A COMPARISON OF CREATING AN ANDROID APPLICATION USER INTERFACE IN XAML AND IN C# CODE

Igor Košťál[1], Martin Mišút[2]

Abstract: Almost every Android user application has some kind of user interface. Android programmers who create Xamarin.Forms applications and who uses the Microsoft Visual Studio development environment to do so can create user interfaces in the XAML (the Extensible Application Markup Language) or in the C# programming language. This paper deals with a comparison of creating this user interface by the first and the second way. We demonstrate the differences in the creation of Android application user interfaces in XAML and in C# by way of using two of our Android applications which were created by the Microsoft Visual Studio 2019 Enterprise, which work as text editors with the ability to store text to disc, and have user interfaces that are visually identical. However, the user interface of the first Android application was created in XAML with a C# support code, whereas the user interface of the second Android application was created entirely in C#. While comparing the source codes of the user interfaces of both these Android applications, we identify the advantages and disadvantages of both approaches to creating a user interface and we try to find out which of these approaches is better for maintaining and modifying user interfaces. In this study, we also contrast the processes of handling events of controls of a user interface created in the XAML code for the first Android application as well as that of the same user interface created in the C# code for the second Android application. Furthermore, we were interested in determining whether the different ways of creating user interfaces affected the execution time of basic operations that included disc files that were performed on the same data by both the applications. We assume that it does not fundamentally affect the execution time, and so, we performed an experiment to confirm or refute our assumption.

UDC Classification: 004.42, **DOI:** https://doi.org/10.12955/pns.v2.153

Keywords: Android Xamarin.Forms application, user interface, XAML, C# code

Introduction

An Android Xamarin.Forms application programmer who uses the Microsoft Visual Studio development environment to create it can construct its user interface in XAML (the Extensible Application Markup Language) or in the C# programming language. This allows him to use the Xamarin.Forms mobile development platform that lets developers write user-interface source codes in C# or XAML and those can be further compiled for iOS, Android, and Windows devices. When we specify this option, the developer can choose one out of three options. He can write the source code of the Xamarin.Forms application user interface entirely in C#, or entirely in XAML, or in a combination of the XAML and C# code, whilst for all the three options the event handlers of the user interface controls are always written in C# code. We were interested in determining the differences in creating the Android Xamarin.Forms application user interface in XAML and C#, as well as in finding out what advantages and disadvantages can result for developers and software companies from either one of the choices.

We created two Android Xamarin.Forms applications in the Microsoft Visual Studio 2019 Enterprise by using the Xamarin.Forms template. These applications worked as text editors and had user interfaces that were visually identical. However, the user interface of the first application was created in XAML with a C# support code and the user interface of the second application was created entirely in C#. Comparatively, we identify various advantages and disadvantages for both the approaches and we try to answer the question of determining which of these approaches is better for maintaining and modifying user interfaces. We also compare the processes of handling events of controls of the same user interface created in both the XAML code and the C# code for the Android application. We were interested to know whether the programmer had to deal with the process of handling events of controls of a user interface created in XAML. During the experiment, we also examined whether the different ways of creating the user interfaces of these Android applications affected the execution time of basic operations that involved disc files and that were performed on the same data in the applications. It is necessary to maintain information about all these aspects while deciding which approach is to be chosen. If it is possible to define all the objects of the Android application user interface controls together with the configurations of their properties in the XAML code, we would only be required to write event handlers for these controls in the C# code. Then, for large Xamarin.Forms projects, software companies could

[1] University of Economics in Bratislava, Faculty of Economic Informatics, Department of Applied Informatics, Bratislava, Slovakia, igor.kostal@euba.sk
[2] University of Economics in Bratislava, Faculty of Economic Informatics, Department of Applied Informatics, Bratislava, Slovakia, martin.misut@euba.sk

split this work on creating user interfaces of applications between designers and programmers. Designers would only need to design a user-friendly design of the Xamarin.Forms application user interface and programmers would only need to write event handlers for the interface controls and the remaining logic of such an application in C#. If our hypothesis is confirmed, the creation of the Xamarin.Forms application user interface in XAML and the event handlers for the interface controls in C# will allow a software company to create Xamarin.Forms applications with an attractive user interface and a running flawless, efficient, and powerful application core.

Using XAML and C# Code for Creating an Android Xamarin.Forms Application User Interface

We can write entire Xamarin.Forms applications in C# (Petzold, 2016). However, we can write some parts of Xamarin.Forms applications, for example, their user interfaces, in an alternative XAML (the Extensible Application Markup Language) provided by Xamarin.Forms.

XAML is a declarative markup language used for instantiating and initializing objects (Microsoft, 2021). We can use XAML for defining tree-structured visual user interfaces characteristic of graphical programming environments (Petzold, 2016). "The XAML implementation in Xamarin.Forms supports the visual elements defined by Xamarin.Forms, such as *Label*, *BoxView*, *Frame*, *Button*, *StackLayout*, and *ContentPage*" (Petzold, 2016), which we can use for creating a Xamarin.Forms application user interface control.

When we write the entire Xamarin.Forms application code in C#, then we usually "define the initial appearance of its user interface in the constructor of a class that derives from the *ContentPage* class" (Petzold, 2016). If we choose to use XAML, the markup generally replaces this constructor code (Petzold, 2016). We further find that *XAML* provides a more succinct and elegant definition of the user interface and has a visual structure that better mimics the tree organization of the visual elements on the page (Petzold, 2016). *XAML* is also generally easier to maintain and modify, and is more concise than its equivalent C# code (Petzold, 2016).

