CHAPTER 3
TOUCHDEVELOP LANGUAGE BASICS

Using the techniques described in the Chapter 2, you can input and edit scripts which perform actions on your Windows Phone. The script is just like a computer program and we can discuss the scripting language in terms of common programming language concepts. In this chapter we give a broad overview of the main features of the TouchDevelop scripting language by looking at an example.

3.1. A SAMPLE TOUCHDEVELOP SCRIPT

A fairly simple small script is shown in Figure 3.1. The script looks at the songs on the phone and posts the names of the songs that have not been played yet. At the top level, a script is composed of actions, events and global data.

- An **ACTION** is just like a function in C or a method in Java or C#. It can take parameters, it can return results generated by some calculations, and it contains statements. The sample script has two actions named ‘main’ and ‘display song’.
- An **EVENT** contains statements which are executed whenever the event occurs. The sample script has two events named ‘shake’ and ‘active song changed’. The first event is triggered whenever the phone is shaken, the second event is triggered whenever our script starts playing a new song.
- **GLOBAL DATA** can be used to remember information between events happening or while the script is inactive (perhaps while taking a phone call). The script of Figure 3.1 shows a global data item named ‘played’. It does not appear in the code listing on the phone itself – it has to be viewed separately from the code.
The statements used inside the actions and events of the sample script should look familiar to anyone who has written programs before. Let’s quickly run through what the script does when it is executed.

Figure 3.1: The ‘new songs’ Script

```
action main()
    Finds songs not played yet.
    var found := 0
    var songs := media→.listdir
    for each song in songs
        where true
        do
            found := found + ▷display song(song)
            ("Songs played with this script: " || △played)
            →post to wall
            ("Songs never played: " || found)
            →post to wall

action display song(song) returns result
    Post a song to the wall if not played yet and returns 1;
    otherwise returns 0.
    if song→play count = 0 then
        song→post to wall
        result := 1
    else
        result := 0

event active song changed()
    Increment the song played counter.
    △played := △played + 1

event shake()
    Pauses and resumes playing.
    if player→is playing then
        player→pause
    else
        player→resume
```
STARTING THE ACTION AND DECLARING LOCAL VARIABLES

1. The script begins with the action named ‘main’. It must be this action because it is the only one not flagged as ‘private’. (If there are two or more public actions, then the one called ‘main’ would be chosen; if none is called ‘main’, the alphabetically first will be chosen.)
2. The main action declares a local variable named ‘found’ and initializes it to 0. This initialization causes ‘found’ to always be a variable with the type ‘Number’.
3. Next another local variable named ‘songs’ is declared. It is initialized to be a collection of all the songs (i.e. music files) that are held on the phone. The service named ‘media’ is accessed from the TouchDevelop library; this service is an object with various properties and methods which allow a TouchDevelop script to access songs, song albums, song playlists, pictures, picture albums (as well as a few other things). The expression media→songs retrieves a collection of all the songs on the phone.

ITERATING OVER ALL SONGS IN THE COLLECTION

4. Now a for each loop accesses each element of the songs collection in turn and assigns it to a new local variable named ‘song’.
5. The body of the for each loop invokes the action named ‘display song’ passing it ‘song’ as a parameter. The result of the call is either the number 1 or the number 0 – it is 1 if the song’s play count is zero and 0 otherwise. Hence the assignment increments the local variable ‘found’ each time the loop encounters a song which has never been played.
6. When the loop is finished, the local variable ‘found’ contains a count of how many songs on the phone have never been played.

DISPLAYING SOME MESSAGES

7. The next statement after the loop contains the expression "Songs played with this app: " || played. That expression constructs a string by obtaining the value of the global data item named played (its value will be the integer 0 when the script is invoked for the first time), converting it to a string, and concatenating it after the end of the string constant "Songs played with this app: ". The constructed string is now displayed on the phone’s screen. We can understand that action as being a result of the method call ‘post to wall’; it is a method possessed by the ‘String’ type.
8. Finally, the script similarly constructs a string value from the expression "Songs never played: " || found and displays that on the phone’s screen too.

WAITING FOR THE USER TO TERMINATE THE PROGRAM

9. After executing that last statement, the script keeps running and waits for events to occur; the script execution does not finish. (This is true for any script which declares an event; if
there were no events declared, the script would stop execution.) The script can be halted by tapping the phone back button.

10. Now, what did the ‘display song’ action do? As well as looking up the ‘play count’ property of the song passed in as a parameter, it also invokes the ‘post to wall’ method on each song whose play count is equal to zero. As one would expect, the ‘post to wall’ method displays the name of the song. However it also displays additional information about the song and it displays a button named ‘play’. Figure 3.2 shows the details and play buttons for two songs as they might be listed on the screen when the script has finished checking all the songs held on the phone.

