

# **WS-Biometric Devices Walkthrough**

How to Build a WS-BD Web Camera Service

For questions or comments, contact [karenm@nist.gov](mailto:karenm@nist.gov).

# 1 Introduction

Web Services for Biometric Devices, or WS-BD, is an open source command & control protocol specifically for biometric acquisition devices. Web services use protocols that underlie the web for machine to machine communication. WS-BD allows a target biometric sensor to be exposed to and controlled by a client(s) via a web service. It replaces the need for proprietary software/hardware (e.g. drivers, firewire/USB connectors), eliminates platform restrictions, and allows wired or wireless communication. With a focus on data acquisition, this RESTFUL service architecture affords biometric sensors of any modality communication with any device that is Internet-enabled. This document is written as a “quickstart” aid for development using the WS-Biometric Devices CSDv1.0 document. The specification can be accessed at [https://www.oasis-open.org/committees/document.php?document\\_id=54815&wg\\_abbrev=biometrics](https://www.oasis-open.org/committees/document.php?document_id=54815&wg_abbrev=biometrics).

A .NET reference implementation exists to demonstrate one way to implement a WS-BD service. Libraries from the reference implementation will be used in this walkthrough to build a service. The complete .NET reference implementation can also be downloaded for free at <http://www.nist.gov/itl/iad/ig/upload/WS-BD-RefImpl-Jan2015.zip>.

## 1.1 Overview

This document provides step by step instructions and source code on how to construct a WS-BD conformant web camera service. It uses a commercial off-the-shelf (COTS) web camera as the biometric sensor. The intent is to provide a quick start to WS-BD development as well as to shorten future WS-BD development time. Portions of this document have been taken verbatim from the WS-Biometric Devices specification but may not be individually noted to keep this document simple.

The service is constructed in two stages. The first part of the walkthrough delivers a minimal WS-BD service that is active and can be accessed by a client. The second half of the instructions implement the core methods of a WS-BD service which are initialize, configure, capture and download. The walkthrough culminates in an operational service that successfully performs the core methods and live preview.

Live preview functionality is an optional feature and does not affect conformance. However, though not discussed at length, the code for it is included for demonstration purposes.

## 1.2 Audience

This document is intended for software developers with little or no experience developing a WS-BD application. Readers are assumed to have basic programming skills and a general understanding of web services. It is helpful to have some familiarity with Visual Studio, XML, and consuming REST-based web services.

## 1.3 Document Conventions

With the exception of some reference URLs, machine-readable information will typically be depicted with a mono-spaced font, such as this.

VB.NET source code is provided to facilitate constructing a new service. Bordered boxes contain source code being referenced in this document.

Text in **gray**, inside a bordered box, is not meant to be copied and is shown only as a reference point to stage the area where text should be inserted.

Text in **green** are source code comments.

Text in **black**, inside a bordered box, is the source code being discussed and is used to build the service.

Throughout this document are section titles shown in **orange text** that give the location in the WS-Biometric Devices CSDv.1.0 specification where expanded explanations can be found.

## 1.4 Requirements

- COTS web camera
- Visual Studio 2010 or later
- .Net framework 4.0 or later.
- Safari web browser
- cURL, command line HTTP client software (<http://curl.haxx.se/>)
- AForge video libraries
  - AForge.video.dll
  - AForge.video.DirectShow.dll [version 2.1.4.0 preferred]

These instructions were written using AForge.NET ([www.aforgenet.com](http://www.aforgenet.com)) an open source C# framework designed for developers and researchers in the fields of Computer Vision and Artificial Intelligence (image processing, neural networks, genetic algorithms, fuzzy logic, machine learning, robotics, etc). The DLLs mentioned above are used for processing video and images.

In the original development of the code discussed, version 2.1.4.0 of AForge was used. At the time of writing, version 2.2.5.0 is the currently available version, which upon compilation results in warnings related to deprecated source code. However, they do not cause fatal errors.

Specific hardware and software products identified in this open source project were used in order to perform technology transfer and collaboration. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products and equipment identified are necessarily the best available for the purpose.