But, XAML has several deficiencies too that are intrinsic to markup languages: XAML has no loops, no flow control, no algebraic calculation syntax, and no event handlers. However, using XAML has several distinct advantages that can compensate for some of these deficiencies.

We are more inclined to know whether the mentioned advantages of using XAML for creating an Android application user interface will be demonstrated while analyzing the source codes of the two of our Android application user interfaces as opposed to the use of the C# language.

Two Android Xamarin.Forms Applications with Visually Identical User Interfaces, but Created in Different Ways

We created two Android Xamarin.Forms applications, the *EditorWithXAML* and the *EditorWithoutXAML*, with the help of the Microsoft Visual Studio 2019 Enterprise by using the same *Xamarin.Forms* template. It, in turn, created three projects: one common project (the Portable Class Library project), and two platform projects — for Android and for iOS. Here, we only deal with Android applications and projects related to them. The *Xamarin.Forms* template prefers using XAML to create the application user interface. Therefore, the common projects of both applications contain XAML files. It is due to using the same template to create solutions for both applications that the common projects of both solutions contain source files with the same name, but with different content. These Android applications work as text editors and have user interfaces that are visually identical. Nevertheless, the user interface of the *EditorWithXAML* application was created in XAML with a C# support code, and the user interface of the *EditorWithoutXAML* application was created entirely in C#.

The source codes of the common projects for both Android Xamarin.Forms applications are divided into four parts which are then saved into files:

- *App.xaml* — this XAML file contains an *x:Class* specification which indicates that the *App* class in the *EditorWithXAML* or *EditorWithoutXAML* namespaces derives from the *Application* class.
- *App.xaml.cs* — this is the code-behind file for the *App.xaml* file. The *App.xaml.cs* file contains the *App* class definition.
- *MainPage.xaml* — this XAML file contains the definition of our *EditorWithXAML* Android application's tree-structured user interface that was created in XAML (Figure 2a). Objects of

all controls are instantiating and initializing in a XAML code of this file. This XAML file for the *EditorWithoutXAML* Android application only contains the empty *ContentPage* element.

- *MainPage.xaml.cs* — this is the code-behind file for the *MainPage.xaml* file. The *MainPage.xaml.cs* file contains a code that supports the markup and creates the underlying logic of the user interface for our *EditorWithXAML* Android application and the full logic of the entire *EditorWithoutXAML* Android application (Figure 2b).

The key class in the source codes of both applications is the *MainPage* class which derives from the *ContentPage* class. Its definition is split into *MainPage.xaml* and *MainPage.xaml.cs* source files in both applications.

The *MainPage* class of the *EditorWithXAML* application contains the following members:

- The *MainPage* constructor
- The *EditorTextChanged*, *OnFileListViewItemSelected*, *OnDeleteTextButtonClicked*, *OnDisplay PathButtonClicked*, *OnSaveButtonClicked*, *OnDeleteButtonClicked*, and *OnNewFileButtonClicked* methods — these are event handlers for the application's user interface controls
- The *RefreshListView* method — reloads all items of the *fileListView* object of the *ListView* class
- The *GetFiles* method — obtains the current filenames from an application local storage
- The *labelPathFileName*, *editor*, *fileListView*, *deleteTextBtn*, *saveFileBtn*, *deleteFileBtn*, and *newFileBtn* fields — the objects of the controls of the application's user interface
- The *fileName8v15e*, *fileName10v17e*, *fileName_userG*, *fileName_LogF*, *selectedItem_str*, and *docsPath* fields — these fields use some event handlers and other methods

Figure 1 shows all members of the *MainPage* and *App* classes of the *EditorWithXAML* application.

The *MainPage* class of the *EditorWithoutXAML* application contains similar and rather identical members as the *MainPage* class of the *EditorWithXAML* application:

- The *MainPage* constructor
- The *Editor_TextChanged*, *FileListView_ItemSelected*, *DeleteTextbtn_Clicked*, *DisplayPathbtn_Clicked*, *SaveFileBtn_Clicked*, *DeleteFileBtn_Clicked*, and *NewFileBtn_Clicked* methods — these are event handlers for the application's user interface controls
- The *RefreshListView* method — reloads all items of the *fileListView* object of the *ListView* class
- The *GetFiles* method — obtains the current filenames from an application local storage
- The *labelPathFileName*, *editor*, *fileListView*, *saveFileBtn*, *deleteFileBtn*, and *newFileBtn* fields — the objects of the controls of the application's user interface
- The *fileName_app1*, *fileName_app2*, *fileName_user*, *fileName_LogF*, *selectedItem_str*, and *docsPath* fields — these fields use some event handlers and other methods

The first object that we instantiate in XAML code of the *EditorWithXAML* application user interface is the object of the *StackLayout* class, which arranges its children in a stack. This *StackLayout* object is expressed as a *StackLayout* XML element in the XAML code of the *MainPage.xaml* source file. The *StackLayout* XML element contains three child XML elements, the *Label*, the *Editor*, and the *Grid*. These child elements represent objects of the *Label*, the *Editor*, and the *Grid* classes. The *Margin* attribute of the *StackLayout* XML element sets the inner padding of the layout.

