APPENDIX A

Code examples for controlling LED using Visual Basic and JavaScript

Here you find additional examples for controlling the LED circuit you developed in Chapter 2, "Universal Windows Platform on devices." The following sample applications offer alternative ways for accessing the UWP and do not show any additional functionality.

Headed app using Visual Basic/XAML

The implementation of the IoT Hello, World! app proceeds as follows:

1. Open the New Project dialog box.
2. In the New Project dialog box:
   a. Type **Visual Basic** in the search box.
   b. Select the **Blank App (Universal Windows)** project template.
   c. Change the app name to **HelloWorldIoTVB** and click the **OK** button.
3. Open MainPage.xaml.vb and edit it according to Listing A-1.
4. Reference Windows IoT Extensions for the UWP.
5. Change the target platform to **ARM** and point to your IoT device as the remote machine. (For help, see the procedure in the section "C#/XAML" in Chapter 2.)
6. Start debugging. The application will be deployed and executed on your IoT device.

LISTING A-1 Visual Basic implementation of LED control

```
Imports Windows.Devices.Gpio

Public NotInheritable Class MainPage
    Inherits Page

    Private Const gpioPinNumber = 5
    Private Const msShineDuration = 4000
```

```
    Protected Overrides Sub OnNavigatedTo(e As NavigationEventArgs)
        MyBase.OnNavigatedTo(e)

        BlinkLed(gpioPinNumber, msShineDuration)
    End Sub

    Private Function ConfigureGpioPin(pinNumber As Integer) As GpioPin
        Dim gpioControl = GpioController.GetDefault()
        Dim pin As GpioPin = Nothing

        If gpioControl IsNot Nothing Then
            pin = gpioControl.OpenPin(pinNumber)

            If pin IsNot Nothing Then
                pin.SetDriveMode(GpioPinDriveMode.Output)
            End If
        End If

        Return pin
    End Function

    Private Sub BlinkLed(gpioPinNumber As Integer, msShineDuration As Integer)
        Dim pin = ConfigureGpioPin(gpioPinNumber)

        If pin IsNot Nothing Then
            pin.Write(GpioPinValue.Low)

            Task.Delay(msShineDuration).Wait()

            pin.Write(GpioPinValue.High)
        End If
    End Sub
End Class
```

The structure of the Visual Basic UWP project is very similar to its C# counterpart. Namely, the `MainPage` view is implemented within two files: MainPage.xaml and MainPage.xaml.vb. The latter implements logic, which in the preceding example turns on an LED for 4 seconds. The general flow of the application is the same as in the case of C# and C++ projects. That is, you first obtain a reference to the instance of `GpioController` class, which is subsequently used to open the GPIO port. The latter is set to a low state to power a LED, and to a high state to turn off a diode, assuming an LED circuit is assembled in an active-low state.

Headless app using Visual Basic/XAML

To implement the background IoT application using Visual Basic, follow these steps:

1. Create the new IoTBackgroundAppVB app using the Background Application (IoT) project template. You can find this template by typing **Visual Basic IoT** in the search box of the New Project dialog box.

2. Reference Windows IoT Extensions for the UWP.
3. Modify the StartupTask.vb file according to Listing A-2.
4. Deploy an app to your IoT. An LED will start blinking.

LISTING A-2 The contents of a StartupTask.vb file

```
Imports Windows.ApplicationModel.Background
Imports Windows.Devices.Gpio

Public NotInheritable Class StartupTask
    Implements IBackgroundTask

    Private Const gpioPinNumber = 5
    Private Const msShineDuration = 4000

    Public Sub Run(taskInstance As IBackgroundTaskInstance) Implements
IBackgroundTask.Run
        BlinkLed(gpioPinNumber, msShineDuration)
    End Sub

    Private Function ConfigureGpioPin(pinNumber As Integer) As GpioPin
        Dim gpioControl = GpioController.GetDefault()
        Dim pin As GpioPin = Nothing

        If gpioControl IsNot Nothing Then
            pin = gpioControl.OpenPin(pinNumber)

            If pin IsNot Nothing Then
                pin.SetDriveMode(GpioPinDriveMode.Output)
            End If
        End If

        Return pin
    End Function

    Private Sub BlinkLed(gpioPinNumber As Integer, msShineDuration As Integer)
        Dim ledGpioPin = ConfigureGpioPin(gpioPinNumber)

        If ledGpioPin IsNot Nothing Then
            While True
                SwitchGpioPin(ledGpioPin)
                Task.Delay(msShineDuration).Wait()
            End While
        End If
    End Sub

    Private Sub SwitchGpioPin(gpioPin As GpioPin)
        Dim currentPinValue = gpioPin.Read()
        Dim newPinValue = InvertGpioPinValue(currentPinValue)

        gpioPin.Write(newPinValue)
```

```
    End Sub

    Private Function InvertGpioPinValue(currentPinValue As GpioPinValue) As
GpioPinValue
        Dim invertedGpioPinValue As GpioPinValue

        If currentPinValue = GpioPinValue.High Then
            invertedGpioPinValue = GpioPinValue.Low
        Else
            invertedGpioPinValue = GpioPinValue.High
        End If

        Return invertedGpioPinValue
    End Function
End Class
```

