Quantum Browser

iOS Quick Start Guide

December 2020

Warranty

The information contained in this document is subject to change without notice. Infinite Peripherals makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties or merchantability and fitness for a particular purpose. Infinite Peripherals shall not be liable for errors contained herein, nor for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Table of Contents

Introduction

This guide describes how to get started using Quantum Browser so your website can interact with Infinite Peripheral's hardware.

Pre-Requisites

- iPod, iPhone or iPad running iOS 10.x or newer
- Q Browser app installed from the iOS App Store

Integration Steps

In order to use Quantum Broswer with your website you will need to do 3 things:

1. Add the JavaScript library
2. Add JavaScript code that utilizes functions provided by the library
3. Link your website to the iOS app

This quick start guide will go over these 3 steps, in order, and give you some sample code for some of the more common use cases of Quantum Browser. These include barcode scanning, magstripe reading and using device settings.

Adding QBrowser.js

In order to use Quantum Browser with your website you will need to include QBrowser.js. This file can be found included with our HTML demo here: link. Simply add the JavaScript file to your website like in the image below and you will have access to all the functions it provides.

```
<head>
    <title>Q Browser Sample</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script type="text/javascript" src="QBrowser.js"></script>
    <script type="text/javascript" src="demo.js"></script>
    <link href="demo.css" rel="stylesheet" type="text/css">
</head>
```

Figure 1: Including QBrowser.js

About the Demo

If you want to see QBrowser.js in action before making changes to your own code, feel free to check out our HTML demo linked above. The code provided in the demo is the same code used for the default page in the iOS app and should give you a good idea of how to use QBrowser.js and its functions.

Barcode Scanning

One of the most common ways to use our devices is to scan barcodes. With only a few simple steps, you can set up your website to be able to handle scanned barcode data.

Setting Up Your Barcode Function

In order for you to receive scanned barcode data, you must have a function properly setup to receive this data. In app settings, the “Barcode Function” field is where you specify the name of the receiving function. In the image below, you can see the default name for this function is “BarcodeData”.

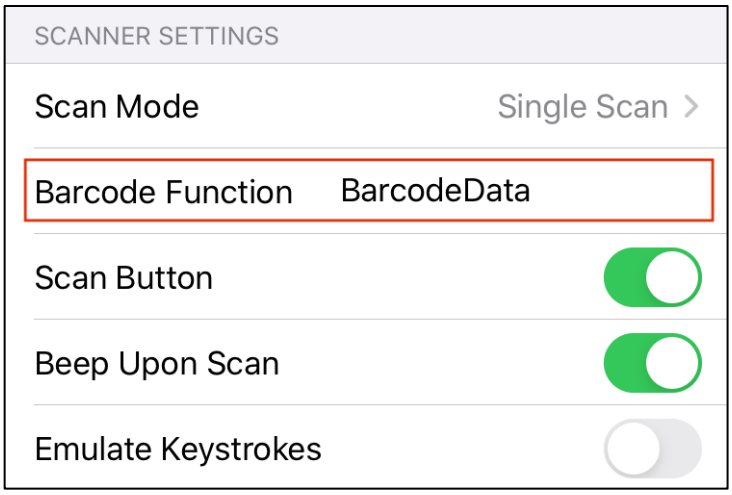

Figure 2: Barcode Function

Whenever a barcode is scanned, Quantum Browser will look for a JavaScript function matching the name in app settings and send it the scanned barcode data. The image below shows a sample function using the default name “BarcodeData”. Since the name matches what is in app settings, the function will receive the barcode, as well as its type. (i.e. Code 128, QR, etc.)

```
// Default function that receives barcodes. You can change it via App Settings.
function BarcodeData(barcode, type, typeText) {
    PrintOutput("Type(" + type + "): " + typeText + " Barcode: " + barcode + "<br/>");
}
```

Figure 3: Sample BarcodeData function

From here, you can handle the barcode data however you want. In the case of this code snippet pulled from our demo, the barcode and its type are printed to the output box on the bottom of the page.

