General Overview

OptOutCode simply requires a device owner to modify the name of a device by adding the standardized prefix “o$S”. For example, the owner of a smartphone would opt-out of the sale or sharing of their personal data collected by their phone, by the apps running on their phone, and by the IoTs connected to their phone by simply turning OptOutCode on their phone, e.g., by changing the name of their phone from "My Phone" to "o$S My Phone." In order for it to work, OptOutCode must be turned on in at least one of two devices that are paired wirelessly, or on the device that runs apps or other software. For all purposes, Consumers should be able to opt out from most if not all “Targeted Advertising or the Sale of Personal Data” by renaming three devices: their smartphone, their personal computer, and their home router.

We settled on the prefix "o$S" for a number of reasons:

A. **It’s memorable:** “o$S” is short for "do not" (zero) "sell" (dollar) "or share" (capital letter S). It’s also a tongue-in-cheek commentary on how much companies should be able to financially benefit from selling and sharing the data of devices when OptOutCode is on (zero-dollar-S).
B. **It's short:** it requires only three (3) extra characters, or four (4) extra characters including a space or underscore to separate the OptOutCode from the original device name.
C. **It’s uncommon:** searching online for "o$S" yields no results and the acronym is also not easily confused with initials or other meanings.
D. **It’s easy to detect:** adding a prefix is preferable to appending a string after the device name because it grants standard positioning (first three characters of name) hence is easier to parse with code even with devices with low computation power (e.g. some IoTs). A prefix is also more visible when a human reads a list of connected device names (for instance on the display of an IoT), making it easier to sort and/or visually confirm the request to opt-out.

We have provided several illustrative examples of how OptOutCode can work to express the Consumer’s desire to opt out in a variety of scenarios in our technical documentation (which will be uploaded to https://optoutcode.com, a micro site currently under construction), but here are a few illustrative ways in which Consumers can opt out:

A. **A Bluetooth IoT connected with a smartphone:** if a Consumer were to pair their smartphone that has OptOutCode on with a Bluetooth Classic/BLE device (e.g. a vehicle, a fitness band, a Bluetooth tracker), that second device can, during the original pairing and every time it comes back in range of the user’s smartphone, read the name of the smartphone (a standardized feature of wireless protocols), parse the first three letters, and if they are “o$S”, it can interpret it as a signal that the user wants to opt out.

B. **An app running on a smartphone:** if a Consumer downloads an app (for instance, a game or a social media app) on a smartphone that has OptOutCode on, the app can read the name of the smartphone it is installed on without requesting special permissions, parse the first three letters, and if the name of the smartphone starts with "o$S", it can interpret it as a signal that the user wants to opt out.
C. **A Wi-Fi IoT connected to a Wi-Fi router:** if a Consumer changes the name of their router to start with "o$S", hence turning OptOutCode on, and connects a Wi-Fi device (say, a home speaker or a Smart TV) to that router, the device can read the name of the Wi-Fi, parse the first three letters, a and if they are "o$S", it can interpret it as a signal that the user wants to opt out.
D. **An application/software running on a laptop/PC/terminal or other traditional computing device:** if a Consumer is running software (for instance, their email program, or a browser) on a PC that has OptOutCode on, the application can read the name of the computer it is installed on without requesting special permissions, parse the first three letters, and if the name of the computer starts with "o$S", it can interpret it as a signal that the user wants to opt out. Specifically in the case of a browser, having OptOutCode on the computer may be a signal that the browser interprets to turn on GPC, hence propagating the user's desire to opt-out from the program (the browser) to the websites it visits (via GPC).
E. **A Bluetooth or Wi-Fi "sniffer"**: Many businesses deploy hardware (e.g., in retail stores) that listen to, locate, and log the Bluetooth and Wi-Fi signals our smartphones silently broadcast at all times. This monitoring happens even without the Consumer actively pairing their device. If a Consumer turns on OptOutCode by renaming their phone to start with the letters "o$S", the beacons/sniffers can read that name, parse the first three letters, and interpret it as a signal that the user wants to opt out.

