

How to get and edit gallery images in Qt and WP

This article explains basic operations of getting an image from the gallery and editing it in Qt and Windows Phone 7.

Introduction



In this article we will discuss how to perform the following operations in Qt and WP7 and 8.

- Opening an image from the Gallery
- Edit the image / Adding a frame to the image
- Saving the modified image

Qt implementation

Qt provides more than one option for opening/accessing an image from the gallery. One option is to use common desktop services to open the gallery and select a file (more details how to implement this can be found in [CS001426 - Using common desktop services in Qt](#)).

This section explains how to use the [DocumentGalleryModel](#) QML Element in the gallery module to access images. Note that [QDocumentGallery](#) offers similar kind of functionality to the QML Element - more details can be found [here](#).

To use the gallery module you need to specify it in your pro file:

```
CONFIG += mobility
MOBILITY = gallery
```

Getting images using QML

To use the gallery module you first need to import it in your QML.

```
import QtMobility.gallery 1.1
```

The DocumentGalleryModel provides a media model to be used with one of the View elements. Below we have used GridView:

```
GridView {
    anchors.fill: parent
    cellWidth: 128
    cellHeight: 128

    model: DocumentGalleryModel {
        rootType: DocumentGallery.Image
        properties: [ "url" ]
    }

    delegate: Image {
        source: url
        width: 128
        height: 128
        MouseArea
        {
            anchors.fill:parent
            onClicked: console.log(url)
        }
    }
}
```



DocumentGalleryModel also supports other media types like Audio - the type it can be accessed by changing the rootType:

```
rootType: DocumentGallery.Image
```

In addition, the DocumentGalleryModel also allows to set a filter using [GalleryWildcardFilter](#) to define which file types are returned/filtered by the model, for example jpg, png, mp3, etc.

```
filter: GalleryWildcardFilter {
    property: "fileName";
    value: ".jpg";
}
```

Editing and saving the image

Once the user has selected the image from the gallery, it's time to edit the image. QImage provides functions for accessing the pixels of an image by its x,y co-ordinates. So all we need to do is create the QImage object for our file, and modify the pixels we want.

For example, the following snippet would edit the upper edge of an image (10 pixels) with green color and save it.



```

QImage iImage(path);
QRgb color;
color = qRgb(0,255,0);

for(int i=0;i<mPixmap->height();i++)
{
    for(int j=0;j<10;j++)
    {
        iImage.setPixel(i,j,color);
    }
}
iImage.save("TestImage.jpg");

```



WP7 implementation

The Windows Phone implementation is similar to that provided by Qt Desktop services - it allows you to launch a chooser to select an image which is then edited. The chooser is an instance of `PhotoChooserTask`.

To start with we create a basic UI for our WP7 app, as shown below.

```

<phone:PhoneApplicationPage
    x:Class="ImageProcessing.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True" Loaded="PhoneApplicationPage_Loaded">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Image Processing" Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="Your Image" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>
        <Image Grid.Row="1" Height="481" HorizontalAlignment="Left" Margin="10,114,0,0" Name="imgContainer" Stretch="Fill" VerticalAlignment="Top" />
        <Button Content="Select" Grid.Row="1" Height="108" Margin="0,0,308,0" Name="btnSelect" VerticalAlignment="Top" Click="btnSelect_Click" />
        <Button Content="Effect" Grid.Row="1" Height="109" HorizontalAlignment="Left" Margin="160,0,0,0" Name="btnEffect" VerticalAlignment="Top" Click="btnEffect_Click" />
        <Button Content="Save" Height="109" HorizontalAlignment="Right" Margin="0,0,6,0" Name="btnSave" VerticalAlignment="Top" Width="60" Click="btnSave_Click" />
    </Grid>
</phone:PhoneApplicationPage>

```

Pressing the "Select" button would call `btnSelect_Click()` function, where we launch the gallery/photo chooser in WP7.

```

void btnSelect_Click(object sender, RoutedEventArgs e)
{
    // Launch photo chooser
    photoChooser.ShowCamera = true;
    photoChooser.Show();
    photoChooser.Completed += new EventHandler<PhotoResult>(photoChooserTask_Completed);
}

