Chapter 6
Camera, Graphics and Video

A smartphone provides a convenient way to carry around and display a collection of photographs – photographs which may have been taken with the built-in camera or imported from a website or computer. It can also be used to show videos. The phone’s camera can be used to capture short videos, which can be played back on the phone. Longer videos would usually be streamed from another source and not held on the phone. TouchDevelop scripts provide an easy way to create, search for and display photos. A script can play videos but it cannot create them with the phone’s camera.

6.1 Camera 83
6.2 Manipulating pictures 86
6.3 Static graphics drawing and display 92
6.4 Playing videos from the internet 95
6.5 Media Player – photos and videos 96

6.1 Camera

The phone should have at least one camera located on the opposite side from the screen, capable of taking high-quality pictures or videos. This is the phone’s primary camera. Many phones have a second camera located above the screen which captures the user’s image and is intended for use in video calls, like those provided by Skype.

The TouchDevelop API provides access to the camera (or cameras) via its senses service. The senses methods relevant to using the cameras are listed in Table 6-1. Two of these methods return instances of type Camera. This
datatype provides methods for retrieving information about the camera and for taking a quick low-quality picture.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>senses → camera: Camera</td>
<td>Returns the primary (rear facing) camera.</td>
</tr>
<tr>
<td>senses → front camera: Camera</td>
<td>Returns the front facing camera, if there is one; otherwise the result is invalid.</td>
</tr>
<tr>
<td>senses → take camera picture: Picture</td>
<td>Uses the primary camera (on the opposite side from the screen) to take a picture.</td>
</tr>
<tr>
<td>wall → set background camera( camera: Camera): Nothing</td>
<td>Causes images from the selected camera to be used as the background for the wall</td>
</tr>
</tbody>
</table>

There are two ways to take a picture with the primary camera. The two approaches behave quite differently. First, the expression

\[
\text{senses} \rightarrow \text{take camera picture}
\]

takes a high-quality picture whereas

\[
\text{senses} \rightarrow \text{front camera } \rightarrow \text{preview}
\]

takes a low-quality picture. Also, the first version, the high-resolution version, does not immediately take the picture whereas the second one does. The first version calls the phone’s built-in software for using the camera and displays a preview image on the screen along with controls for adjusting the zoom level, exposure and flash etc. Control is not returned to the TouchDevelop script until the button to take the picture has been pressed and the picture taken.

All methods (except for invalid and post to wall) are listed in Table 6-2.

The user has the option of cancelling the picture capture in the high-quality version. Therefore the normal pattern of usage in a script should be code like the following.

To determine whether the phone has a front facing camera (a secondary camera), a script should invoke the senses → front camera method. If the
result is the invalid value, then this camera is absent.

Table 6-2: Methods of the Camera datatype

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>height : Number</td>
<td>Returns the height of the camera image in pixels.</td>
</tr>
<tr>
<td>is front : Boolean</td>
<td>Returns true if this is the camera facing the user, false if this is the primary camera.</td>
</tr>
<tr>
<td>preview : Picture</td>
<td>Takes a low quality picture with the camera, returning the picture immediately.</td>
</tr>
<tr>
<td>width : Number</td>
<td>Returns the width of the camera image in pixels.</td>
</tr>
</tbody>
</table>

```javascript
var pic := senses → take camera picture
if not pic → is invalid then
  // use the picture pic
  ...
else
  // user cancelled the picture
```

6.1.1 A sample program

The script *the poor man's camcorder (ptxfa)* asks you to sweep the camera slowly around while it takes 10 pictures in succession using the preview method of the Camera type. It then plays back the 10 pictures, giving the effect of a camcorder whose recordings are 2 seconds long.

A simplified and reprogrammed version of this script is presented in Figure 6-1.

This sample program also illustrates the use of an objects declaration in the Records section of a script which is used to create a data structure known as a cyclic list. It is a linked-list where every element of the list contains a reference to the next element, except that the last element references the first element, creating a cycle.
action main( number of pics: Number )
   // number of pics is the number of pictures to take and
   // display repeatedly; the recommended value is 10
   if number of pics ≤ 0 then
      number of pics := 10
   else
      // do nothing
      var camera := senses → camera

   var firstpic := PicList → create
   firstpic → pic → set( camera → preview )
   var prev := firstpic
   for 0 ≤ i < number of pics – 1 do
      var newpic := PicList → create
      newpic → pic → set( camera → preview )
      prev → next → set(newpic)
      prev := newpic
   // Make last pic in list point to first pic, i.e. a cyclic list
   prev → next → set(firstpic)