MagStripe Reading

Another common way to use our devices is to read the magstripe data from cards. This differs slightly from barcode reading, since card data can be encrypted.

Setting Up Your MS Function

Similar to the Barcode Function that receives scanned barcode data, the MagStripe class has two functions it uses to return magstripe data. In app settings they are “MS Function” for unencrypted data and “MS Encrypted Function” for encrypted data. In the image below, you can see the fields and their default names in app settings.

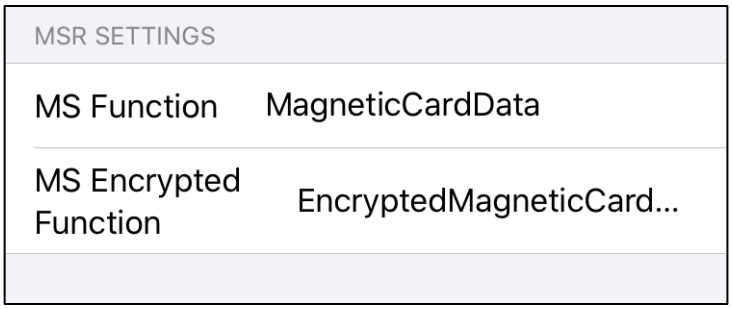

Figure 4: MSR functions

Just like the Barcode Function, swiped card data is sent to either the MS Function or the MS Encrypted Function. The image below shows a sample function using the default name “MagneticCardData”. This function receives the read data in the “card” data object then prints out the information to the output box.

```
function MagneticCardData(card) {
    for (var i = 0; i < 3; i++) {
        if (card.tracks[i]) {
            PrintOutput("Track" + (i + 1) + ": " + card.tracks[i] + "<br/>");
        }
    }

    if (card.cardholderName) {
        PrintOutput("Cardholder Name: " + card.cardholderName + "<br/>");
    }

    if (card.accountNumber) {
        PrintOutput("Account Number: " + card.accountNumber + "<br/>");
    }

    if (card.expirationMonth && card.expirationYear) {
        PrintOutput("Expires: " + card.expirationMonth + "/" + card.expirationYear + "<br/>");
    }
}
```

Figure 5: Sample MagneticCardData function

The data object sent to the MS Function can have these properties:

- card.tracks(array) - An array with card track data. Every track is returned as a string or null if the track was not read.
- card.cardholderName(string) - In the case of a valid financial card and correctly parsed track 1, cardholder name is stored here.
- card.accountNumber(string) - In the case of a valid financial card and correctly parsed track 1 or 2, account number is stored here.
- card.expirationMonth(number) - In the case of a valid financial card and correctly parsed track 1 or 2, expiration month is stored here.
- card.expirationYear(number) - In the case of a valid financial card and correctly parsed track 1 or 2, expiration year is stored here.

Device Status & Settings

Now that you know how to receive barcode and magstripe data, lets learn how to utilize the app's settings and status callbacks to help Quantum Browser interact with your website more effectively.

Status Monitoring

In QBrowser.js there are multiple classes that utilize callback functions allowing you to monitor your device's status. Using these functions, you can handle any changes to your devices in real time and make changes to your website accordingly. Check out the image below for an example of this.

```
357 window.onload = (e => {
358     // Assign callback functions to handle device status changes
359     QBrowser.Barcode.monitorStatus(BarcodeStatus);
360     QBrowser.Printer.monitorStatus(PrinterStatus);
361
362     // Set QBrowser settings
363     QBrowser.Settings.set({emulateKeystrokes:true, submitForm:true});
364 });
365
366 function PrinterStatus(info) {
367     if (info.outOfPaper == true)
368         Alert("Alert!", "Printer status: Out of paper");
369     if (info.lowBattery == true)
370         Alert("Alert!", "Printer status: Low battery");
371 }
372
373 function BarcodeStatus(info) {
374     if (info.disconnected) {
375         Alert("Scanner Disconnected", "Press scan button to wake up
              device or manually type barcode value with keyboard.");
376         UpdateUI();
377     }
378 }
```

Figure 6: Sample status monitoring functions

In this code snippet we have 2 functions being used to monitor the status of the connected scanner and printer. We do this by passing in our callback functions as the argument to each respective “monitorStatus” function. Now, if there is ever a change in our scanner or printer’s status we can act accordingly.