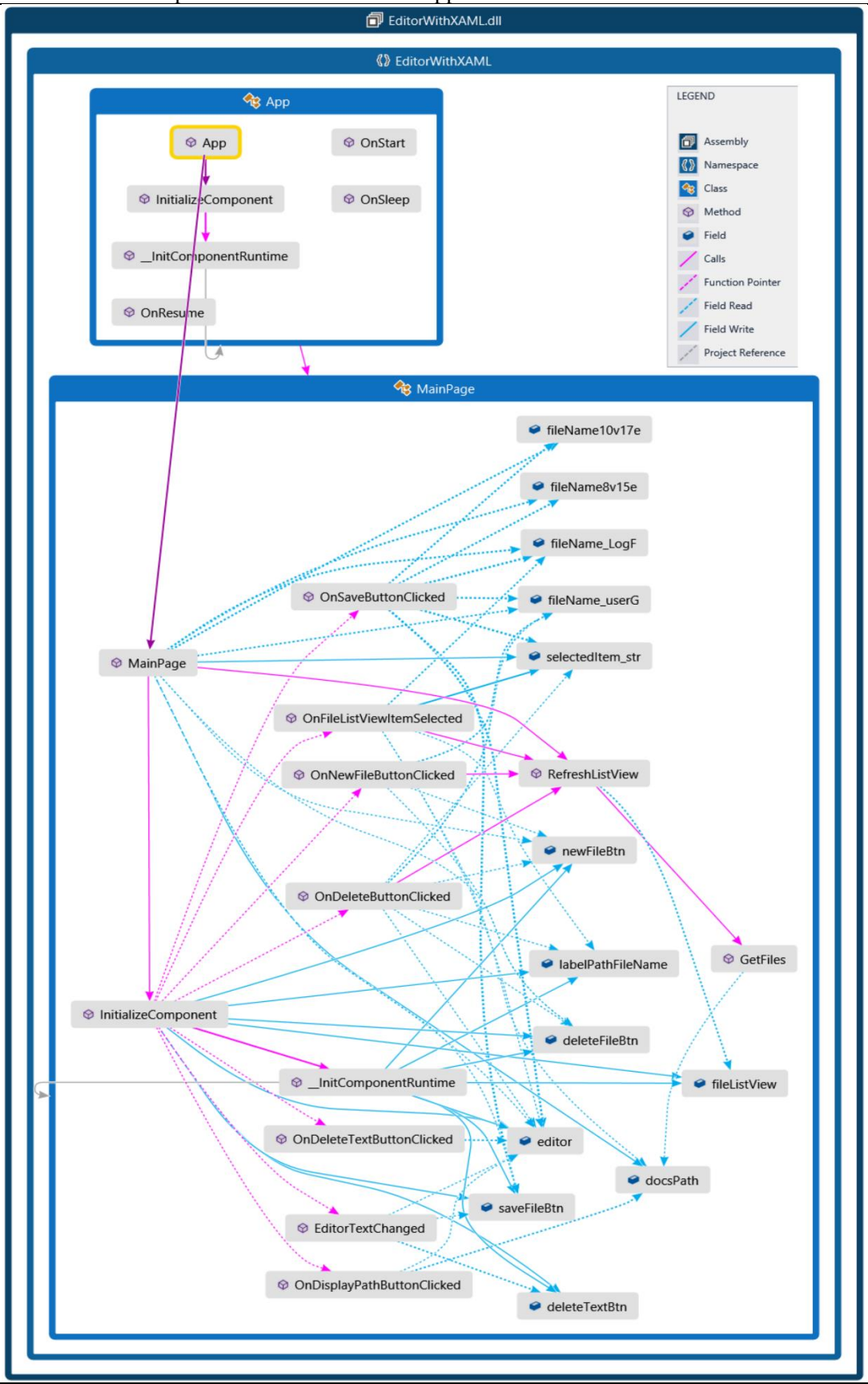
The *Text* attribute of the *Label* child element sets the text for the label, the *HorizontalOptions* attribute specifies how the label is to be positioned relative to its parent (in this case the *StackLayout* object) and the *FontAttributes* attribute sets the font style of the text for this label.

The *TextChanged* attribute of the *Editor* child element attaches the *EditorTextChanged* event handler to the *TextChanged* event of this element. This event is raised when the text of the editor changes. The *EditorTextChanged* event handler resides in the *MainPage.xaml.cs* code-behind file.

The *Placeholder* attribute of the *Editor* child element sets the text that is displayed when the control is empty and the *HeightRequest* attribute sets the desired height override for this element.

The *x:Name* attribute of the *Label* and *Editor* child elements gives specific names to objects of the *Label* and *Editor* classes.

Figure 1: The code map of the *EditorWithXAML* application



Source: Authors

Figure 2: The parts of the *MainPage.xaml* file of the *EditorWithXAML* application a) and the *MainPage.xaml.cs* file of the *EditorWithoutXAML* application b)

```
<ContentPage . . . x:Class="EditorWithXAML.MainPage">
  <StackLayout Margin="10,10,10,10">

    <Label x:Name="labelPathFileName"
           Text="Directory of an opened file"
           HorizontalOptions="Center"
           FontAttributes="Bold"/>

    <Editor x:Name="editor"
           TextChanged="EditorTextChanged"
           Placeholder="Enter your text"
           HeightRequest="100" />

    <Grid>
       <Grid.RowDefinitions>
           <RowDefinition Height="*" />
           <RowDefinition Height="*" />
           <RowDefinition Height="*" />
           <RowDefinition Height="*" />
           <RowDefinition Height="*" />
       </Grid.RowDefinitions>
       <Grid.ColumnDefinitions>
           <ColumnDefinition Width="*" />
           <ColumnDefinition Width="*" />
           <ColumnDefinition Width="*" />
           <ColumnDefinition Width="*" />
       </Grid.ColumnDefinitions>

    <ListView x:Name="fileListView" Grid.Row="0"
           Grid.Column="0" Grid.RowSpan="3"
           Grid.ColumnSpan="2"
           ItemSelected="OnFileListViewItemSelected">
    </ListView>

    <Button x:Name="deleteTextBtn"
           Text="Delete Text" Grid.Row="0"
           Grid.Column="2"
           Clicked="OnDeleteTextButtonClicked"
           FontSize="12"
           FontAttributes="Bold"/>

  <Button Grid.Row="0" Grid.Column="3" Text="Display
Path"
           Clicked="OnDisplayPathButtonClicked"
           FontSize="12"
           FontAttributes="Bold"/>

    <Button x:Name="saveFileBtn" Grid.Row="1"
           Grid.Column="2" Text="Save File"
           Clicked="OnSaveButtonClicked" FontSize="12"
           FontAttributes="Bold" />

    <Button x:Name="deleteFileBtn" Grid.Row="1"
           Grid.Column="3" Text="Delete File"
           Clicked="OnDeleteButtonClicked"
FontSize="12"
           FontAttributes="Bold"/>

    <Button x:Name="newFileBtn" Grid.Row="2"
           Grid.Column="3" Text="New File"
          Clicked="OnNewFileButtonClicked"
FontSize="12"
           FontAttributes="Bold"/>

    </Grid>
  </StackLayout>
</ContentPage>
```