It is important to notice two things: (1) the timing of these signals matter, as it offers both Consumers a way to simply opt out and companies a way to offer an opportunity to those Consumers to opt back in, and (2) how the opt-out signal should be interpreted varies slightly depending on the situation. IoTs, specifically, have special considerations due to a combination of one or more of the following factors: limited computing power, the fact that manufacturer and Controller may not be the same entity, they can store sensitive Consumer data with little or no security, and they can exchange hands across multiple users and owners.

Technical Specification

The spec is extremely simple:

A. The Consumer renames their device by adding "o$S" as the first three letters in the device name.
B. Businesses read the name of the device using established IT protocols, determine if the name starts with "o$S" and, if so, consider it an opt-out.

Companies can easily create their own code to read the first three letters of a name of a device and trigger the opt-out request, but for illustrative purposes Privacy4Cars is including examples of code and sample technical implementations across a variety of platforms below. We have also added sample code on how to automate (for Android), or semi-automate (for iOS) the renaming of a smartphone to turn on or off OptOutCode with a simple "switch" on an app. If, as we hope,

the Colorado Department of Law decides to shortlist OptOutCode for consideration of a valid UOOM, Privacy4Cars plans to release a simple and free app for Consumers to be able to turn on and off OptOutCode (and test if OptOutCode is on or off).

Android

Android code to enable renaming a device with the o$S prefix (code also checks if "o$S" was already set as a prefix). Please note, this automation will require permissions.

```
/**
 * "Ensure that you check Android permissions before executing read/write operations with the
Bluetooth Manager."
 *  Additional checks like if name already have prefix o$S then show user information or show another
flow to user could be done
 * <!--Before Android 12 (but still needed location, even if not requested)-->
 * <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
 * <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
 * <uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"
/>
 * <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
android:maxSdkVersion="30" />
 *
 * <!--From Android 12-->
 *  <uses-permission android:name="android.permission.BLUETOOTH_SCAN"
android:usesPermissionFlags="neverForLocation" />
 * <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
 */
@SuppressLint("MissingPermission")
fun renameBluetoothDeviceName(context: AppCompatActivity) {
    val btManager = context.getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    val mBluetoothAdapter = btManager.adapter

    if(!isBtNameAlreadyHavePrivacyPrefix(context)) {
        mBluetoothAdapter.name = "o\$S " + mBluetoothAdapter.name
    }else{
        Toast.makeText(context, "Device name already have a prefix o\$S.",
Toast.LENGTH_SHORT).show()
    }
}

/**
 * This method can be used to check if device name already have a o$S prefix or not.
 */
@SuppressLint("MissingPermission")
fun isBtNameAlreadyHavePrivacyPrefix(context: AppCompatActivity): Boolean {
    val btManager = context.getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    val mBluetoothAdapter = btManager.adapter
    return mBluetoothAdapter.name?.startsWith("o\$S") == true
}
```

Android sample code an app can use to read the name of the device on which it is installed and determine if it includes the "o$S" prefix and trigger an opt-out or not.

```
/**
 * This method can be used to check if device name already have a o$S prefix or not to determine
```

```
 * if UniversalOptOut Enabled or not,This method returns boolean true/false
 * where true mean UniversalOptOut Enabled
 * and false mean UniversalOptOut Disabled
 */
@SuppressLint("MissingPermission")
fun isUniversalOptOutEnabled(context: AppCompatActivity): Boolean {
    val btManager = context.getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    val mBluetoothAdapter = btManager.adapter
    val isOptedOut = mBluetoothAdapter.name?.startsWith("o\$S") == true
    if (isOptedOut){
        Log.e("UniversalOptOutStatus","UniversalOptOut Enabled")
    }else{
        Log.e("UniversalOptOutStatus","UniversalOptOut Disabled")
    }
    return isOptedOut
}
```