   // Go around the cyclic list forever
   prev := firstpic
   while true do
      wall → clear
      prev → pic → get → post to wall
      time → sleep( 0.2 )
      prev := prev → next → get

records PicList objects
   pic: Picture
   next: PicList

6.2 Manipulating pictures

The phone holds various picture albums with names such as Camera Roll and Saved Favorites. The API call

phone → picture albums
retrieves all the albums currently maintained on the phone, while the two calls

\begin{verbatim}
    phone → pictures
    phone → saved pictures
\end{verbatim}

return collections of all pictures, and the pictures held in the album named ‘saved pictures’, respectively. The methods for working with the Picture Album, Picture Albums and Pictures (a Picture collection) datatypes are listed in Table 6-3.

**Table 6-3: Methods of Picture Album and Pictures datatypes**

<table>
<thead>
<tr>
<th>Methods of Picture Album Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>albums : Picture Albums</td>
<td>Returns a collection of all the nested albums held inside this album.</td>
</tr>
<tr>
<td>name : String</td>
<td>Obtains the name of the album.</td>
</tr>
<tr>
<td>pictures : Pictures</td>
<td>Returns a collection of all the pictures held in the album.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods of Pictures Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find(name : String) : Number</td>
<td>Returns the index of a picture in the collection which has the given name; the result is -1 if the picture cannot be found.</td>
</tr>
<tr>
<td>random : Picture</td>
<td>Returns a random picture.</td>
</tr>
<tr>
<td>thumbnail(index : Number) : Picture</td>
<td>Returns a thumbnail of the picture at the given index position in the collection.</td>
</tr>
</tbody>
</table>

Once a Picture value has been obtained, perhaps by retrieving it from a collection, or by using the camera, there are many methods for manipulating the picture before it is displayed on the screen. These are covered in later sections of this chapter. Display of an image uses the post to wall method, as in this example.

\begin{verbatim}
    var pic1 := media → choose picture
    pic1 → post to wall
\end{verbatim}

An alternative way to display a picture is to set the wall’s background image, as follows.
\textbf{var} pic1 := media \rightarrow \text{choose picture}
\text{wall} \rightarrow \text{set background picture(pic1)}

The general-purpose methods of the Picture type are listed in Table 6-4.

\textbf{Table 6-4: General Picture methods}

<table>
<thead>
<tr>
<th>Methods of Picture Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{at(index: Number): Color}</td>
<td>Returns the color of the pixel at the given index in the picture</td>
</tr>
<tr>
<td>\text{clone : Picture}</td>
<td>Returns a copy of the Picture</td>
</tr>
<tr>
<td>\text{count : Number}</td>
<td>Returns number of pixels</td>
</tr>
<tr>
<td>\text{crop(left : Number, top : Number, width : Number, height : Number) : Nothing}</td>
<td>Crops the picture to a rectangular portion of the original.</td>
</tr>
<tr>
<td>\text{date : DateTime}</td>
<td>Returns a date associated with the picture (if there is one)</td>
</tr>
<tr>
<td>\text{flip horizontal : Nothing}</td>
<td>Flips the picture left to right</td>
</tr>
<tr>
<td>\text{flip vertical : Nothing}</td>
<td>Flips the picture top to bottom</td>
</tr>
<tr>
<td>\text{height : Number}</td>
<td>Returns height of picture in pixels</td>
</tr>
<tr>
<td>\text{is panorama : Boolean}</td>
<td>Returns true if width &gt; height</td>
</tr>
<tr>
<td>\text{location : Location}</td>
<td>Returns a location associated with the picture (if there is one)</td>
</tr>
<tr>
<td>\text{pixel(left : Number, top : Number) : Color}</td>
<td>Gets the pixel color at the specified x,y position</td>
</tr>
<tr>
<td>\text{post to wall : Nothing}</td>
<td>Displays the picture</td>
</tr>
<tr>
<td>\text{resize(width : Number, height : Number) : Nothing}</td>
<td>Scales the picture to have a new width and height</td>
</tr>
<tr>
<td>\text{save to library : String}</td>
<td>Stores the picture in the Saved Pictures album and returns the filename</td>
</tr>
<tr>
<td>\text{set pixel(left : Number, top : Number, color : Color) : Nothing}</td>
<td>Sets the pixel color at the specified x,y position</td>
</tr>
<tr>
<td>\text{update on wall : Nothing}</td>
<td>If this picture has been displayed and then changed, this method replaces the displayed image with the new one</td>
</tr>
<tr>
<td>\text{width : Number}</td>
<td>Returns width of picture in pixels</td>
</tr>
</tbody>
</table>

The table omits methods which change the colors or the brightness, or methods which overlay shapes etc. on top of the picture. All such methods are covered in the following subsections of this chapter.
6.2.1 Care in Using the at, pixel and set pixel methods

Table 6-4 includes the at, pixel and set pixel methods. Before any of these methods is used in a script, some thought should be given to how large the picture is.