a)

```
public partial class MainPage : ContentPage  {  . . .
  Label labelPathFileName; Editor editor;
  ListView fileListView; Button deleteTextbtn;
  Button saveFileBtn; Button deleteFileBtn;
  Button newFileBtn;

  public MainPage()  {
  StackLayout mainStack = new StackLayout  {
Padding = new Thickness(10, 10, 10, 10)  };

  labelPathFileName = new Label  {
           Text = "Directory of an opened file",
           FontAttributes = FontAttributes.Bold,
           HorizontalOptions = LayoutOptions.Center  };

  editor = new Editor  {
  Placeholder = "Enter your text", HeightRequest = 100  };
  editor.TextChanged += Editor_TextChanged;

  mainStack.Children.Add(labelPathFileName);
  mainStack.Children.Add(editor);

  Grid grid = new Grid  {
    RowDefinitions =  {
         new RowDefinition { Height = GridLength.Star },
         new RowDefinition { Height = GridLength.Star },
         new RowDefinition { Height = GridLength.Star },
         new RowDefinition { Height = GridLength.Star },
         new RowDefinition { Height = GridLength.Star }  },
    ColumnDefinitions =  {
         new ColumnDefinition { Width = GridLength.Star },
         new ColumnDefinition { Width = GridLength.Star },
       new ColumnDefinition { Width = GridLength.Star } }};

  fileListView = new ListView  { };
fileListView.ItemSelected += FileListView_ItemSelected;
  grid.Children.Add(fileListView, 0, 2, 0, 3);

  deleteTextbtn = new Button  {
           Text = "Delete Text", FontSize = 12,
           FontAttributes = FontAttributes.Bold  };
  deleteTextbtn.Clicked += DeleteTextbtn_Clicked;
  grid.Children.Add(deleteTextbtn, 2, 0);

  Button displayPathbtn = new Button  {
           Text = "Display Path", FontSize = 12,
           FontAttributes = FontAttributes.Bold  };
  displayPathbtn.Clicked += DisplayPathbtn_Clicked;
  grid.Children.Add(displayPathbtn, 3, 0);

  saveFileBtn = new Button  {
           Text = "Save File", FontSize = 12,
           FontAttributes = FontAttributes.Bold  };
  saveFileBtn.Clicked += SaveFileBtn_Clicked;
  grid.Children.Add(saveFileBtn, 2, 1);

  deleteFileBtn = new Button  {
           Text = "Delete File", FontSize = 12,
           FontAttributes = FontAttributes.Bold  };
  deleteFileBtn.Clicked += DeleteFileBtn_Clicked;
  grid.Children.Add(deleteFileBtn, 3, 1);

  newFileBtn = new Button  {
           Text = "New File", FontSize = 12,
           FontAttributes = FontAttributes.Bold  };
  newFileBtn.Clicked += NewFileBtn_Clicked;
  grid.Children.Add(newFileBtn, 3, 2);

  mainStack.Children.Add(grid);
  Content = mainStack;  . . . }
```

b)

Source: Authors

We have used objects of the *Grid* class to lay out additional controls in both the Android applications' user interfaces. This *Grid* class provides a powerful layout mechanism that organizes its children into rows and columns of cells (Petzold, 2016). The *Grid* class is designed specifically for two-dimensional arrays of children and it can also be very useful for managing layouts that adapt to both portrait and landscape modes (Petzold, 2016). A *Grid* object can be defined and filled with children in either C# code or XAML, but the XAML approach is easier and clearer, and hence, by far, the more common one (Petzold, 2016).

The *Grid* object definition of the *EditorWithXAML* application contains definitions of two important properties — the *RowDefinitions* and *ColumnDefinitions*. The *RowDefinitions* property defines a collection of *RowDefinition* objects; one *RowDefinition* object for every row in the *Grid* object. The *ColumnDefinitions* property defines a collection of *ColumnDefinition* objects; one *ColumnDefinition* object for every column in the *Grid* object. Both these collections define the row and column characteristics of the *Grid* object (Petzold, 2016). The *Grid* objects of the *EditorWithXAML* and *EditorWithoutXAML* applications have five rows and four columns. All *RowDefinition* objects of both *Grid* objects set the *Height* equal to "*" or *GridLength.Star* and all *ColumnDefinition* objects of both *Grid* objects set the *Width* equal to "*" or *GridLength.Star*, which means that the height and width of both the *Grid* objects' areas are divided equally between the five rows and the four columns.

