ETO.FORMS UND OMNIGUI

Einmal nach Rom, bitte!

Ein Blick auf alternative Cross-Plattform-Entwicklungsumgebungen.

Gab es desktopseitig früher nur Windows, hat sich die Welt in den letzten Jahren stark gewandelt. Heute gilt es bei einer Entwicklung darauf zu achten, ob die Applikation später auch auf allen möglichen Systemen laufen kann, die gerade angesagt sind. Um dieses Ziel zu erreichen, gibt es natürlich die bekannten Lösungen wie Xamarin oder Electron, aber mittlerweile auch eine Vielzahl kleinerer Frameworks.

In diesem Artikel werfen wir einen Blick auf zwei eher unbekannte Vertreter der .NET-Richtung: OmniGUI und Eto.Forms.

OmniGUI: Überblick

Der erste Kandidat heißt OmniGUI. Das Framework ist, wie auch Eto.Forms, auf GitHub beheimatet [1] und benutzt die offene MIT-Lizenz. Als UI-Sprache kommt ein eigenes XAML-Derivat zum Zuge. und auch MVVM beziehungsweise ein Data Binding soll möglich sein.

Der erste Eindruck ist damit gut und auch der Umfang der unterstützten Systeme, wie Android, iOS, Windows 10, klingt vielversprechend. Allerdings findet für OmniGUI aktuell leider keine Weiterentwicklung mehr statt.

OmniGUI: Probleme

Die letzten Commits auf GitHub stammen vom November 2017, was bei einer stabilen Bibliothek an sich kein Hindernis wäre, hier aber durchaus problematisch ist, da selbst die Demo-Applikation ein altes Windows-10-SDK verwendet.

Schwerer wiegt allerdings, dass es im Vorfeld dieses Artikels nicht gelungen ist, das Projekt überhaupt zum Bauen zu bekommen, und ohne Build kann man es leider auch nicht nutzen, da es kein fertig gebautes NuGet-Package gibt.

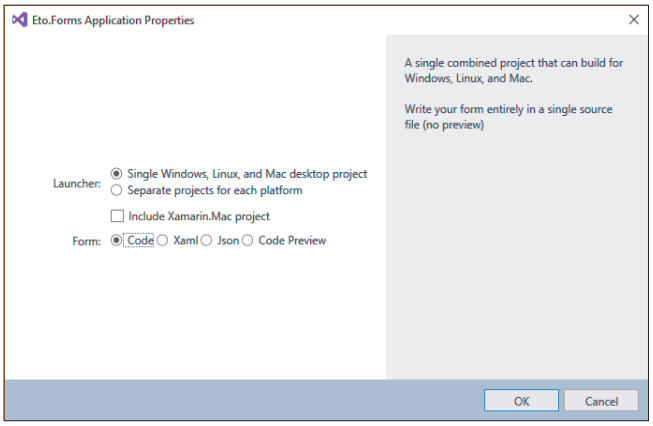
OmniGUI: Fazit

Das Projekt, das sich selbst mit Xamarin.Forms vergleicht, ist damit für unsere Reise nicht zielführend. Der Code und die Versprechungen machen zwar Lust auf mehr, allerdings ist es zum jetzigen Zeitpunkt leider nicht mehr so einfach nutzbar.

In der Readme-Datei auf GitHub ist auch vermerkt, dass das Projekt eher ein „Lernprojekt“ war – aus diesem Blickwinkel ist der Code sicherlich spannend zu lesen.

Eto.Forms: Überblick

Eto.Forms gehört bei unserem Thema eher zu den Veteranen als zu den Neuankömmlingen. Die erste Ankündigung dieser Bibliothek gab es im Jahr 2012 – damals bereits mit dem Ziel, .NET-Applikationen unter Windows, Mac OS X und Linux laufen zu lassen.



Das Visual-Studio-Template von Eto.Forms in Aktion (Bild 1)

Das Projekt nutzt die BSD-3-Lizenz, die ebenso gefahrlos kommerziell eingesetzt werden kann wie die MIT-Lizenz, und ist natürlich ebenfalls auf GitHub zu finden [2].

Wie der Name bereits erahnen lässt, hat die Bibliothek einen gewissen „Forms“-Charakter. Grundsätzlich wird nun aber neben den Desktop-Betriebssystemen noch iOS unterstützt, und die Android-Unterstützung ist aktuell in Entwicklung. Eto.Forms bietet dabei im Unterschied zu OmniGUI ein recht gutes Wiki, ein eigenes Visual-Studio-Template und ein NuGet-Package [3].

