# Inhaltsverzeichnis

# 01) Einleitung

In diesem Tutorial geht es darum eine Komplett eigene Komponente zu erstellen und sie in Lazarus zu Intrigieren. Ihr werdet sehen, dass es gar nicht so schwer ist. Zu nächst die Frage was ist eine Komponente eigentlich ? Eine Komponente ist eine einfache Klasse die meistens von einer vorherigen Klasse abgeleitet wird. Diese Klasse heißt auch „Abstracte Bases Klasse“, weil sie stellt ein Standard Verhalten das für alle Komponenten gilt Verfügung.

Die meisten GUI'S(Engl. **G**raphical **U**ser **I**nterface), haben sogar mehrere Bases Klasse, je nach Aufgabe. Es gibt in einer GUI verschiedene Bereiche z.b. eine Sichtbare Komponente wie Beispielsweise das TMemo, oder aber auch zunächst Unsichtbare Komponenten wie ein Dialog, oder die Timer Komponenten.

Lazarus unterscheidet im Prinzip grob gesehen zwei Bereiche: Eine Komponente die vom Toolkit wie z.b. GTK2 zuvergfügung gestellt wird, diese Komponenten sind dann von **TWinControl** abgeleitet oder aber auch Eigene Komponenten wie z.b. das **Panel** oder die **OI-Komponente**. Diese sind dann direkt von **TCustomControl** abgeleitet. **TCustomControl** spielt in diesem Kapitel ein große Rolle. Denn wir wollen eine komplett eigene Komponente entwickeln.

# 02) Ziel dieses Kapitels

Die Komponente wird nur eine einfache Aufgabe erfüllen, aber sie zeigt wie in Lazarus üblicherweise vorgegangen wird. Das Ziel ist es ein Eigenes Panel zu entwickeln, was im Gegensatz zum Standard **Panel** von Lazarus nicht nur eine Farbe hat sondern ein Farbverlauf anzeigt auf den ein Text da gestellt werden kann. Wir werden Standard Methoden und Eigenschaften von **TCustomControl** verwenden, außerdem werde ich Zeigen, wie diese Komponente in Lazarus Intrigiert werden kann und wie ein eigenes Icon zugewiesen werden kann. Auf einer Extra Seite gebe ich einige Tipps wie die Entwicklung beschleunigt werden kann. Es gibt eine ganze reihe Praktischer aber oft versteckter Funktionen im **Code Editor(SynEdit)** von Lazarus.

**Meine Lazarus Version lautet**
Version: 0.9.29 Beta,
SVN-Revison: 22080M
I386-linux-gtk2(beta)

# 03) Voraussetzungen

Da gibt es folgende:

1. Eine Funktionsfähige Lazarus-Umgebung, die jeder zeit neu erstellt werden kann. Um das zu Test: Im Menü unter **Werkzeuge\Lazarus Kompilieren**. Wenn das Fehler frei durchgeführt wird, kann davon ausgegangen werden, dass es keine Probleme gibt.
2. Elementare Kenntnisse in die Syntax von **Object Pascal**, außerdem sollte die Umgebung von Lazarus vertraut sein. Z.b. wie Kompiliere ich Programme und so weiter.
3. Erfahrung um Umgang mit der **OOP** sind unbedingt erforderlich

Alle drei Punkte sind nicht Teile dieses Tutorials !!! Wenn trotzdem damit angefangen wird, kann es sehr schnell zu Problemen und zu Frust führen.

# 04) Sonstige Hinweise

Auf eigene Verantwortung kann das Tutorial durchgearbeitet werden. Für Schäden in welcher Form auch immer bin ich nicht verantwortlich / haftbar !

# 05) Einige Hinweise und Tricks im Umgang mit Klassen

Hier habe ich einige Hinweise und Tricks und Tipps zusammen getragen die den Umgang mit Klassen vereinfachen soll und die Entwicklung erheblich beschleunigen sollen. Das geht meistens so das lästige Tipp Arbeit abgenommen wird.

## 01) Klassen Schneller Erzeugen

Um eine Klasse schneller zu erzeugen und um nicht immer die gleichen Sachen schreiben zu müssen gibt es die Code-Schablonen. Genutzt werden können sie mit **STRG+J**. Davor ein Wort eingeben, für Klassen z.b. **classC** und dann **STRG+J** und es wird ein Standard Header von einer Klasse erzeugt.