To specify the height of *RowDefinition* objects and the width of the *ColumnDefinition* objects in the *EditorWithoutXAML* application's *grid* object, we use the *Star* value of the *GridLength* structure.

The *Grid* object of the *EditorWithXAML* application is expressed as the *Grid* XML element in the XAML code of the *MainPage.xaml* source file. The *Grid* XML element contains six child XML elements, the *ListView*, and five *Button* elements. These child elements represent one object of the *ListView* class and five objects of the *Button* class. The *Grid.Row* and *Grid.Column* attributes of these child elements specify cells of the *Grid* object which these elements occupy. The *Grid.RowSpan* and *Grid.ColumnSpan* attributes of the *ListView* child element specify that this element occupies the two rows of the first column and the two columns of the first row.

The *ItemSelected* attribute of the *ListView* child element attaches the *OnFileListViewItemSelected* event handler to an *ItemSelected* event. When a user selects some item in the *ListView* control, it fires an *ItemSelected* event. The *OnFileListViewItemSelected* event handler itself resides in the *MainPage.xaml.cs* code-behind file.

The *Clicked* attribute of every *Button* child element attaches a given event handler to a *Clicked* event of a given child element. When a user taps a *Button* control, it fires a *Clicked* event (Petzold, 2016). Event handlers of all *Button* child elements reside in the *MainPage.xaml.cs* code-behind file.

The *Text* attribute of every *Button* child element sets the text to display the content of a given button.

The *FontSize* and *FontAttributes* attributes of every *Button* child element set the size of the font of a given button text and the font style of this button text.

The *x:Name* attribute of the *ListView* child element and four *Button* child elements gives specific names to objects of the *ListView* and *Button* classes.

As we mentioned above in a description of the XAML code of the *EditorWithXAML* application user interface elements, event handlers of the user interface controls, which represent these elements, are written in C# code saved into the *MainPage.xaml.cs* code-behind file.

All that we did in the XAML code of our *EditorWithXAML* application (Figure 2a) is also described in detail above. We did the same in the second *EditorWithoutXAML* application in the C# code (Figure 2b).

We created the *mainStack*, *grid* objects and the *labelPathFileName*, *editor*, *fileListView*, *deleteTextbtn*, *displayPathbtn*, *saveFileBtn*, *deleteFileBtn*, *newFileBtn* objects of all controls of the *EditorWithoutXAML* application user interface using constructors of the *StackLayout*, *Grid*, *Label*, *Editor*, *ListView* and *Button* classes. We set and specified all properties of these objects within these constructors. We attached event handlers to particular events of all the controls by *EventHandler* delegates. We also used *Add* methods for adding children to the *Children* collections of the *mainStack* and *grid* objects.

For example, here we create a new *editor* object by using the constructor of the *Editor* class, setting its two *Placeholder* and *HeightRequest* properties within the constructor, attaching the *EditorTextChanged* event handler to the *TextChanged* event of this object, and by adding the *editor* child to the *Children* collection of the *mainStack* object.

```
editor = new Editor
{
   Placeholder = "Enter your text",
   HeightRequest = 100
};
editor.TextChanged += Editor_TextChanged; // Attach the EditorTextChanged event handler to
                                          // the TextChanged event
mainStack.Children.Add(editor);  // Add the editor as a second child of mainStack.
```

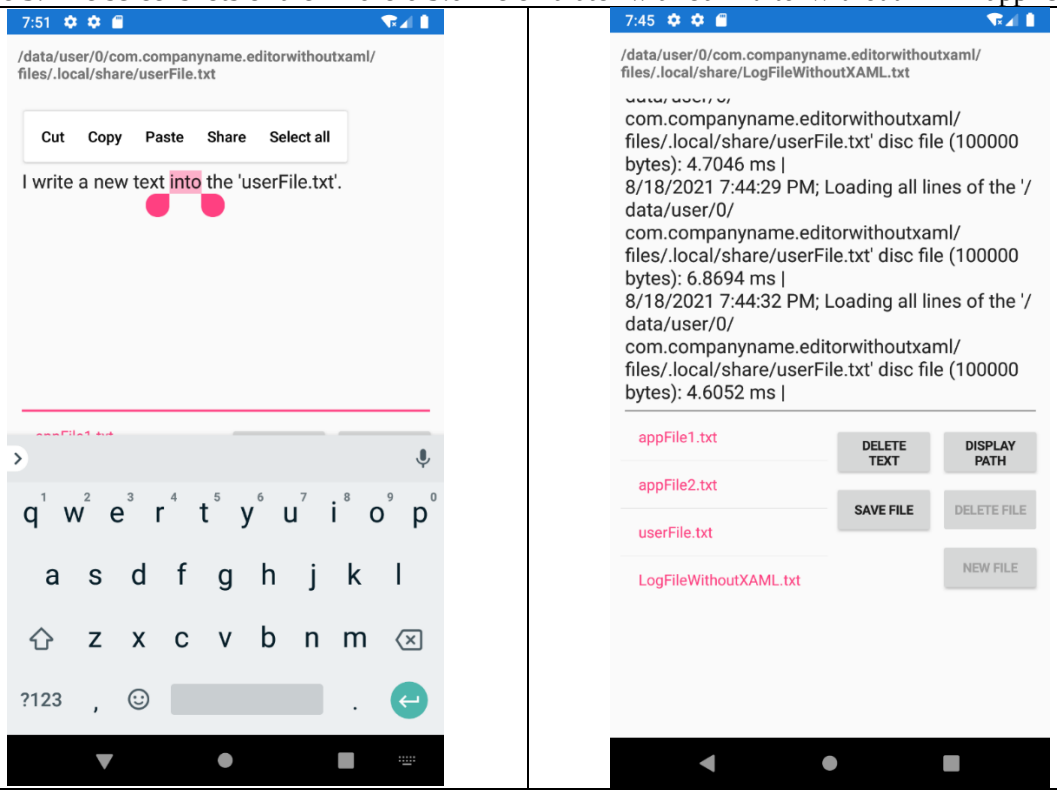
Other constructors and other codes of the main *MainPage* constructor are displayed in Figure 2b).