```

where `photoChooser` is an instance of `PhotoChooserTask`. The event handle returns to `photoChooserTask_Completed()` function once the user has selected an image. Once the image is selected, the selected image is shown to the user through the `Image` component `imgContainer`.

```

private void photoChooserTask_Completed(object sender, PhotoResult e)
{
    // Handler after the user selects the photo
    photoResult = e;
    if (e.TaskResult == TaskResult.OK)
    {
        image.SetSource(e.ChosenPhoto);
        imgContainer.Source = image;
    }
    else if (e.TaskResult == TaskResult.None)
    {
        return;
    }
}

```

Editing the image

When the user presses the "Effect" button, we edit the image to apply a frame to the image, as shown in the screenshot. The difference to note here between Qt and WP7 is that `QImage` allows to access pixels of an image using x,y coordinates; while in WP7 we can only access the pixels of the image by its pixel number through the `Pixels.SetValue()` function of a `WriteableBitmap`, as shown below.

```

private void btnEffect_Click(object sender, RoutedEventArgs e)
{
    // Function for editing image

    // defining color
    int color = unchecked((int)0xFFCC3366);
    WriteableBitmap wbmp = new WriteableBitmap(image);

    for (int i = 0; i < wbmp.PixelHeight; i++)
    {
}

```

```

`int index = (i * wbmp.PixelWidth);
// width of the frame border is 30, hence -14 to 15
for (int counter = -14; counter < 15; counter++)
{
    if (index + counter <= wbmp.PixelHeight * wbmp.PixelWidth && index + counter > 0)
    {
        wbmp.Pixels.SetValue(color, index + counter);
    }
}

for (int i = 0; i <wbmp.PixelWidth; i++)
{
// width of the frame border is 15
for (int counter=0;counter<15;counter++)
{
    wbmp.Pixels.SetValue(color, i + (counter * wbmp.PixelWidth));
}

for (int i = (wbmp.PixelWidth * (wbmp.PixelHeight - 1)); i < (wbmp.PixelWidth * wbmp.PixelHeight); i++)
{
// width of the frame border is 15
for (int counter = -14; counter < 1; counter++)
{
    wbmp.Pixels.SetValue(color, i + (counter * wbmp.PixelWidth));
}

// setting the edited image in imgContainer
imgContainer.Source = wbmp;

// assigning to global variable
wb = wbmp;
}

```

Saving the image

Finally, the edited image can be saved by the SaveImage() function.

```

public void SaveImage(Stream imageStream, int orientation, int quality, WriteableBitmap wb)
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplication())
    {
        if (isolatedStorage.FileExists(imgContainer.Name))
            isolatedStorage.DeleteFile(imgContainer.Name);

        IsolatedStorageFileStream fileStream = isolatedStorage.CreateFile(imgContainer.Name);
        image.SetSource(photoResult.ChosenPhoto);

        wb.SaveJpeg(fileStream, wb.PixelWidth, wb.PixelHeight, orientation, quality);
        fileStream.Close();
    }

    using (IsolatedStorageFile myIsolatedStorage = IsolatedStorageFile.GetUserStoreForApplication())
    {
        using (IsolatedStorageFileStream fileStream = myIsolatedStorage.OpenFile(imgContainer.Name, FileMode.Open, FileAccess.Read ))
        {
            MediaLibrary mediaLibrary = new MediaLibrary();
            Picture pic = mediaLibrary.SavePicture(imgContainer.Name, fileStream);
            fileStream.Close();
        }
    }

    // Launch the photo chooser again to show the saved image
    PhotoChooserTask photoChooserTask = new PhotoChooserTask();
    photoChooserTask.Show();
}

```

Here's how the app would look on a WP7 phone:



Summary

In this article, we saw how to implement basic image processing in Qt and Windows Phone 7.