## 02) Typische Fehler vermeiden

Wenn eine Eigenschaft nicht im OI auftaucht liegt es meistens daran das sie nicht Installisiert wurde. Das tritt meistens im Zusammenhang mit Klassen auf. Beispielweise bei einer **TstringList**. Es empfiehlt sich auch in bei jeder Eigenschaft das Schlüsselwort Default zu nutzen. z.b. so
**property Test:String read fTest write fTest default „Hallo“;**

Das ist allerdings nur für die RTTI relevant, trozdem sollten alle Variablen im **Create** mit Relevanten Werten belegt werden.

## 03) Methoden schneller erzeugen

Um Methoden schneller erzeugen gibt es die Funktion Quelltext vervollständigen.
Über eine Tastenkombination kann sie aufgerufen werden, bei mir ist das **STRG+UMSCHALT+C**

## 04) Die Übersicht in Großen Units mit vielen Klassen und Datentypen behalten

Wer eine Unit hat mit sehr vielen Klassen und Datentypen verliert schnell die Übersicht, um das zu vermeiden gibt es in Lazarus die Funktion: **Code-Explorer** sie Analysiert den Code und erstellt ein Baum, um das Navigieren zu erleichtern.

## 05) Lösungen zu Problemen finden

Da Lazarus komplett **OpenSoruce** ist, gibt es für sehr viele Probleme bereits Lösungen, um in Erfahrung zu bringen, wie z.b. ein OI-Editor geschrieben wird kann so vorgegangen werden:
Wenn in der Komponenten-Leiste der Eintrag **RTTI** auftaucht kann dort der **OI** aufs From platziert werden. Jetzt einfach im **Source-Code** Editor einfach die Rechte Maustaste drücken und den ersten Menü-Eintrag auswählen: **Zur Deklaration springen.** Jetzt könnt ihr euch z.b. die **uses** Klausel anschauen. Einige Namen dürfte sofort auffallen.
Eine andere Möglichkeit ist die Funktion **In Dateien Suchen.**

## 06) Los geht's

Wir erstellen uns eine neue Unit. Die Unit sollte so aussehen

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

implementation

end.
```

**Quellcode 1** Eine Leere Unit

jetzt werden wir diese Unit erweitert, das was neu Hinzugekommen ist werde ich Fett da stellen

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Controls, Graphics;

type
  TMyPanel = class(TCustomControl)
  private

  protected

  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;

    procedure Paint; override;
  published

  end; // TMyPanel

implementation

end.
```

**Quellcode 2** Eine Modifizierte Unit

Zunächst habe ich zwei Units hinzugefügt. In der Unit **Controls** Steckt die Klasse **TCustomControl** , die wir verwenden wollen, in der Unit **Graphics** sind einige Sinnvolle Konstanten definiert, wie z.b. clRed oder aber auch die System Farben.

Jetzt habe ich eine erste Klasse erstellt, die direkt von **TCustomControl** abgeleitet ist. Ich habe der Klasse den Namen **TMyPanel** gegeben. Das **T** ist nur Geschmackssache. Es hat sich in der Benutzer Gemeinde eingebürgert, vor komplexeren eigene Datentypen wie Klassen oder Record z.b. ein T zu schreiben. Meiner Meinung nach basiert das auf Historischen Gründen. Früher gab es für Dos ein GUI mit den Namen, TVsion. Dieses gibt es sogar noch heute und ist sogar in FPC enthalten.

In dieser Klasse werden einige Standard Methoden überschrieben wie z.b. **Create** und **Destroy**.

An dieser stelle eine kleine Übersicht über weitere Nützliche Methoden

| **Methode** | **Beschreibung** |
|---|---|
| Create | Wird aufgerufen bei der Erzeugen der Komponente, hier können eigene Variablen Installisiert werden. |
| Destory | Wird aufgerufen bei der Vernichtung der Komponente, hier können eigene Variablen Freigeben werden. |
| Resize | Bei jeder Größen Veränderung wird diese Methode aufgerufen. |
| MouseDown | Wenn die Maustaste gedrückt wird. |
| MouseUp | Wenn die Maustaste losgelassen wird. |
| MouseMove | Wenn die Maus bewegt wird. |
| KeyDown | Wenn eine Taste gedrückt wird. |
| KeyPress | Wenn eine Taste gedrückt bleibt. |
| KeyUp | Wenn eine Taste losgelassen wird. |
| WMEraseBkgnd | Wenn der Hintergrund neu gezeichnet wird. Auch wenn hier ein WM vorsteht geht es Plattform übergreifend. |
| EraseBackground | Wird aufgerufen wenn der Hintergrund Neu gezeichnet wird. |

**Tabelle 1** Übersicht über Standard Methoden

Je nach Aufgabe können die hier vorgestellten Methoden Praktisch sein. Diese Methoden werden Automatisch aufgerufen von der LCL. Je nach Botschaft die eingeht. Es gibt sehr viele Botschaften für einige gibt es eine Methode für andere nicht. Wenn eine Botschaft wie z.b. **WMEraseBkgnd** aufgerufen werden soll muss die Unit **Messages** eingebunden werden. Wie hier gesehen werden kann, übernimmt die Klasse **TCustomControl** sehr viele Aufgaben für uns.