Any TouchDevelop script which accesses every pixel of a picture can be exceedingly slow to run, as well as draining the phone’s battery. The implication is that the Picture type’s at, pixel and set pixel methods should be used only on pictures containing a modest number of pixels. Pictures taken by the camera contain as many pixels as the camera’s resolution. For example, one model of phone is advertised as having an 8 megapixel camera. Pictures downloaded from the internet or copied from your computer when synchronizing may contain even more pixels.

Although a picture shown on the screen is scaled to fit within the screen’s size, which is 800 by 480 pixels, the picture still has its original number of pixels in the phone’s memory.

Unless the script will be storing the picture back in the Saved Pictures album to be later copied to another device or printed, it would usually be appropriate to reduce the picture’s resolution to match the screen size. Note that any method which processes all the pixels in a single call, such as resize, is reasonably fast.

The at method is useful for determining various aggregate properties of a picture, such as its average brightness. In more sophisticated scripts, the pixel method could for example be used for analyzing a picture and extracting details such as edges or, when set pixel is used too, for sharpening edges.

An example script which computes a picture’s average brightness is shown in Figure 6-2. Each pixel in the picture has a color value composed from red (R), green (G) and blue (B) components whose values range from zero intensity or 0.0 up to the maximum intensity which is 1.0. From the R,G,B values of a pixel, its luminosity can be calculated. (See, for example, the explanation of the YUV color space and the conversion formula for computing YUV values in Wikipedia.) The luminosity is a measure of the brightness of that pixel.
The at and pixel methods are similar because they both retrieve the color of a particular pixel. Generally speaking, the at method should be used when it does not matter where the pixel is located within the picture, as is the case for the brightness calculation in Figure 6-2. It provides more efficient access because only one for loop to access all the pixels is needed. The pixel and set pixel methods would normally be placed inside two nested for loops, one to run through the rows and the other through the columns. The equivalence between the two ways to access a particular pixel is as follows.

\[
\text{pic1} \rightarrow \text{pixel(x,y)} \equiv \text{pic1} \rightarrow \text{at(y*pic1} \rightarrow \text{width + x)}
\]

Note that y coordinate values are measured from the top edge of the picture down. It is the opposite convention to that used in geometry.

### 6.2.2 Picture colorizing effects

The colors, the contrast and the brightness of a picture can all be modified using more methods of the Picture type which are listed in Table 6-5.

The brightness method can be used to increase or decrease the luminosities of all the pixels in the image in unison, so that the picture appears brighter or darker. The contrast method can be used to increase or decrease the range of luminosities, so that there is greater or smaller contrast between light and dark regions.

The colorize method is intended for creating a two color image from a greyscale image. All pixels darker than a specified threshold value (a number in the range 0.0 to 1.0) are replaced by the background color, while all those

---

**Figure 6-2: Computing brightness**

```plaintext
// use this only for low resolution pictures!
action compute brightness( pic : Picture ) returns avg y : Number
  var sum y := 0
  for 0 ≤ i < pic → count do
    var c := pic → at(i)
    // compute y = luminosity of pixel (i.e. brightness level)
    var y := c → R * 0.299 + c → G * 0.587 + c → B * 0.114
    sum y := sum y + y
  avg y := sum y / pic→ count
```
brighter are replaced by the foreground color. The method can also be applied to color images, but that image is converted to grayscale before the colorization is applied.

Table 6-5: Colorizing/intensity picture effects

<table>
<thead>
<tr>
<th>Methods of Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brightness(factor : Number) : Nothing</td>
<td>Increases of decreases the brightness of the picture. The parameter ranges from -1 to +1.</td>
</tr>
<tr>
<td>colorize(background : Color, foreground : Color, threshold : Number) : Nothing</td>
<td>Changes the picture to a two color scheme. Pixels darker than threshold become the background color, those brighter become the foreground color.</td>
</tr>
<tr>
<td>contrast(factor : Number) : Nothing</td>
<td>Increases or decreases contrast level of the picture. The parameter ranges from -1 to +1.</td>
</tr>
<tr>
<td>desaturate : Nothing</td>
<td>Converts the picture to grayscale.</td>
</tr>
<tr>
<td>invert : Nothing</td>
<td>Inverts the intensity of each of the R, G, B color components</td>
</tr>
<tr>
<td>tint(color : Color) : Nothing</td>
<td>Converts the picture to grayscale, and then tints with the supplied color</td>
</tr>
</tbody>
</table>

The final picture will no longer have any variations in intensity. All pixels of the foreground color have the same intensity, as do all those with the background color.