Using App Settings

You’ll notice on line 363 there is also a line of code that is changing Quantum Browser’s settings. These settings can also be manually changed in the iOS app settings menu, but sometimes it makes more sense to change these programmatically.

Imagine a situation where a certain page of your webapp has an input field that you want the barcode entered into. When arriving at that page you can tell Quantum Browser you want the scanned barcode to be “typed” by emulating keystrokes instead of being sent to your Barcode Function. By changing the settings programmatically, you can leverage Quantum Browser to better fit your workflow.

Connecting Your Website

When you are ready to connect your website to Quantum Browser, open the app on your iOS device and follow the steps below.

1. Click the button “Connect your Application”
2. At the next page, type in your website’s URL and click “Connect”

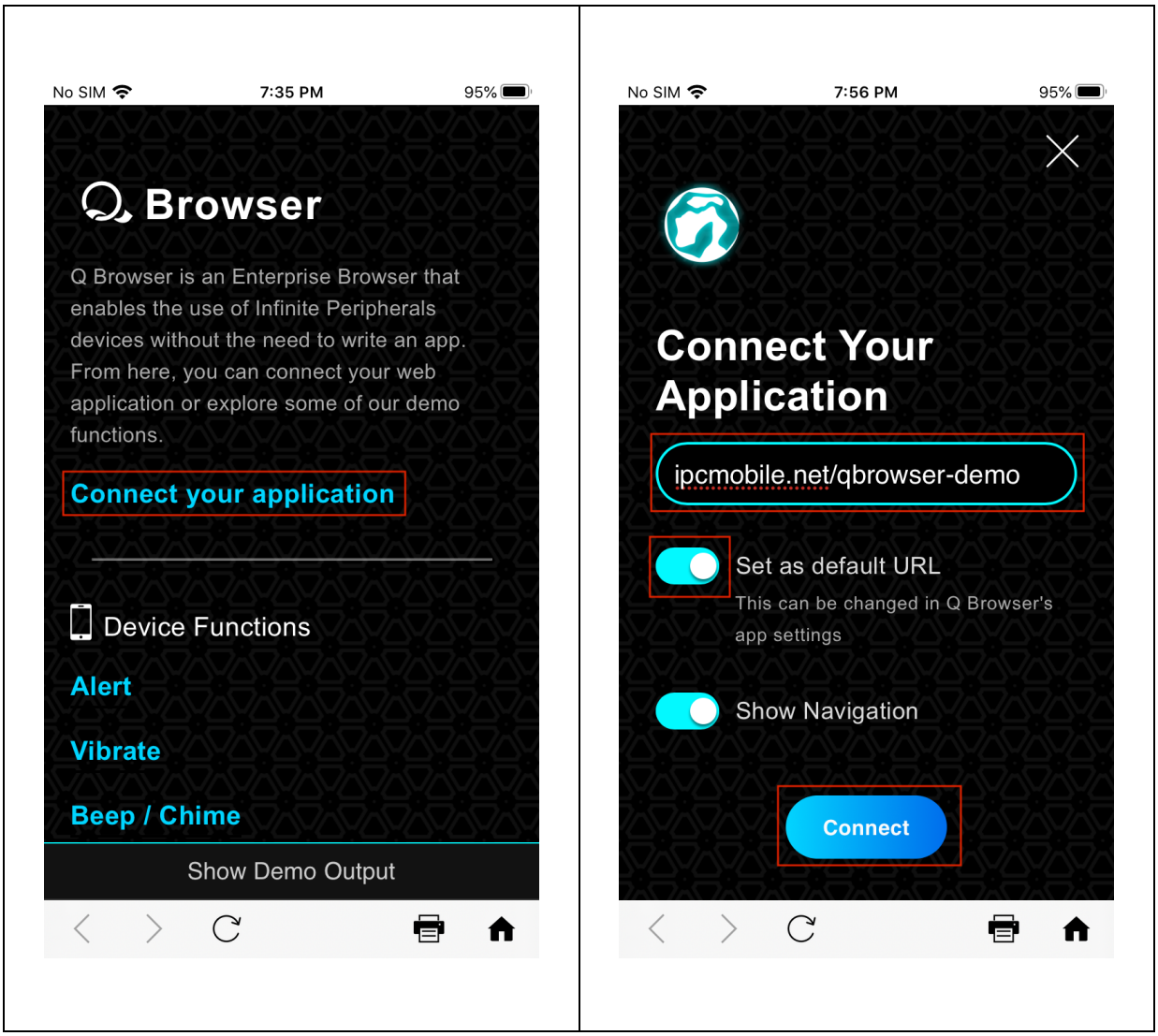