As observed, it is feasible to define all objects of the Android application user interface controls "in C# code, but usually without the clarity or orderliness of the XAML definition" (Petzold, 2016).

When we compare the XAML code with the C# code for the two of our Android Xamarin.Forms application user interfaces, which have the same visual appearance, we can clearly see that the XAML code "provides a more succinct and elegant definition of the user interface and has a visual structure that better mimics the tree organization of the visual elements on the page" (Petzold, 2016).

Also, as is evident, it is fairly viable to define all objects of the Android application user interface controls together with configurations of their properties in the XAML code, and we would only be required to write the event handlers for these controls in the C# code. The software company could then divide the work on creating Android Xamarin.Forms application user interfaces between designers and programmers. This confirms the hypothesis that we formulated towards the end of the Introduction.

Figure 3: The screenshots of the Android 9.0 Pie emulator with our *EditorWithoutXAML* application



Source: Authors

Figure 3 shows screenshots of the Android 9.0 Pie emulator with our *EditorWithoutXAML* application running on it. This Android Xamarin.Forms application works as a text editor and allows a user to work with the three *appFile1.txt*, *appFile2.txt*, and *LogFileWithoutXAML.txt* disc files. The user can create a new *userFile.txt* file to work with it and store the file's new content, and then, he/she can also delete this file. The *EditorWithoutXAML* application provides an Android keyboard and Copy & Paste tools to

the user while working with these disc files. Our application also displays the directory path to the currently opened disc file.

A Comparison of the Process of Handling Events in the *EditorWithXAML* and *EditorWithoutXAML* Applications

The processes of handling events in the *EditorWithXAML* and *EditorWithoutXAML* applications are quite unalike.

The *EditorWithXAML* application has all the user interface elements created in the *MainPage.xaml* XAML file, which means that each element in this file is instantiated with a call to the constructor of the corresponding class, and the resultant object is initialized by setting properties apart from attribute values for this element. The attachment of the *EditorTextChanged*, *OnFileListViewItemSelected*, *OnDeleteTextButtonClicked*, *OnDisplayPathButtonClicked*, *OnSaveButtonClicked*, *OnDeleteButtonClicked*, and *OnNewFileButtonClicked* event handlers to the *TextChanged*, *ItemSelected*, and *Clicked* events in XAML is performed by setting the event attributes of particular elements. The programmer sees through the instantiation and initialization of the user interface controls' objects and attaches event handlers to the events of these objects, as described previously. In fact, these actions are performed during the application build process when the *MainPage.xaml* XAML file is parsed and the following *MainPage.xaml.g.cs* code file is generated by a tool:

Figure 4: The *MainPage.xaml.g.cs* file of the *EditorWithXAML* application

```
[assembly: global::Xamarin.Forms.Xaml.XamlResourceIdAttribute("EditorWithXAML.MainPage.xaml", "MainPage.xaml",
typeof(global::EditorWithXAML.MainPage))]

namespace EditorWithXAML {

    [global::Xamarin.Forms.Xaml.XamlFilePathAttribute("MainPage.xaml")]
    public partial class MainPage : global::Xamarin.Forms.ContentPage {

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Label labelPathFileName;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Editor editor;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.ListView fileListView;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button deleteTextBtn;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button saveFileBtn;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button deleteFileBtn;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private global::Xamarin.Forms.Button newFileBtn;

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private void InitializeComponent() {
            global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MainPage));
            labelPathFileName =
        global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Label>(this, "labelPathFileName");
            editor = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Editor>(this, "editor");
            fileListView = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.ListView>(this,
                                                                                                    "fileListView");
            deleteTextBtn = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Button>(this,
                                                                                                "deleteTextBtn");
            saveFileBtn = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Button>(this,
                                                                                                    "saveFileBtn");
            deleteFileBtn = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Button>(this,
                                                                                                "deleteFileBtn");
            newFileBtn = global::Xamarin.Forms.NameScopeExtensions.FindByName<global::Xamarin.Forms.Button>(this,
                                                                                                    "newFileBtn");
        }
    }
}
```

Source: Authors

This generated file contains a partial class definition of *MainPage* which derives from the *Xamarin.Forms.ContentPage* class and which contains the *InitializeComponent* method. Right here, in the code of this partial class, we execute real instantiation of the *EditorWithXAML* application user interface controls' objects. Each element with an *x:Name* attribute becomes a private object of a given class. The *LoadFromXaml* method, called by the *InitializeComponent* method, performs the real initializing the controls' objects and the real attaching the event handlers to the controls' events. Therefore, it initializes all the objects defined in the XAML file, connects them all together in a parent-child relationship, attaches event handlers defined in code to events set in the XAML file, and sets the resultant tree of objects as the content of the page (Microsoft, 2021).

After the *MainPage.xaml.g.cs* file is generated, this file along with the *MainPage.xaml.cs* code-behind file are compiled together by the C# compiler.