The invert method produces a result similar to a color negative, as would be observed with a 35mm camera using chemically developed color film. (This is a type of camera which is becoming rare.)

6.2.3 Picture overlaying

The next section in this chapter, section 6.3, is all about drawing text, lines and various shapes on top of a picture. What about superimposing another picture on top of a picture? That facility is provided by the blend method. It is called blend as opposed to ‘superimpose’ say, because one of the method’s parameters chooses the opacity of the overlaid image. By choosing a low degree of opacity, the image at the bottom can be seen through the image on the top – achieving a blending of the two images.

The following few lines of code illustrate the concept.
The relationship of the two pictures to each other is illustrated in Figure 6-3.

![Figure 6-3: Blending two pictures](image)

The top left corner of \( \text{pic2} \) is located at the x,y coordinates given by the second and third arguments to \( \text{blend} \). This picture is rotated clockwise by the number of degrees given by the fourth argument. The opacity of the picture has been set at 0.7, which means that in the overlaid region, each pixel is a blend of 70% from \( \text{pic2} \) and 30% from \( \text{pic1} \). Finally, the bottom right of \( \text{pic2} \) has been clipped to fit within \( \text{pic1} \).

### 6.3 Static graphics drawing and display

A picture can be a photograph, a drawing, or a combination of the two. The methods provided by the Picture datatype for drawing lines and shapes are listed in Table 6-6.
Table 6-6: Drawing methods of the Picture datatype

<table>
<thead>
<tr>
<th>Methods of Picture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear(color : Color) : Nothing</td>
<td>Sets all pixels to the given color</td>
</tr>
<tr>
<td>draw ellipse(left: Number, top: Number, width: Number, height : Number, angle : Number, c: Color, thickness: Number) : Nothing</td>
<td>Draws an ellipse; its bounding rectangle has the width and height provided and is located at the specified position; its orientation is rotated clockwise by the angle given; the line has the color and thickness specified.</td>
</tr>
<tr>
<td>draw line(x1: Number, y1: Number, x2: Number, y2: Number, color: Color, thickness : Number) : Nothing</td>
<td>Draws a line from x1,y1 to x2,y2; the line has the color and thickness specified.</td>
</tr>
<tr>
<td>draw rect(left : Number, top : Number, width : Number, height : Number, angle : Number, c : Color, thickness : Number) : Nothing</td>
<td>Draws a rectangle which has the width and height provided and is located at the specified position; its orientation is rotated clockwise by the angle given; the line has the color and thickness specified.</td>
</tr>
<tr>
<td>draw text(left : Number, top : Number, text : String, font size : Number, angle : Number, color : Color) : Nothing</td>
<td>Writes text at the position specified, at the given font size and in the specified color; the text is rotated clockwise by the angle given</td>
</tr>
<tr>
<td>fill ellipse(left : Number, top : Number, width : Number, height : Number, angle : Number, color : Color) : Nothing</td>
<td>Like draw ellipse except that it is a solid (filled) ellipse.</td>
</tr>
<tr>
<td>fill rect(left : Number, top : Number, width : Number, height : Number, angle : Number, color : Color) : Nothing</td>
<td>Like draw rect except that it is a solid (filled) rectangle.</td>
</tr>
</tbody>
</table>

In addition to these, there is the set pixel method which has already been covered and also the create picture method of the media resource for creating a new empty picture as illustrated below. All pixels in the new picture are colored white.
// create picture 400 pixels wide, 200 pixels high
var pic := media -> create picture( 400, 200 )

Figure 6-4 shows how the parameters of the draw ellipse method control the placement and orientation of the ellipse within a picture. An ellipse fits within a bounding rectangle, as drawn, and it is the width and height of that rectangle which are specified as arguments of the method.

The left and top parameters provide the x, y coordinates of the top left corner of that bounding rectangle. Note that y coordinate values are measured downwards from the top of the picture.