Headed app using JavaScript, HTML, and CSS

Implement the headed app as follows:

1. Open the New Project dialog box.
2. Type **JavaScript** in the search box, and then pick **Blank App (Universal Windows)** from the list of project templates.
3. Change the project name to **IoTBackgroundAppJS** and click the **OK** button.
4. Reference Windows IoT Extensions for the UWP.
5. Modify the default.js file according to Listing A-3.
6. Change the target platform to ARM.
7. Open the Project Properties dialog box and do the following:
   a. Change the Platform setting to **ARM**.
   b. Select **Remote Machine** from the Debugger to Launch drop-down list.
   c. Discover your IoT device using <Locate> in the Machine Name input box. See Figure A-1.
8. Start app debugging.

LISTING A-3 GPIO port control using JavaScript

```
(function () {
    "use strict";

    var app = WinJS.Application;
```

```
    var activation = Windows.ApplicationModel.Activation;

    var gpio = Windows.Devices.Gpio;
    var gpioPinNumber = 5;
    var msShineDuration = 1000;

    app.onactivated = function (args) {
        if (args.detail.kind === activation.ActivationKind.launch) {
            if (args.detail.previousExecutionState !==
                activation.ApplicationExecutionState.terminated) {
            } else {
            }
            args.setPromise(WinJS.UI.processAll());

            blinkLed(gpioPinNumber, msShineDuration);
        }
    };

    function configureGpioPin(pinNumber) {
        var gpioController = gpio.GpioController.getDefault();

        var gpioPin = null;

        if (gpioController) {
            gpioPin = gpioController.openPin(pinNumber);

            if (gpioPin) {
                gpioPin.setDriveMode(gpio.GpioPinDriveMode.output);
            }
        }

        return gpioPin;
    }

    function blinkLed(pinNumber, msShineDuration) {
        var ledGpioPin = configureGpioPin(pinNumber);

        if (ledGpioPin) {
            ledGpioPin.write(gpio.GpioPinValue.low);

            setTimeout(function () {
                ledGpioPin.write(gpio.GpioPinValue.high);
            }, msShineDuration);
        }
    }

    app.oncheckpoint = function (args) {
    };

    app.start();
})();
```

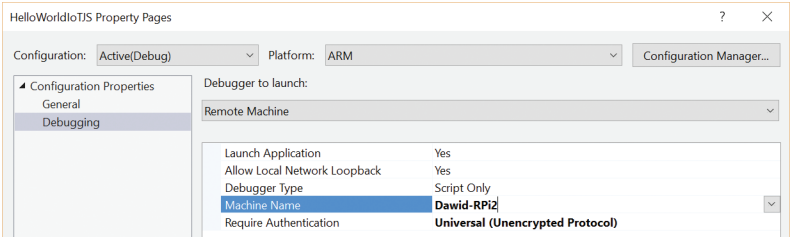

FIGURE A-1 JavaScript Universal Windows app debugging configuration.

As in the case of the Visual Basic sample, this application is deployed and executed to the IoT device. After a short while, the appropriate LED is turned on for 1 second.

Look at the project skeleton generated using JavaScript Blank App (Universal Windows). Besides the application manifest file, it contains the following elements:

- **Css folder** The storage for style sheets
- **Images folder** A template-provided location for image resources
- **Js folder** A storage for JavaScript files
- **WinJS folder** Stores the files of the JavaScript library for building HTML/CSS UWP applications
- **Default.html** UI declaration of the application main view (page)

By default, css and js folders contain one file each: default.css and default.js, respectively. Those files, together with default.html, implement the main page of the JavaScript UWP application.

I did not modify the UI declaration of `IoTBackgroundAppJS` but updated the default contents of the JavaScript file, default.js, which implements the logic of the view. It consists of the single anonymous function, which starts the UWP application and attaches event handlers to the application `onactivated` event handler. Within this handler, I call the `blinkLed` method, which as before (assuming an active-low state) configures the GPIO pin and drives it to the output-low state. Subsequently, the GPIO pin is driven to the output-high state to turn off the LED. To implement a delay between subsequent calls to a `write` method exposed by the `gpioPin` object, I use the `setTimeout` JavaScript function.

An entry point of the headed JavaScript app

I did not discuss the entry point of the Visual Basic app because it is analogous to the C# and C++ projects. On the other hand, JavaScript/HTML UWP applications are activated in a slightly different manner from XAML applications. They include no static `Program` class implementing the `Main` method. Instead, the hosting environment, which is the instance of the WWAHost.exe process, loads the start page indicated in the package.appxmanifest file. In the case of the HelloWorldIoTJS project, this file points to a page, defined within the default.html file. This configuration can be changed either by manually editing the appropriate entry in the manifest file, shown in Listing A-4, or by using Visual Studio package.appx-manifest editor (see Figure A.2).