Android sample code an app can use to read the name of a device the smartphone is paired with over Bluetooth (in case the “o$S” prefix is applied not to the smartphone but to a device that is set with a universal opt-out signal).

```
/**
 * Following function can be used to retrieve device name if it is already connected to a Bluetooth device.
 * If required connection update as soon as they happen i.e., Listening to connected or disconnected state
 * following broadcast receivers can be used
 * BluetoothDevice.ACTION_ACL_CONNECTED
 * BluetoothDevice.ACTION_ACL_DISCONNECT_REQUESTED
 * BluetoothDevice.ACTION_ACL_DISCONNECTED
 */
@SuppressLint("MissingPermission")
fun getConnectedDeviceName(context: AppCompatActivity): String? {
    val btManager = context.getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
    for (device in btManager.adapter.bondedDevices) {
        if (isConnected((device))) {
            return device.name
        }
    }
    return "No device connected currently."
}

private fun isConnected(device: BluetoothDevice): Boolean {
    return try {
        val m: Method = device.javaClass.getMethod("isConnected")
        m.invoke(device) as Boolean
    } catch (e: Exception) {
        throw IllegalStateException(e)
    }
}
```

Apple

iOS code to enable renaming a device with the o$S prefix. Please note, this automation will require permissions.

```
@IBAction func changeNamePressed(_ sender: Any) {
```

```
    let alert = UIAlertController(title: "Alert", message: "Go to About > Name and Type \"0$S\" in front of your current device name to enable the 0$S Global Opt-Out Mechanism", preferredStyle: .alert)

    let cancelAction = UIAlertAction(title: "Cancel", style: .destructive)

    let okAction = UIAlertAction(title: "Ok", style: .default, handler: { action in

      self.openSettings()

    })

    alert.addAction(cancelAction)

    alert.addAction(okAction)

    self.present(alert, animated: true)

  }

  func openSettings() {

    if let url = URL(string:UIApplication.openSettingsURLString) {

      if UIApplication.shared.canOpenURL(url) {

        UIApplication.shared.open(url, options: [:], completionHandler: nil)

      }

    }

  }
```

iOS sample code an app can use to read the name of the device on which it is installed and determine if it includes the "0$S" prefix and trigger an opt-out or not.

```
if UIDevice.current.name.hasPrefix("0$S") {

      print("Universal Opt Out Enabled")

    } else {

      print("Universal Opt Out Disabled")

    }
```

Windows

The Windows OS applications (e.g., Chrome browser) are built using ASP.NET framework (C# programming language). This framework provides four ways to get the name of the machine or computer:

1. string MachineName 1 = Environment.MachineName;
2. string MachineName2 = System.Net.Dns.GetHostName();

3. string MachineName3 = Request.ServerVariables[“REMOTE_HOST”].ToString(); and
4. string MachineName4 = System.Environment.GetEnvironmentVariable(“COMPUTERNAME”).

OS sample code

```
// Sample for the Environment.MachineName property

using System

class Sample

{

   public static void Main()

   {

   Console.WriteLine();

//  <-- Keep this information secure! -->

   Console.WriteLine("MachineName: {0}", Environment.MachineName);

   }

}

/*
```

Bluetooth

Bluetooth Classic mechanism to read a device name:

A. When a device (e.g., a smartphone) is in discoverable mode, the second device (e.g., the infotainment system of a vehicle) can read the first device’s name with an Extended Inquiry; or
B. When devices are paired, the commands LMP_NAME and LMP_NAME_RES are used at the beginning of the session (each device can read the other device’s name).

Bluetooth Low Energy (BLE) mechanism to read a device name:

A. When smartphone (Central) is connectable mode, BLE device (Peripheral) can read Central’s Device Name via GATT (typically not protected, pairing not required); or
B. When Central and Peripheral are paired, each device can read the other device’s name at the beginning of the session via the command GATT READ CHARACTERISTIC.

Contact us at info@optoutcode.com to learn more about supporting OptCode in your browser, app, IoT, or website.