**EVENT HANDLING CODE**

11. If, while the script is active, we shake our phone, the ‘shake’ event is activated. The code inside this event first tests to see if the phone is currently playing any music. If it is, the music is paused and if not, the playback of any paused music is resumed.

12. How do we play a song while the script is active? We simply have to tap one of the ‘play’ buttons which have been displayed. As well as starting the playback of that song, the ‘active song changed’ event is triggered in our script. Its only action is to increment the global data variable named ‘played’.

![Figure 3.2: Songs as Displayed on the Screen](image)

3.2. **WHAT IS A TOUCHDEVELOP SCRIPT COMPOSED OF?**

A TouchDevelop script contains four main components which are **CODE**, **EVENTS**, **DATA** and **ART**. The script also has access to resources on the phone and on the internet which are made available through the TouchDevelop API (short for ‘Application Program Interface’).

The code component consists of actions. A reference to a particular action in the program is made with a triangle symbol ▸ used like this
An action with or without parameters can be invoked from another action in the script or from the code for an event. An action which has not been marked as PRIVATE can be called directly from the TouchDevelop environment to start up execution of the script with that action. (If parameters are needed for that initial action, the user will be prompted for them.)

Events are similar to actions except that they are invoked by something happening outside the script, such as the phone being shaken.

The data component consists of global variables, declared externally to any action or event in the script. The example script seen in Figure 3.1 has one such variable named ‘played’. Access to the variable in actions and events uses a special symbol △ to indicate that a global data item is coming, like this

△played

A global data variable can have any of the datatypes listed in the TouchDevelop API. This list includes the simple types as well as a rich selection of composite types which are useful for accessing the features of a Windows phone.

The Art component consists of pictures and sounds which are special kinds of data values used by the script (and which can be shared with other scripts).

Services available to a script include math, media, senses and maps. A more complete list will be covered in the following chapters. These services are similar to classes in Java or C# which have only static members. Access to the methods and properties of these services is obtained via an arrow notation. Whereas languages like Java or C# would use a dot to separate an object or class from a method name, TouchDevelop uses an arrow for its better visibility on the screen. Some examples of the arrow notation are as follows.

math→sin(0.5)
math→π
media→choose picture
senses→camera

For example, the script shown in Figure 3.1 uses the expression media→songs to retrieve a collection of all the songs held on the Windows phone.

### 3.3. SOME LANGUAGE BASICS

For anyone who has programmed in a modern language such as Java, C# or Python, Table 3.1 summarizes the main features of the scripting language. More details will be provided in this and later chapters.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| Type System    | • The language uses *static typing* for variables (i.e. a variable has a datatype which cannot change as the script runs).  
  • The language uses *strict type checking* (i.e., any attempt to apply an operation to a value of the wrong type is reported as an error before the script begins execution). |
| Datatypes      | • The types include Number, Boolean, DateTime and String.  
  • There are many other datatypes provided by the TouchDevelop API.  
  • The Number type contains both integers and real numbers (i.e. IEEE floating point numbers).  
  • Some types, such as Number, are known as *value types* while others, such as the String Collection type, are known as *reference types*. (This is terminology borrowed from C#.) The distinction between value types and reference types, and its relevance to how assignments and parameter passing work, are explained below.  
  • All datatypes contain a special value called ‘invalid’ which indicates an error. |
| Variables      | A local variable is declared with an assignment of an initial value. That value determines the variable’s datatype. |
| Variable’s Scope | • A variable is accessible from its point of declaration down to the end of the enclosing block of statements.  
  • A parameter or a result variable is visible throughout the body of an action. |
| Actions        | • Actions are similar to procedures and functions in other languages. They accept zero or more arguments and they return zero or more result values.  
  • The parameter passing mechanism is similar to that used in the Java and C# languages. It is explained in more detail below.  
  • An action can return zero or more result values. Result variables can be named as a property of the action and the final values of these variables are used as the results.  
  • Result variables are initialized with the ‘invalid’ values of their respective types. |
<p>| Statement Types | A statement can be an expression which is simply evaluated for its side-effects (such as an invocation of an action or displaying a value on the screen). Other forms of statement are: assignment statement, if statement, for each loop, for loop, and while loop. |
| Assignment statement | The assignment operator is ‘:=’. It can assign multiple values to several variables if the right-hand expression is a call to an action which returns multiple results. An assignment copies either a <em>value</em> or a <em>reference to a value</em> to a left-hand side varia- |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If statement</td>
<td>Uses an expression of Boolean type to select between a <em>then</em> clause and an <em>else</em> clause (which may be empty).</td>
</tr>
<tr>
<td>For each loop</td>
<td>A loop which steps through each value in a collection. The current value being accessed is held by the loop’s index variable.</td>
</tr>
<tr>
<td>For loop</td>
<td>A loop which has an index variable that steps through an increasing sequence of integer values starting at 0 and continuing up to but not including a specified limit value.</td>
</tr>
<tr>
<td>While loop</td>
<td>A loop controlled by an expression which is re-evaluated on each iteration and must yield true for one more iteration of the loop body to be performed.</td>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>These are the usual operators + − * and /</td>
</tr>
</tbody>
</table>
| Comparison Operators | • For numbers, the comparison operators are < ≤ = ≠ ≥ >  
|                      | • To compare strings, the compare or equals method must be used.                                                                                 |
| String Operators | The only operator is || and is used for string concatenation. It can be used with non-string operands too; they will be converted to the String type before concatenating. |
| Boolean Operators | The Boolean operators are and, or, and not.                                                                                                         |
| Events        | • Statements can be provided which execute whenever a particular event occurs. (For example, shaking the phone or changing the phone’s orientation triggers events.)  
|              | • The code for an event will be completed before control is returned to an action or before another event is executed. Events are executed in the same order as they occur. |