Figure 7: Connecting Your Website

3. Note: On this page you will see a switch to set the URL as default. When switched on, the app will automatically load your URL when launched instead of the default HTML page. This can also be changed in the app's settings as shown in the figure below.

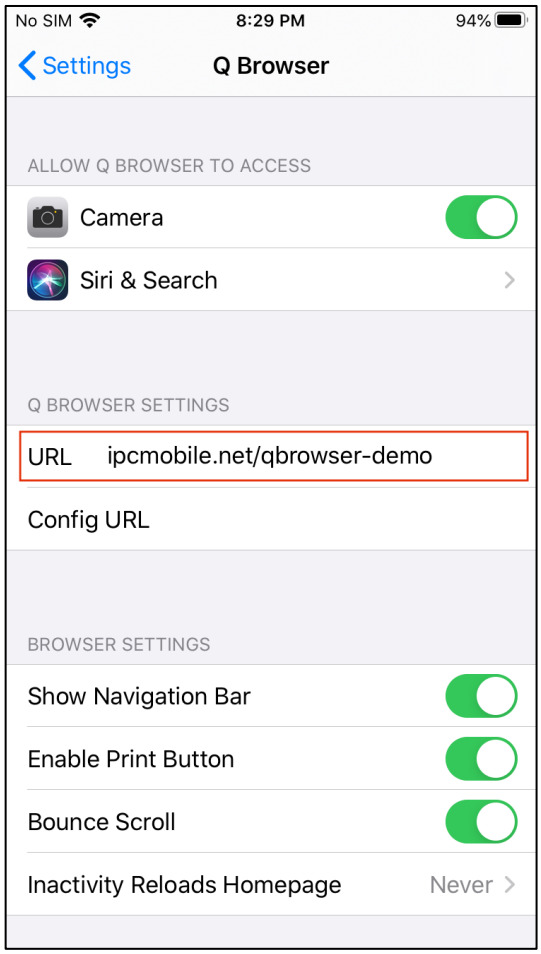

Figure 8: Default URL App Setting

4. If you ever want the app to open to the default HTML page again, just leave the URL field in app settings blank.

Additional Resources

Please refer to the following resources for application program interface (API) documentation and a sample project:

- API Documentation: http://www.ipcmobile.net/qbrowser/docs
- Sample Project: https://github.com/InfinitePeripherals/qbrowser-demo

Conclusion

Thank you for reading the iOS Quick Start Guide for Quantum Browser. If you have any questions, please visit our website at https://ipcmobile.com/products/software or contact Infinite Peripherals at 001.949.222.0300.