## 1.5 Terminology

The following terms are used throughout this document and are defined by the WS-BD specification as follows:

A **client** is any software component that originates requests for biometric acquisition.

A biometric **sensor** is any component that is capable of acquiring a digital biometric sample.

The **sensor service** is the “middleware” software component that exposes a biometric sensor to a client through web services. The sensor service adapts HTTP request-response operations to biometric sensor command & control.

## 1.6 Intended Use<sup>1</sup>

A distinguishing characteristic of an implementation is the physical location of the sensor service component. WS-BD is designed to support two scenarios:

---

<sup>1</sup> This section taken verbatim from the WS-Biometric Devices CSDv1.0

1. **Physically separated.** The sensor service and biometric sensor are hosted by different physical components. A *physically separated service* is one where there is both a physical and logical separation between the biometric sensor and the service that provides access to it.
2. **Physically integrated.** The sensor service and biometric sensor are hosted within the same physical component. A *physically integrated service* is one where the biometric sensor and the service that provides access to it reside within the same physical component.

Figure 1 depicts a physically separated service. In this scenario, a biometric sensor is tethered to a personal computer, workstation, or server. The web service, hosted on the computer, listens for communication requests from clients. An example of such an implementation would be a USB fingerprint scanner attached to a personal computer. A lightweight web service, running on that computer could listen to requests from local (or remote) clients—translating WS-BD requests to and from biometric sensor commands.

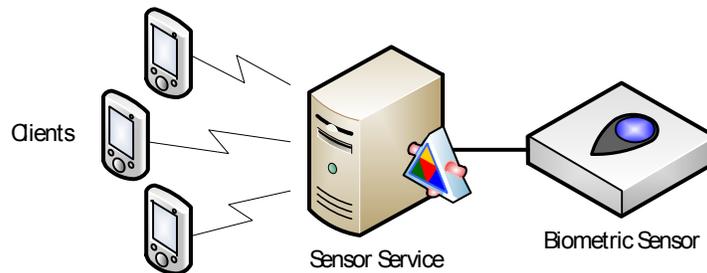


Figure 1. A physically separated WS-Biometric Devices (WS-BD) implementation.

Figure 2 depicts a physically integrated service. In this scenario, a single hardware device has an embedded biometric sensor, as well as a web service. Clients make requests directly to the integrated device; and a web service running within an embedded system translates the WS-BD requests to and from biometric sensor commands.

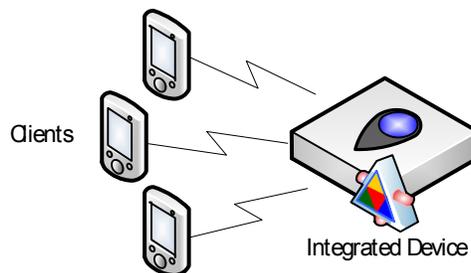


Figure 2. A physically integrated WS-Biometric Devices (WS-BD) implementation.

This walkthrough is based on the physically separated model.

## 2 Web Camera Service Solution

The Web Camera Service Solution will be comprised of five projects. The WebCameraService project is the end result of this walkthrough. The remaining four reference implementation projects/libraries are used for reference and require no additional coding. The following summarizes the projects:

**Infrastructure.** This project contains classes for threading, synchronous and asynchronous job management used in the reference implementation. The reference implementation used neither the .NET thread pool nor the Task Parallel Library—this is so that threads could be forcefully interrupted, instead of relying on cancellation tokens or other cooperative multithreading techniques. This is a valuable feature for cancelling operations that are tied to hardware; cooperative cancellation cannot always interrupt a thread blocked due to a hardware operation. This project is core to the reference implementation.

**ISensorService.** This project contains the data type and .NET interface to a WS-BD web service. This interface (and its accompanying data types) is exposed as a web service through the Windows Communications Foundation (WCF) REST bindings (specifically, the WCF *DataContractSerializer* and *WebInvoke* attributes). The classes in this project map directly to the WS-Biometric Devices specification’s Data Dictionary ([Section 3 Data Dictionary](#)).