3.4. **VALUE TYPES VERSUS REFERENCE TYPES**

TouchDevelop implements datatypes in a similar manner to Java and C#. Therefore the following explanation should be unnecessary for anyone who has programmed in these languages.
Consider the following short fragment of code which manipulates values of type Number.

```plaintext
var num1 := 99
var num2 := num1
num2 := num2 + 1
num1→post to wall  // displays 99
num2→post to wall  // displays 100
```

The numbers displayed, 99 and 100, are as you would expect. When `num2` was incremented, it had no effect on the value of `num1`. That is because `num1` and `num2` hold two distinct Number values.

Now consider the following fragment of code.

```plaintext
var list1 := collections→create string collection // assign an empty list of strings to list1
list1→add("Hello") // add one string to the list
var list2 := list1
list2→add("Goodbye") // add one string to the list
list1→post to wall // displays Hello;Goodbye
->list2→post to wall // displays Hello;Goodbye
```

Here, when we added one more string to the `list2` collection, it changed the `list1` collection too. How is that possible? The explanation is that the script created only one string collection, doing that in the first line. The assignment `line1 := line2` causes these two variables to both refer to the same list of strings.

To understand the scripting language fully, it is important to know when an assignment is copying a value and when it is copying a reference to a value. Fortunately, there are many datatypes where the distinction simply does not matter. These are the immutable types. For example, String is an immutable type. When a String value is copied, as in this code fragment

```plaintext
var s1 := "abc"
var s2 := s1
```

there is no way to alter the string instance referenced by `s2`. String values cannot be changed. We can assign `s2` a new string value, as in

```plaintext
s2 := s2 || "def"
```

but that simply creates a new string and assigns a reference to that new string to `s2`. The other variable, `s1`, is completely unaffected.

Finally, we should observe that when an action is invoked, the caller supplies values which are assigned to the parameters of the action. Whether the action can affect the values being passed depends on whether the values are mutable and, if they are, whether the types of these parameters are value types or reference types.
3.5. THE NOTHING TYPE AND THE INVALID VALUE

The name Nothing represents a datatype similar to void in C# or Java. It is used to describe the result datatype of a method which is invoked purely for its side-effects and returns no useful result. For example, the statement

   player→next

causes the music player to move to the next song in a list, and no useful result is returned by that invocation. The type of the next method of the player service is shown as NOTHING in the API documentation.

All types have a special value (or be in a special state) which is described as INVALID in the API documentation. For example, the statement

   pic := senses→take camera picture

should cause the phone’s camera to take a picture and assign that picture to the variable pic. However the user has an opportunity to cancel the camera capture so that pic does not receive a valid picture as the result. The script can (and should) test for this situation by testing whether the expression

   pic→is invalid

has a true or false value. It would not be incorrect to consider the INVALID value to be similar to null for an object reference in a language like C# or Java.

3.6. THE WEBSITE VIEW OF A SCRIPT

The current version of the TouchDevelop website shows scripts in a slightly different format from those seen on the phone and in this book. It is expected that the displayed formats for scripts on the website will be changed in the near future to be much closer to the format used on the Windows phone. The differences are very minor, and you should not have any trouble reading scripts on the website.