Hier eine weitere Übersicht, welche Aufgaben von der Klasse übernommen werden:

1. User Events werden Automatisch Verfügung gestellt.
2. Wenn das Fenster neu gezeichnet wird, werden Automatisch alle Komponenten neu gezeichnet
3. Wenn das darunter liegende Fenster vergrößert wird, werden alle Komponenten benachrichtigt.
4. Bei bedarf werden ScrollBalken angezeigt
5. Die Intrigiation in die Lazarus IDE wird Zurverfügungstellen gestellt.

Zur Erinnerung unser Source-Code sieht im Moment so aus

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Controls, Graphics;

type
  TMyPanel = class(TCustomControl)
  private

  protected

  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;

    procedure Paint; override;
  published

  end; // TMyPanel

implementation

end.
```

**Quellcode 3** Eine Modifizierte Unit

Für das Panel was wir erstellen wollen, werden wir es so erweitern

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Controls, Graphics;

type
  TMyPanel = class(TCustomControl)
  private
    fTextStyle:TTextStyle;
    fGradientDirection:TGradientDirection;
    fStartColor, fEndColor:TColor;

  protected

  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;

    procedure Paint; override;
  published
    property GradientDirection:TGradientDirection read fGradientDirection write
SetGradientDirection;
    property StartColor:TColor index 0 read GetColor write SetColor;
    property EndColor:TColor index 1 read GetColor write SetColor;

    property Alignment:TAlignment read fTextStyle.Alignment write SetAlignment;
    property Layout:TTextLayout read fTextStyle.Layout write SetLayout;
    property ShowPrefix:boolean read fTextStyle.ShowPrefix write SetShowPrefix;
    property Wordbreak:Boolean read fTextStyle.Wordbreak write SetBreak;
    property SystemFont:Boolean read fTextStyle.SystemFont write SetSystemFont;

  end; // TMyPanel

implementation

end.
```

**Quellcode 4** Unser Panel mit allen Notwendigen Eigenschaften

Ganz wichtig sind die Property's im **published** Bereich. Die hier eingetragenen Eigenschaften werden beim drauf klicken im Lazarus-Desiner im OI(**O**bjekt **I**nspektor) angezeigt. Die Setter Methoden sind insofern notwendig, da hier ein Automatisches neu zeichnen der Komponenten realisiert werden kann. mit **fTextStyle** hat es eine Besonderheit. Beim Datentyp **TTextStyle** handelt es sich um ein **Record**, dieses wird aber im OI nicht angezeigt. Auch wenn die RTTI hierfür Methoden und Möglichkeiten vorsieht, wird es einfach vom OI nicht unterstützt.

Jetzt sind wir soweit und können die Methoden ausschreiben. Um das zu vereinfache und um Zeit zu sparen gibt es auf der Hinweise Seite ein Trick.

```
function TMyPanel.GetColor(AIndex: integer): TColor;
begin
  case AIndex of
    0: result:=fStartColor;
    1: result:=fEndColor;
  end; // case
end; // TMyPanel.GetColor

procedure TMyPanel.SetAlignment(const AValue: TAlignment);
begin
  if fTextStyle.Alignment=AValue then exit;
  fTextStyle.Alignment:=AValue;
  Paint;
end; // TMyPanel.SetAlignment

procedure TMyPanel.SetBreak(const AValue: Boolean);
begin
  if fTextStyle.Wordbreak=AValue then exit;
  fTextStyle.Wordbreak:=AValue;
  Paint;
end; // TMyPanel.SetBreak

procedure TMyPanel.SetColor(AIndex: integer; const AValue: TColor);
begin
  case AIndex of
    0: fStartColor:=AValue;
    1: fEndColor:=AValue;
  end; // case
  Paint;
end; // TmyPanel.SetColor

procedure TMyPanel.SetGradientDirection(const AValue: TGradientDirection);
begin
  if fGradientDirection=AValue then exit;
  fGradientDirection:=AValue;
  Paint;
end; // TmyPanel.SetGradientDirection

procedure TMyPanel.SetLayout(const AValue: TTextLayout);
begin
  if fTextStyle.Layout=AValue then exit;
  fTextStyle.Layout:=AValue;
  Paint;
end; // TmyPanel.SetLayout

constructor TMyPanel.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  fGradientDirection:=gdVertical;
  fStartColor:=clBlue;
  fEndColor:=clYellow;
  fTextStyle.Layout:=tlCenter;
  fTextStyle.Alignment:=taCenter;
end; // TMyPanel.Create
```