**SensorService.** This project contains an abstract implementation that can be used to help implement a WS-Biometric Devices service. Implementations for the registration, locking, and cancellation operations are implemented here in addition to delegating other operations to use the threading classes from Infrastructure. A storage provider interface and a basic implementation to use the local file system also are in this project. This project is core to the reference implementation.

**StreamingInfrastructure.** This project provides the interfaces and classes that support the Live Preview capability. This project is optional to the reference implementation.

**WebCameraService.** This is the project that will be developed during this walkthrough. Using the .Net Reference Implementation to create a WS-BD service for a biometric sensor of any kind, all development is done in this project (the service project). At a minimum, it will contain a module that drives the service and the service class. (For this WebCameraService, it will also have a device class that contains device specific requirements.)

## 2.1 Step 1: Create the solution

### Create the solution in Administrator mode

Steps may differ slightly if Visual Studio is set with something other than Visual Basic as its default.

1. To Open Visual Studio in **Administrator mode**, click **Start**, then point to **Microsoft Visual Studio**.
2. Right click and select **Run As Administrator**.
3. Click the **File** menu, and then click **New Project**.
4. Under **Installed Templates**, expand **Visual Basic** and click **Windows**.
5. Click **Console Application**, then in the **Name** box type: **WebCameraService** and Click OK.
6. On the **File** menu, click **Save All**.
7. In the **Save Project** dialog box, ensure **“Create directory for solution”** box is checked then click **Save**.

### Select the .Net 4.0 framework

1. Right click the **WebCameraService** project and click on **Properties**.
2. Click the **Compile** tab, then click **Advanced Compile Options**.
3. In the **Advanced Compiler Settings** window, under **Target framework (all configurations)**, choose **.NET Framework 4**. (Do not choose the .Net 4.0 Client Profile.)
4. When asked if you want to change the target framework for this project, click **Yes**.

## Download the AForge DLLs

1. Download DLLs from [www.aforgenet.com/framework/downloads.htm](http://www.aforgenet.com/framework/downloads.htm).
2. Click the link “[Download Libraries only]”.
3. Unzip downloaded files (DLLs can be found in the Release folder).

## Add AForge DLLs to the WebCameraService solution

1. In Windows Explorer, create a “lib” folder inside the **WebCameraService** project folder.
2. Copy the AForge.video.dll and AForge.video.DirectShow.dll into the “lib” folder.

## Download cURL client (This is not a part of the solution. This will be used to test the solution.)

1. Download cURL from <http://curl.haxx.se/download.html>.
2. For this demonstration, place curl.exe in C:\Program Files\Curl\curl\_742\_1\ directory.
3. Add the C:\Program Files\Curl\curl\_742\_1\ path to the PATH environment variable of your machine.

cURL is not needed to build the service. It is being used (as a client to form and send HTTP requests) to test the solution. Additionally, cURL comes preinstalled on most MAC and Linux operating systems.

## 2.2 Step 2: Add Service References

The WebCameraService project must have access to certain non-default dependencies. Default dependencies are those created automatically when the solution was created. Non-default dependencies are all others. The .NET and AForge references must be added manually. WS-BD references can be added using NuGet Online Package Manager to download and reference DLLs or by manually adding WS-BD projects to the solution and referencing the libraries.

### 2.2.1 Using NuGet

The “NuGet Package Manager” is available as an extension to Visual Studio. Visual Studio 2010 does not come with the NuGet Package Manager preinstalled. It must be added via the Extension Manager. Those Visual Studio versions later than 2010 come with the extension preinstalled.

If NuGet Package Manager is not listed in your Visual Studio **Tools** menu, instructions to install the extension are listed. The Package Manager Console is a part of the NuGet Package Manager. It is a powershell tool used to manage access and installation of NuGet packages. After adding the WS-BD references, add the .NET and AForge references.

#### Install Nuget Package Manager

1. On the **Tools** menu, click **Extension Manager**.
2. Click **Online Gallery**, point to **NuGet Package Manager** then click **Download**.
3. Click **Install**, click **Close** and then click **Restart Now**.

You must restart Visual Studio in order for changes to take effect.