A circle is a special case of an ellipse. Most graphics drawing software would draw a circle of a given radius and with its center at a specified position. To use the draw ellipse method, some extra arithmetic is needed. An action which accepts radius and the position of the center would be programmed as follows.

// Draws a circle with radius r and center at x, y.
action Draw Circle( pic: Picture, r: Number, x: Number, y: Number, thickness: Number, color: Color )
    pic -> draw ellipse( x-r, y-r, r*2, r*2, 0, color, thickness )

Drawing rectangles and filled versions of ellipses and rectangles is handled
similarly; further explanation should not be needed.

6.4 Playing videos from the internet

Video files tend to be very large. The phone’s memory can therefore hold only short videos. Videos held on the phone can be created by using the phone’s camera or by copying from a PC when the phone is synchronized using Zune.

TouchDevelop does not provide access to video files held on the phone. Nor can a script download a video file to the phone. However a script can access and play videos which are streamed from the internet or from a home media server. Use of a home media server is covered in the next section of the chapter.

Given the URL of a video file which is in a format that the phone can play, a TouchDevelop script can open and play that file. The supported video formats depend on the model of phone. However most video files with the filename suffix `.mp4` should work. (An H.264 encoded MP4 file is the format which works on every Windows phone.)

Given the URL, a direct way to play the video is as follows.

```csharp
// url is a String variable holding the URL of the file
web → play media( url )
```

It displays the video using the entire screen. The phone’s back button can be used to stop the video.

An alternative means of playing the video is to create a Link value. Some sample code is as follows.

```csharp
// url is a String variable holding the URL of the file
var lnk := web → link media( url )
...
if not lnk → is invalid then
   lnk → post to wall
   “Tap video link to play it” → post to wall
else
   (“Could not access url ” || url) → post to wall
```
The second approach shown above allows a script to show a list of links on the screen, allowing the user to select which one to play.

6.5 Media Player – photos and videos

Chapter 5 included a section on accessing a home media player to stream audio to the phone over a Wi-Fi network. Most media players are also capable of sending photographs to the phone and to stream videos to the phone.

Following the same steps described in Chapter 5, a media server must first be selected. Then, given that instance, one of the methods listed in Table 6-7 should be called.

Table 6-7: Media Server methods for pictures & videos

<table>
<thead>
<tr>
<th>Methods of Media Server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>choose picture : Media Link</td>
<td>Displays a list of all pictures on the server and allows the user to select one</td>
</tr>
<tr>
<td>choose video : Media Link</td>
<td>Displays a list of all videos on the server and allows the user to select one</td>
</tr>
<tr>
<td>search pictures by date(start : DateTime, end : DateTime) : Media Link Collection</td>
<td>Returns a list of all pictures on the server which have an associated date in the range which is provided</td>
</tr>
<tr>
<td>search videos(term : String) : Media Link Collection</td>
<td>Searches for videos whose names match the term supplied as the argument</td>
</tr>
<tr>
<td>search videos by date(start : DateTime, end : DateTime) : Media Link Collection</td>
<td>Returns a list of all videos on the server which have an associated date in the range which is provided</td>
</tr>
<tr>
<td>videos : Media Link Collection</td>
<td>Returns a list of all pictures on the server</td>
</tr>
</tbody>
</table>

A Media Link instance which references a photograph or a video can be obtained. Whether the link references a photograph or a video, the link can be displayed and then the photograph is displayed or the video is played if the link is tapped. The code might be the following.

```java
// mlnk refers to a value of the Media Link type
mlnk → post to wall
"Tap the link when ready" → post to wall
```

Alternatively, tapping a link displayed on the screen can be omitted if the
following statement is executed by the script.

```plaintext
// mlnk refers to a value of the Media Link type
mlnk → play
```

This again works regardless of whether the link references a photograph or video.

A sample program very similar to that given in Chapter 5, but modified to play a video, appears in Figure 6-5.

**Figure 6-5: The play video script (/jccc)**

```plaintext
action main( name: String )
  wall → set reversed(true)
  var server := home → choose server
  var link := server → search videos(name)
  // link is a collection of the matching videos
  if link → count > 0 then
    "Select video to play" → post to wall
    // display the collection, each item has a play/pause
    // button, the user can select which one to play
    link → post to wall
  else
    "No videos found" → post to wall
```

One feature of the previous script had to be dropped, however. It is not possible to stop the video playing from within the script. The user has to tap the screen and a pause/continue button will appear. Pausing the video, and tapping the phone's back button causes the script to stop execution.