Working with attached to repair documents in the system of automatic registration of repairs

Arseny Zorin

1 Introduction

Since the appearance of computers, progress isn't standing still. Technology has enormously improved in last decade. Among trends that appear one can find an ambient wireless Internet access to printing prosthesis on 3D printers or cloud storages. The latter became storage de-facto standard for past few years. It is used not only for private, but also for corporates document storing and data archiving.

The object of research is the authorized service center (ASC) that provides warranty and paid repair of household appliances. There are four branches situated in different parts of the city that take equipment for repair. Each repair operation performed by ASC produces an number of attached documents, their names equal to the ticket number:

- A photo of schild from back of equipment. This file is stored in a common folder for schilds' photos of this manufacturer.
- A scanned warranty ticket. This file is stored in a common folder for scanned documents of this manufacturer.
- An outbound repair receipt. Produced in case the repair was carried out at the client's residence. This file is stored in a common folder of outbound repairs for this manufacturer.
- Various acts provided to a client including a technical condition act, a non-repairable act, a replacing device act.

All document folders are stored at branches offices and at the head office server. Data is transmitted between branches and the head office through FTP protocol.

This work was motivated by a need for modernization of tools for documents storage and transmition. The software programming language was C# instead of VBA. For scanning documents and making photo of them new libraries will be used. All documents storage were moved to the cloud. Data will be stored and treated at the distributed servers in network. While using cloud storages, users have a possibility to access the data from different places and from different devices which have an Internet access. Also cloud storage can prevent losing information in case of failure of a local storage.

2 Practical implementation

2.1 Working with AForge.NET

Taking photo of a schild: after the button pressed, a new window with video player element will open. Stream video from selected web-camera will be played on this element. After the button "Take a snapshot" pressed, there will be capture of the picture from video player element and an obtained picture will be stored in the specified directory in .jpg format.

The .NET Framework does not support working with web-cameras it means that there is no video playback element. However, there is namespace AForge.Controls with VideoSourcePlayer element in AForge.NET. This element was created only for using in Windows Forms software. The decision of developing software with using WPF was taken and there are few methods of solution:

- The creation of a hybrid application. In addition to the WPF window, there will be

one Windows Forms window.
- Using a WindowsFormsHost element. This element allows to place Windows Forms element on WPF page.

The latter variant was selected. Forits realization we have to add: System.Windows.Forms.Integration and AForge.Controls namespaces, WindowsFormsHost element and VideoSourcePlayer inside it (Fig. 1).

```
    xmlns:wfi =
        "clr-namespace:System.Windows.Forms.Integration;assembly=WindowsFormsIntegration"
    xmlns:aforge =
        "clr-namespace:AForge.Controls;assembly=AForge.Controls">
<Grid>

    <wfi:WindowsFormsHost HorizontalAlignment="Left" Height="379" Margin="10,41,0,0"
                          VerticalAlignment="Top" Width="472">
        <aforge:VideoSourcePlayer x:Name="sourcePlayer" Width="472" Height="379"
            NewFrame="videoSourcePlayer1_NewFrame"/>
    </wfi:WindowsFormsHost>
```

Fig. 1. Adding namespces and elements

In addition to the playback element we have to add elements for interaction with VideoSourcePlayer element:
- ”Take a snapshot” button
- List of playback sources

After addition of all elements, the window in ”Constructor” mode will looks like on fig. 2.

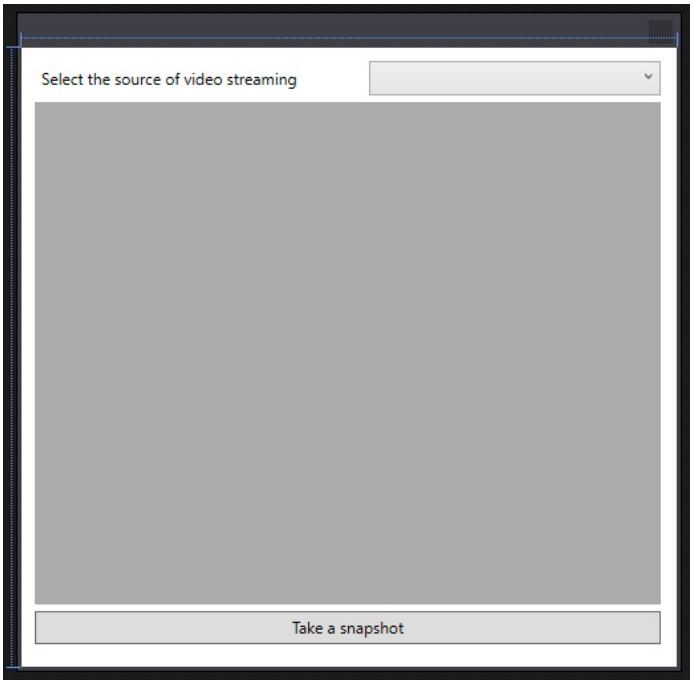

Fig. 2. Window view in the constructor mode

There is the AForge.Video.DirectShow. VideoCaptureDevice class for capturing video from an input-output (I/O) device. It is necessary to set device’s moniker (the name of a cpecific instance of an object. It is a COM-object that has its own way to create and maintain a special interface - IMoniker), which will capture the picture. In addition, we have to add event handler NewFrame. Every time new frame will be received, this event will raise. Received frame sends to the handler in the Bitmap format. It can be processed and stored in the specified format in the right place.

The list will be filled with names of all connected I/O devices and the default device will be the first in the list.

The event comboBox-Sources_SelectedIndexChanged rises after changing selection of source. It stops streaming from previous I/O device, set a moniker to the VideoCaptureDevice class of a new selected device and resumes streaming.

Pressing the button ”Take a snapshot” changes the value of the take_pict boolean variable on true. The value of this variable checks in the event raised after receiving of a new frame. If its value equals to true, then received image saves in .jpg format in specified place. In our case - Yandex.Disk.

The method EndOfWork, that stops streaming and frees used resources, is called after after closing the window.

2.2 Working with TWAIN library

2.3 Working with Microsoft.Office.Interop

2.3.1 Working with Microsoft.Office.Interop.Word

2.3.2 Working with Microsoft.Office.Interop.Excel

2.4 Working with Yandex.Disk

3 Conclusion