checkbox1', Checkbox1.checked);
8.  MEINE_ERSTE_INI.WriteInteger('Form1','Spinedit1',Spinedit1.value);
9.  MEINE_ERSTE_INI.free;
10. end;
```

Alle "Readxxxx" wurden zu "Writexxxx" und der dritte Parameter bei Writexxx ist nun einfach der Wert, welcher geschrieben wird.

Aufgabe
Führe obige Änderungen im Quellcode durch.
Kompiliere das Programm und nach Programmstart verändere den Text von Edit1, Checke Checkbox1 und trage eine Zahl in Spinedit1 ein.
Beende das Programm und starte es gleich wieder...

Wenn dies alles so weit geklappt hat, könntest du das Serielle Projekt mit der LazSerial Komponente um die INI Funktionalität erweitern und den gewählten COM Port speichern und laden.

Optional
Wir haben jeweils in der onShow und onClose Procedure das INI File erzeugt und über .free wieder zerstört.
Wir wandeln jetzt unser Programm so ab, dass beim Programmstart das INI-Objekt erzeugt wird und erst beim Beenden des Programm zerstört (freigegeben) wird.
Dazu müssen wir MEINE_ERSTE_INI aber GLOBAL definieren.
Lazarus hat uns dafür bereits einen Abschnitt eingerichtet: **Privat**
Als "Privat" sind dort definierte Variablen in unserer Form1 verfügbar.

Im Gegensatz dazu steht **Public**
Diese Variablen wären auch verfügbar in weiteren Fenstern unserer Anwendung! Ja, wir könnten weitere Fenster in

unserem Programm erzeugen und verwenden...

hier der Code zur abgewandelten Version

LAZARUS-Quellcode

```
1. procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
2. procedure FormShow(Sender: TObject);
3. private
4. { private declarations }
5. MEINE_ERSTE_INI : TINIFile;
6. public
7. { public declarations }
8. end;
9. var
11. Form1: TForm1;
12. implementation
14. {$R *.lfm}
16. { TForm1 }
19. procedure TForm1.FormShow(Sender: TObject);
21. begin
22. // Erzeuge unsere INI Datei beim Programmstart
23. MEINE_ERSTE_INI := TINIFile.Create( extractfilepath(Application.exename) + 'Meine.ini');
25. Edit1.text := MEINE_ERSTE_INI.ReadString('Form1', 'Edit1', '');
26. Checkbox1.checked := MEINE_ERSTE_INI.ReadBool('Form1', 'Checkbox1', false);
27. Spinedit1.value := MEINE_ERSTE_INI.ReadInteger('Form1','Spinedit1',0);
28. end;
30. procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
31. begin
32. MEINE_ERSTE_INI.WriteString('Form1', 'Edit1', Edit1.text);
33. MEINE_ERSTE_INI.WriteBool('Form1', 'Checkbox1', Checkbox1.checked);
34. MEINE_ERSTE_INI.WriteInteger('Form1','Spinedit1',Spinedit1.value);
35. // Zerstöre das Objekt; wird nicht mehr gebraucht
36. MEINE_ERSTE_INI.free;
37. end;
```

Alles anzeigen

Michael Köcher aka six1