Eto.Forms-Praxisbeispiel

Um Eto.Forms etwas besser kennenzulernen, ist es vermutlich am einfachsten, eine kleine Applikation zu bauen, die ohne großen Aufwand unter Windows, Linux und (theoretisch – nicht getestet) unter macOS laufen sollte. Installiert man das Visual-Studio-Template (vergleiche Bild 1), so bekommt man beim Erstellen eines neuen Projekts auch einen ersten Einblick, wie eine Eto.Forms-Applikation aufgebaut ist.

Grundsätzlich werden im Template die großen Desktop-Systeme direkt angeboten. Zudem kann man sich entscheiden, in welcher Form das UI erstellt werden soll: In der Variante *Code* wird die komplette Form einfach im Code erstellt – im Grunde also so, wie der Name es schon erahnen lässt.

Die Varianten *Json* und *Xaml* erlauben es, die View deklarativ zu erstellen, und man erhält eine Code-behind-Datei, über die man einfache Interaktionen abfangen kann.

Die Variante *Code Preview* ist eine Mischung aus den genannten beiden Konzepten. Grundsätzlich erlaubt diese Variante es, die View im Code-behind zu erstellen, und man bekommt in Visual Studio eine Vorschau des UI angezeigt.

Listing 1: Hauptprojekt

```
class Program
{
  [STAThread]
  static void Main(string[] args)
  {
    new Application(Eto.Platform.Detect).Run(new
      MainForm());
  }
}
```

Eto.Forms: UI-Möglichkeiten

Liest man nun „XAML“, werden vermutlich einige Mitglieder der Entwicklergemeinde jubeln, während andere darüber fluchen möchten. Eto.Forms nutzt XAML nur, um das vorhandene UI zu serialisieren. Die bereitgestellten Controls in Eto.Forms sind im Grunde nur Wrapper für die einzelnen nativen Controls auf den jeweiligen Systemen.

Damit sieht eine Eto.Forms-Applikation auf macOS oder Linux nicht wie ein Alien aus, sondern sie passt sich in die definierte UI-Ästhetik ein.

Die Kehrseite davon ist allerdings, dass die eingebauten Style-Möglichkeiten recht begrenzt sind und eine Eto.Forms-Applikation auf Windows meist aussieht wie eine schlachtschiffgraue, „alte“ Windows-Forms-Applikation. Zwar lassen sich Hintergrundfarben, Schriftarten und andere Eigenschaften definieren, aber die Möglichkeiten, die man zum Beispiel in WPF genießt, gibt es in diesem Rahmen nicht.

Eto.Forms: Unterstützte Plattformen

Das Mapping der Eto.Forms-Controls auf die entsprechende Betriebssystemseite geschieht über die laufende Eto-Plattform. In der Bibliothek wird grob zwischen den Desktop- (WinForms, WPF, Gtk und macOS) und den mobilen Plattformen (iOS, WinRT und Android) unterschieden. Nutzt man aber keine plattformspezifischen Controls, lässt man Eto.Forms sinnvollerweise selbst die passendste Plattform auswählen.

Listing 2: StackLayout

```
Content = new StackLayout
{
  Padding = 10,
  Orientation = Orientation.Horizontal,
  Items =
  {
    "Hello World!",
    "Test"
  }
};
```

Hat man über das Visual-Studio-Template eine Solution erstellt, werden zwei Projekte angelegt:

- Das Hauptprojekt ist eigentlich nur dazu da, die passende Plattform auszuwählen und die Form zu laden. Natürlich kann man, wie in dem Dialog angeboten, auch pro Plattform ein eigenes Startprojekt hinzufügen.
- Die eigentliche Form ist in einem Extraprojekt definiert.

Der Code auf dem Hauptprojekt ist in Listing 1 zu sehen.

Nutzt man nun anstelle von *Detect* eine konkrete Plattform, so erkennt man zum Beispiel zwischen WPF und WinForms schnell diverse Unterschiede. Ein WPF-Button hat ein geringfügig anderes Erscheinungsbild als ein WinForms-Button. Es gibt natürlich noch andere Controls, die dies wesentlich deutlicher zeigen. Wichtig an dieser Stelle ist: Eto.Forms nutzt im Grunde die nativen Controls der jeweiligen Plattform.