The *EditorWithoutXAML* application does not use XAML to create controls for its user interface. It instantiates and initializes the objects of these controls directly in the code of its *MainPage.xaml.cs* file (Figure 2b). Also directly attaching the *Editor_TextChanged*, *FileListView_ItemSelected*, *DeleteTextbtn_Clicked*, *DisplayPathbtn_Clicked*, *SaveFileBtn_Clicked*, *DeleteFileBtn_Clicked*, and *NewFileBtn_Clicked* event handlers to the *TextChanged*, *ItemSelected*, and *Clicked* events is performed in the code of this file (Figure 2b). Hence, the *MainPage.xaml.g.cs* file generated from the *MainPage.xaml* file of the *EditorWithoutXAML* application contains instantiating none objects of controls for its user interface.

Figure 5: The *MainPage.xaml* file of the *EditorWithoutXAML* application with the empty *ContentPage* element

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="EditorWithoutXAML.MainPage">

</ContentPage>
```

Source: Authors

Figure 6: The *MainPage.xaml.g.cs* file of the *EditorWithoutXAML* application

```
[assembly: global::Xamarin.Forms.Xaml.XamlResourceIdAttribute("EditorWithoutXAML.MainPage.xaml", "MainPage.xaml",
typeof(global::EditorWithoutXAML.MainPage))]

namespace EditorWithoutXAML {

    [global::Xamarin.Forms.Xaml.XamlFilePathAttribute("MainPage.xaml")]
    public partial class MainPage : global::Xamarin.Forms.ContentPage {