LISTING A-4 Fragment of the HelloWorldIoTJS app manifest file

```
<Applications>
    <Application
      Id="App"
      StartPage="default.html">
      <uap:VisualElements
        DisplayName="HelloWorldIoTJS"
        Description="HelloWorldIoTJS"
        BackgroundColor="transparent"
        Square150x150Logo="images\Square150x150Logo.png"
        Square44x44Logo="images\Square44x44Logo.png">

        <uap:DefaultTile Wide310x150Logo="images\Wide310x150Logo.png"/>
        <uap:SplashScreen Image="images\splashscreen.png" />

      </uap:VisualElements>
    </Application>
</Applications>
```

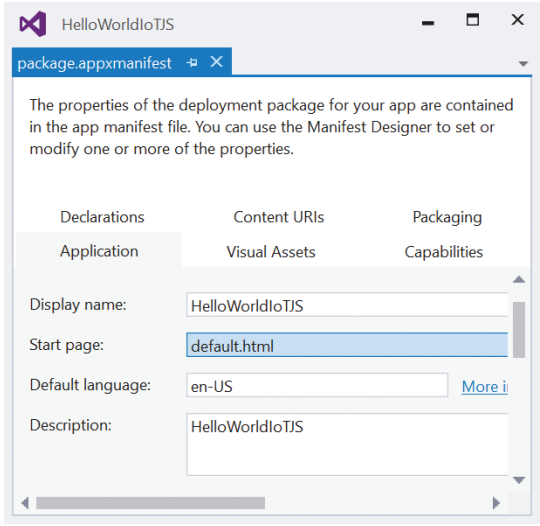

FIGURE A-2 The package.appxmanifest editor of the HelloWorldIoTJS application. The name of the file implementing the default application view is highlighted.

The automatically generated default.html file appears in Listing A-5. It contains typical HTML markup and is what WWAHost.exe loads and processes when the app is launched. When WWAHost.exe encounters referenced JavaScript files—in this case the WinJS library and the app's startup code in default.js (see Listing A-6)—it loads and then executes that code. Default.js is structured as a self-executing anonymous function for app initialization to retrieve the `Application` object and set up listeners for the `onactivated` and `oncheckpoint` events, which are fired when the app is launched and suspended, respectively.

LISTING A-5 The contents of the default.html file of HelloWorldIoTJS project

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>HelloWorldIoTJS</title>

    <!-- WinJS references -->
    <link href="WinJS/css/ui-dark.css" rel="stylesheet" />
    <script src="WinJS/js/base.js"></script>
    <script src="WinJS/js/ui.js"></script>

    <!-- HelloWorldIoTJS references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>
</head>
<body class="win-type-body">
    <p>Content goes here</p>
</body>
</html>
```

LISTING A-6 Default JS UWP app entry point

```
(function () {
    "use strict";

    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;

    app.onactivated = function (args) {
        if (args.detail.kind === activation.ActivationKind.launch) {
            if (args.detail.previousExecutionState !==
                activation.ApplicationExecutionState.terminated) {
            } else {
            }
            args.setPromise(WinJS.UI.processAll());
        }
    };

    app.oncheckpoint = function (args) {
    };

    app.start();
})();
```

Implement the headless JavaScript app by following these steps:

1. Open the New Project dialog box.
2. Type **JavaScript** in the search box, and then pick the **Background Application (IoT)** project template.
3. Change the project name to **IoTBackgroundAppJS** and click the **OK** button.
4. Reference Windows IoT Extensions for the UWP.
5. Modify startuptask.js according to Listing A-7.
6. Deploy an app to your IoT device. After a while, the LED will start blinking.

LISTING A-7 JavaScript implementation of the IoT background application

```
(function () {
    "use strict";

    var gpio = Windows.Devices.Gpio;
    var gpioPinNumber = 5;
    var msShineDuration = 1000;

    function configureGpioPin(pinNumber) {
        var gpioController = gpio.GpioController.getDefault();

        var gpioPin = null;

        if (gpioController) {
            gpioPin = gpioController.openPin(pinNumber);

            if (gpioPin) {
                gpioPin.setDriveMode(gpio.GpioPinDriveMode.output);
            }
        }

        return gpioPin;
    }

    function switchGpioPin(gpioPin) {
        var currentPinValue = gpioPin.read();
        var newPinValue = !currentPinValue;

        gpioPin.write(newPinValue);
    }

    function blinkLED(pinNumber, msShineDuration) {
        var ledGpioPin = configureGpioPin(pinNumber);

        if (ledGpioPin) {
            setInterval(function () {
                switchGpioPin(ledGpioPin);
```

```
            }, msShineDuration);
        }
    }

    blinkLED(gpioPinNumber, msShineDuration);
})();
```