Eto.Forms: Positionierung

Ähnlich wie in WPF gibt es eine Handvoll Layout-Controls, die als Container für Kindelemente dienen und diese entsprechend auf der Oberfläche zeichnen. Die Layoutelemente können natürlich auch kombiniert werden.

Der simpelste Layout-Container ist das StackLayout, das die Kindelemente einfach horizontal oder vertikal anordnet. Listing 2 zeigt hierfür ein einfaches Beispiel.

Da die Möglichkeiten bei diesem Layout-Typ schnell erschöpft sind, gibt es noch das mächtigere TableLayout. Wie der Name bereits erahnen lässt, handelt es sich hier um einen klassischen Tabellenaufbau. Man definiert die Rows und Cells und positioniert darin die entsprechenden Controls. Die Spaltengröße braucht eigentlich nicht bestimmt zu werden, da es einen gewissen Autosizing-Mechanismus gibt, allerdings kann man auch manuelle Größenangaben setzen. ▶

Listing 3: TableLayout

```
var layout = new TableLayout
{
  Padding = new Padding(10), // padding around cells
  Spacing = new Size(5, 5),
    // spacing between each cell
  Rows =
  {
    new TableRow(new Label {Text = "First Row"},
      new TextBox()),
    new TableRow(new Label {Text = "Second Row"},
      new ListBox()),
    new TableRow(new Label {Text = "Third Row"},
      TableLayout.AutoSized(new DropDown {Items =
      {"Item 1", "Item 2"}},
      null))
  }
};
Content = layout;
```

Listing 4: DynamicLayout

```
var layout = new DynamicLayout();

layout.BeginVertical(); // fields section
layout.AddRow(new Label { Text = "Field 1" },
  new TextBox());
layout.AddRow(new Label { Text = "Field 2" },
  new ComboBox());
layout.EndVertical();

layout.BeginVertical(); // buttons section
// passing null in AddRow () creates a scaled column
layout.AddRow(null, new Button { Text = "Cancel" },
  new Button { Text = "Ok" });
layout.EndVertical();
Content = layout;
```

Den Aufbau einer einfachen Tabelle sieht man in Listing 3. Recht ähnlich, aber mit mehr Flexibilität gibt es neben dem TableLayout noch das DynamicLayout. Dieser Layout-Typ nutzt intern ebenfalls ein TableLayout, allerdings erinnert der Aufbau eher an eine Art Flexbox in der Webentwicklung.

In Listing 4 sieht man einen einfachen Aufbau eines Formulars über das DynamicLayout. Die Schreibweise ist deutlich kompakter. Dieser Layout-Typ lässt allerdings zur Laufzeit keine Änderungen zu, sodass etwa dynamische Formulare über das normale TableLayout erstellt werden müssten.

Als letzten Layout-Typ gibt es noch das PixelLayout, bei dem man über genaue Pixelangaben Controls auf der Oberfläche anordnen kann. Listing 5 zeigt hierfür ein Beispiel.

Da man in Eto.Forms die verschiedenen Layouts sehr einfach mischen kann, lassen sich auch komplexere Oberflächen recht elegant umsetzen. Die Layouts können natürlich sowohl

Listing 5: PixelLayout

```
var layout = new PixelLayout();
var label = new Label { Text = "Hello!" };
layout.Add(label, 10, 10);
// during runtime you can move controls
layout.Move(label, 10, 100);
Content = layout;
```

im Code als auch in den entsprechenden XAML- oder JSON-Formen definiert werden.

Eto.Forms: MVVM und Data Binding

Ein zentrales Element moderner Oberflächen-Frameworks ist die Möglichkeit, die Eingaben auf einfache Weise an ein ViewModel weiterzugeben. Eto.Forms unterstützt diesen Ansatz ebenfalls, sodass auch das MVVM-Pattern eingesetzt werden kann. In Listing 6 ist zu sehen, wie man im Code das Data Binding erstellt. Dort wird erst das ViewModel definiert, das anschließend mit einer Textbox interagiert. Beim Eintippen von Text wird der Wert direkt in das ViewModel gespeichert.

Da wir nun Controls anordnen und mit Daten versorgen können, bleibt noch ein Punkt übrig: das Styling.