**Quellcode 5** Die ausgeschriebenen Methoden von der Klasse **TMyPanel** Teil 1

```
procedure TMyPanel.Paint;
var
  r:TRect;
begin
  inherited Paint;
  r:=Rect(0,0,Width,Height);
  Canvas.GradientFill(r,StartColor, EndColor,GradientDirection);
  Canvas.TextRect(r,r.left,r.top,Caption,fTextStyle);
end; // TMyPanel.Paint

end.
```

**Quellcode 6** Die ausgeschriebenen Methoden von der Klasse **TMyPanel** Teil 2

Für einen Ersten Test sollten wir diese Komponente Dynamisch erzeugen, dass heißt manuell per Source-Code. Dazu fügen wir die **unit2** in **unit1** ein. Mit Folgendem Code kann nun die Komponente erzeugt werden. Vorgesetzt es befindet sich ein Panel mit dem Namen **Panel1** auf dem From

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  myPanel:=TMyPanel.Create(Panel1);
  myPanel.Align:=alClient;
  myPanel.Parent:=Panel1;
  myPanel.Caption:='Dies ist ein, einfacher Test von Heute';
    Button:=TButton.Create(myPanel);
    Button.Left:=10;
    Button.Top:=10;
    Button.Caption:='Klick Mich';
    Button.Parent:=myPanel;
end;
```

**Quellcode 7** Ein erster Test zu Laufzeit.

Die Variable **myPanel** und **Button** sollte im private Bereich des Froms angelegt werden. Ein erster Test sollte folgendes Bild Zeigen

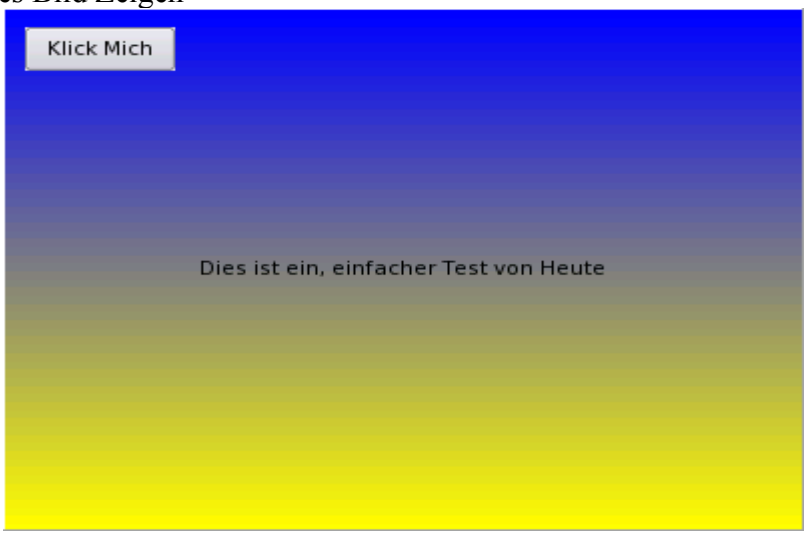


**Abbildung 1** Ein erstes ScreenShot der Komponente

# 07) Die IDE Integration

Im Letzten schritt geht es darum die Komponente zu Installieren, da das jedoch nicht Teil dieses Tutorial ist, verweise ich hier auf zwei Wikipedia Einträge in der Lazarus-Dokumenation.

1. **Packte erstellen und Komponente Installieren**

   http://wiki.lazarus.freepascal.org/Lazarus_Packages/de

2. **Icon für die Eigene Komponente erstellen**

   http://wiki.freepascal.org/Extending_the_IDE/de#Einer_neuen_Komponente_ein_Icon_f.C3.BCr_die_Komponentenpalette_verpassen

## 08) Weitere Nützliche Links zum Thema

1. http://wiki.freepascal.org/Extending_the_IDE/de
2. http://wiki.lazarus.freepascal.org/Streaming_components/de