        [global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]
        private void InitializeComponent() {
            global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(MainPage));
        }
    }
}
```

Source: Authors

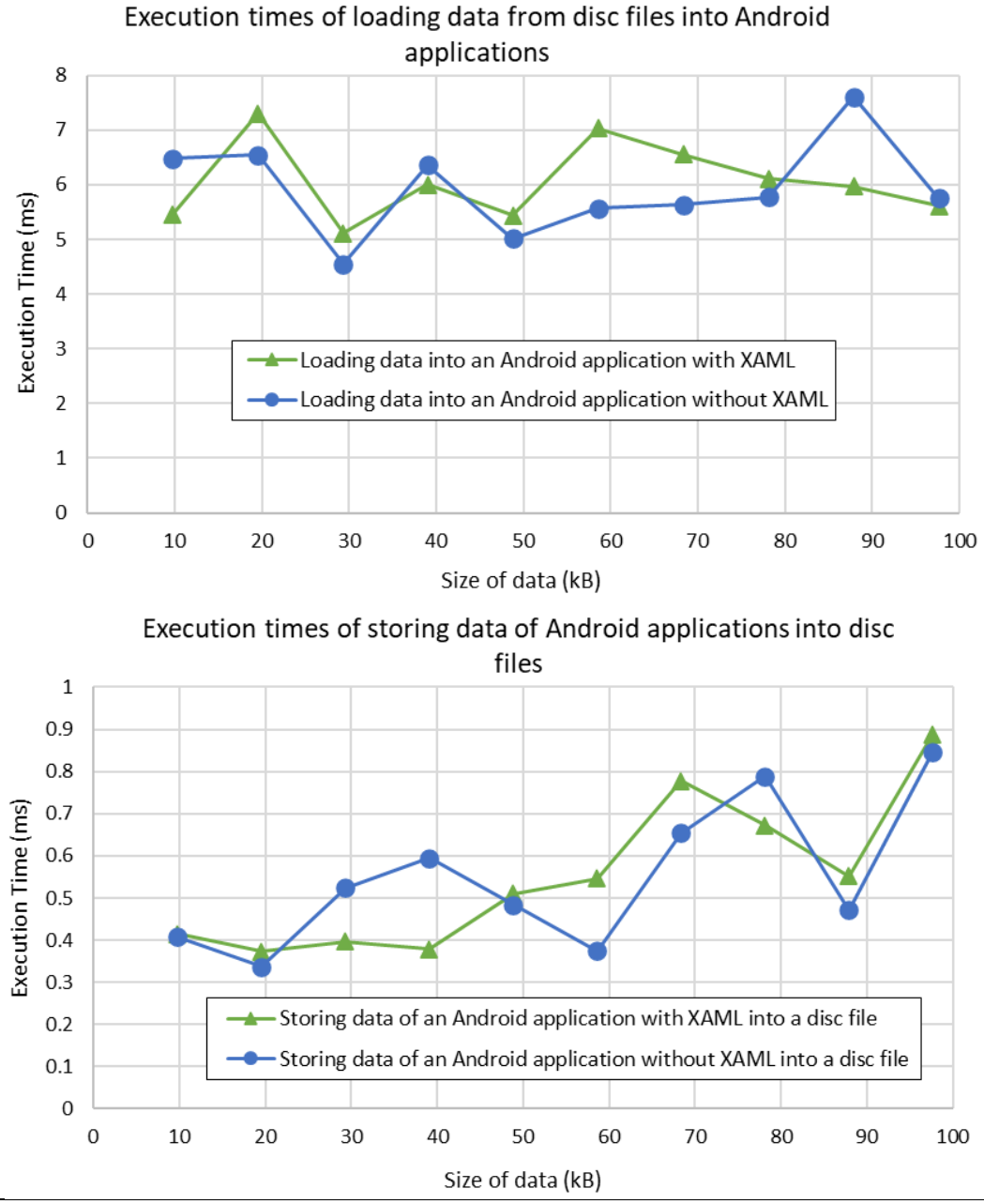
The process of handling events in the *EditorWithXAML* application that uses XAML to create its user interface controls is much more complex than in the *EditorWithoutXAML* application that creates its user interface controls directly in the C# code. However, this process of handling events is performed by the *EditorWithXAML* application itself; the programmer does not have to deal with it. If we do not load and do not parse the XAML file of the *EditorWithXAML* application by calling the *LoadFromXaml* method at run time (that is the same in our case), then, using XAML to create controls of its user interface does not significantly affect the application's execution efficiency. Besides, it can also provide several benefits to the programmer, e.g., easier modification and maintenance of the user interface XAML code, etc.

The Experiment, its Results and their Brief Analysis

We assume that the different ways of creating the user interface of Android applications, either in XAML with a C# support code, or only in C# code, do not fundamentally affect the execution time of the basic operations with disc files in these applications.

To verify this assumption, we performed an experiment using the two of our *EditorWithXAML* and *EditorWithoutXAML* Android applications. During this experiment, both Android applications performed the loading of data from the *userFile.txt* disc files into their *editor* controls and the storing of data from these controls into the *userFile.txt* disc filesEach of these applications was measured for the execution time of this particular operation, later written into its log file. The *EditorWithXAML* Android application wrote its execution time into the *LogFileWithXAML.txt* file, and the *EditorWithoutXAML* Android application wrote its own execution time into *LogFileWithoutXAML.txt*. Both operations, loading and storing data, were performed by both the Android applications on the same 10 data sets which contained ASCII characters and had the following sizes: 10000 bytes (9.77 kB), 20000 bytes (19.53 kB), 30000 bytes (29.30 kB), 40000 bytes (39.06 kB), 50000 bytes (48.83 kB), 60000 bytes (58.59 kB), 70000 bytes (68.36 kB), 80000 bytes (78.13 kB), 90000 bytes (87.89 kB), 100000 bytes (97.66 kB). Both the Android applications were running on the Android 9.0 Pie emulator, which in turn ran on the Microsoft Windows 10 Home operating system. The computer on which the experiment was performed had the following basic hardware configuration: Intel Core i5-8250U Processor (6MB Cache, 1.60 GHz, 4 GT/s, 4 Cores, 8 Threads), RAM: 8 GB.

Figure 7: Execution time operations of loading and storing data of both Android applications



Source: Authors

One of the outputs of the *EditorWithoutXAML* Android application displayed in its *editor* control, which shows the measured execution times of the operation for loading data from the *userFile.txt* disc file with a size of 100000 bytes into the *editor* control, is shown in Figure 3 (the right). Similar outputs were displayed by the *EditorWithXAML* Android application in its *editor* control.

The execution time of loading and storing the same 10 data sets by both Android applications are clearly displayed in the graphs (Figure 7).

Brief results analysis

It is obvious from the above graphs which show the execution time of loading and storing the same 10 data sets by both the Android applications that the differences between the execution time of the same operations on the same data by the *EditorWithXAML* application and *EditorWithoutXAML* application are minimal. The greatest measured variance in the execution time while loading data between the applications is 1.643 ms, and while storing data, it is 0.2163 ms. These are very small deviations. Thus, it follows that the different ways of creating the user interface of an Android application, whether by XAML with C# support code or only by C#, do not fundamentally affect the execution time of the basic operations including disc files that are performed by the application. This confirms our hypothesis. So, the use of XAML to create the user interface of the Android application does not worsen the execution time of the examined basic operations of the Android application.

Conclusion

It is evident from the comparison of the XAML code and the C# code for the two of our Android Xamarin.Forms application user interfaces, having the same visual appearance, that the XAML code "provides a more succinct and elegant definition of the user interface and has a visual structure that better mimics the tree organization of the visual elements on the page" (Petzold, 2016) and the XAML code is more concise than the equivalent C# code (Petzold, 2016). Additionally, we can see from this comparison that it is possible to define all objects of the Android application user interface controls together with the configurations of their properties in the XAML code, and we would only need to write event handlers for these controls in the C# code. The software company can then divide the work on creating the Android Xamarin.Forms applications user interfaces between designers and programmers. This would allow the company to develop applications with an attractive user interface with a running flawless, efficient, and powerful application core. These facts confirm the hypothesis that we formulated towards the end of the Introduction.

It is evident from our comparison of the process of handling events in *EditorWithXAML* and *EditorWithoutXAML* applications that the programmer doesn't have to deal with the process of handling events of controls of a user interface created in the XAML code because this process is already performed by the *EditorWithXAML* application itself.

From the experiment, it is obvious that the method used to create the Android application user interface (in XAML with C# support code, or in C#) does not fundamentally affect the execution time of basic operations with disc files in the application. So, using XAML to create the Android application user interface does not worsen the execution time of some of thebasic operations of the application.

Our findings suggest that it is advantageous to create an Android application user interface in XAML with the C# support code as it also agrees with the results of this experiment.

Acknowledgements

This work was supported by project VEGA No. 1/0373/18 entitled "Big data analytics as a tool for increasing the competitiveness of enterprises and supporting informed decisions" by the Ministry of Education, Science, Research and Sport of the Slovak Republic.

References

Albahari, J., & Albahari, B. (2017). C# 7.0 in a Nutshell. O'Reilly Media, Inc.

Butow, E., & Ryan, T. (2002). C# Your visual blueprint for building. NET applications. Hungry Minds, Inc.

Microsoft Corporation. (2021, March 8). Documentation. Retrieved from https://docs.microsoft.com

Petzold, Ch. (2016). Creating Mobile Apps with Xamarin.Forms. Xamarin Inc., Redmond: Microsoft Press.

Snider, E. (2018). Mastering Xamarin.Forms. Packt Publishing Limited.

Versluis, G., & Thewissen, S. (2018). Xamarin.Forms Solutions. APress.