Eto.Forms: Styling

Über Styles lassen sich in Eto.Forms komfortabel bestimmte Styling-Eigenschaften von Controls setzen. In Listing 7 ist ein einfaches Beispiel zu sehen.

Globale Styles lassen sich ebenfalls definieren, sodass man die eigene Applikation auch an einem zentralen Ort stylen kann. Allerdings bieten die Styles, wie bereits erwähnt, nicht die Detailtiefe eines WPF-Styles an. Ein Button bleibt – solange keine plattformspezifischen Controls verwendet werden – ein Button, egal welche Plattform man nutzt.

Listing 6: Data Binding

```
public class MyModel : INotifyPropertyChanged
{
  string myString;
  public string MyString
  {
    get { return myString; }
    set
    {
      if (myString != value)
      {
        myString = value;
        OnPropertyChanged();
      }
    }
  }
   void OnPropertyChanged([CallerMemberName]
    string memberName = null)
    {
      PropertyChanged?.Invoke(this, new
      PropertyChangedEventArgs(memberName));
    }
    public event PropertyChangedEventHandler
      PropertyChanged;
}
// Form Code
var textBox = new TextBox();
textBox.TextBinding.BindDataContext((MyModel m) =>
  m.MyString);
```

Listing 7: Styles

```
Eto.Style.Add<TableLayout>("padded-table", table =>
{
  table.Padding = new Padding(10);
  table.Spacing = new Size(5, 5);
});

// will automatically get the padding & spacing
// applied
var myTable = new TableLayout { Style =
  "padded-table" };
```

Eto.Forms: Plattformspezifischer Code und Ausführung

Wie schon eingangs erwähnt, kann man in Eto.Forms für die verschiedenen Plattformen eigene Startprojekte definieren und so auch plattformspezifische Controls oder Verhalten mit einbringen. Eto.Forms erlaubt es zum Beispiel, native Controls über die jeweiligen .NET-Implementierungen einzubetten. So wäre es etwa möglich, WPF-Buttons mit einem moderneren Look zu erzeugen, als es im Framework eingebaut ist.

Beim Einsatz von Eto.Forms auf macOS und Linux sollte darauf geachtet werden, dass man Gtk# installiert hat, da dieses die Bindings zu den nativen Controls bereitstellt.

Wie bei allen Cross-Plattform-Entwicklungen gilt jedoch auch hier: Man sollte unbedingt das Aussehen der resultierenden Applikation auf einem nativen System prüfen.

Eto.Forms: Fazit

Eto.Forms zielt auf .NET-Entwickler, genauer gesagt auf WPF- und WinForms-Entwickler als Zielgruppe. Die Herangehensweisen sind den dort verwendeten zumindest recht ähnlich, und der erste Einstieg ist dank Wikis [4] und Visual-Studio-Projektvorlage schnell gemacht.

Die große Stärke von Eto.Forms ist, dass man die jeweiligen Plattform-UI-Controls nutzt, sodass sich die Applikation auf natürliche Weise in das System integriert. Dies ist aber zum Teil auch eine Schwäche, da man mit dem Styling zwar gewisse Eigenschaften setzen kann, die Anwendung aber dennoch ihren limitierten System-UI-Charme behält.

Haben wir den Weg nach Rom gefunden?

OmniGUI hat uns leider auf der Reise nicht weitgeholfen, aber Eto.Forms ist durchaus eine Option. Das Framework wird schon seit recht langer Zeit stetig weiterentwickelt, und im Notfall würde der Code auf GitHub bereitstehen.

Eto.Forms ist vermutlich nicht das Killer-Framework für Cross-Plattform-UI-Projekte, da es am Ende leider auch an der Bekanntheit mangelt, aber es kann sehr wohl ein zielführender Weg nach Rom sein. ■

[1] OmniGUI auf GitHub, https://github.com/OmniGUI/OmniGUI
[2] Eto.Forms auf GitHub, https://github.com/picoe/Eto
[3] NuGet-Package für Eto.Forms, https://www.nuget.org/packages/Eto.Forms
[4] Eto.Forms-Wiki, https://github.com/picoe/Eto/wiki

Robert Mühsig
ist Softwareentwickler, Microsoft MVP und arbeitet in der Schweiz bei der Sevitec Informatik AG. Seine Schwerpunkte sind die Desktop- und Webentwicklung und generell alles, was im Microsoft-Umfeld passiert.

dnpCode A1809Eto