#### Add WS-BD references

1. Ensure the WebCameraService solution is open.
2. On the Visual Studio **Tools** menu, point to **NuGet Package Manager**.
3. Click **Package Manager Console**.
4. Run the command

```
>Install-Package WS-BiometricDevicesReferences -version 1.0.0
```

#### Add .NET references

1. On the Visual Studio Solution Explorer menu, right click WebCameraService project.
2. Click **Add Reference**, then click the **.NET** tab.
3. Select the .NET library to be referenced, Click OK.
4. Repeat steps 1-4 for each of the following .Net libraries: System.Runtime.Serialization, System.ServiceModel, System.Drawing, System.ServiceModel.Web.

#### Add AForge references

1. On the Visual Studio Solution Explorer menu, right click WebCameraService project.
2. Point to **Add Reference**, then click the **Browse** tab.
3. Double click the **lib** folder

4. Select the AForge libraries to be referenced, Click OK.
5. Repeat steps 1-4 for each of the following AForge libraries: AForge.Video.dll, AForge.Video.Directshow.dll.

### 2.2.2 Using Projects

The solution will contain the four WS-BD projects from the WS-BD Reference Implementation and the WebCameraService project. Non-default dependencies for each project need to be added.

**Download the .NET Reference Implementation from <http://bws.nist.gov>.**

#### **From the reference implementation, add the four WS-BD projects to the solution**

1. In Windows Explorer, copy the five WS-BD projects to the WebCameraService solution folder.
2. In Visual Studio Solution Explorer, click the WebCameraService solution.
3. On the **File** menu, point to **Add**, then point to **Existing Project**, then double click the project folder
4. Navigate to the project's .vbproj file, select it and click Open.
5. Repeat for each of the four projects.

#### **Add each reference from column two to the project listed in column one**

For example:

1. From Visual Studio Solution Explorer menu, right click [*project name*] (e.g., ISensorService).
2. Click **Add Reference**, then click the **Projects** tab.
3. Click the project to be referenced (e.g., StreamingInfrastructure), then click OK.

PROJECT NAME	REFERENCES
<b>SensorService</b>	Infrastructure ISensorService StreamingInfrastructure
<b>ISensorService</b>	StreamingInfrastructure
<b>Infrastructure</b>	<i>no additional dependencies</i>
<b>StreamingInfrastructure</b>	<i>no additional dependencies</i>
<b>WebCameraService</b>	ISensorService SensorService StreamingInfrastructure System.ServiceModel System.ServiceModel.Web System.Drawing System.Runtime.Serialization AForge.Video.DirectShow AForge.Video

**NOTE:**

Project references System.ServiceModel, System.ServiceModel.Web, System.Runtime.Serialization, and System.Drawing are located under the **.NET** tab. AForge DLLs can be found under the **Browse** tab.

**Set the startup project**

Ensure the WebCameraService project is set as the startup project. To verify this, in the Solution Explorer, observe the WebCameraService project name. If it is set as the startup project, the text will be **bolded**. If it is not, right click on **WebCameraService** project, and from the drop down menu click **Set as StartUp Project**.

## 2.3 Step 3: Host the Service

Web services can be hosted in different types of applications; such as IIS, Managed Windows Service or WAS. This service is self-hosted in a console application. The service class that will be created as a library and will be started and referenced by a module of the console application. The module will define the host and launch the service.

Progressively throughout the construction of this service will be an instruction to run the service. This will test compilation, provide regression testing and an opportunity for debugging before continuing. The steps to build the module are as follows:

1. In the Solution Explorer, change the name **Module1.vb** to **EntryPoint.vb**.
2. Right click the **WebCameraService** project, click **Properties** then click the **Application** tab.
3. In the **Startup object** field select **Sub Main** from the drop down menu.
4. Open the **EntryPoint.vb** class of the **WebCameraService** project.
5. Add the following imports statements and code to create and start the thread that launches the service.

```
Option Explicit On
Option Strict On

Imports System.Threading
Imports System.Threading.Tasks
Imports System.ServiceModel
Imports System.ServiceModel.Description
Imports System.ServiceModel.Web
Imports Nist.Bcl.Wsbd

Public Module EntryPoint
Public Sub Main()
    Dim ServerThread As New Threading.Thread(AddressOf Service)
    ServerThread.Start()
    Thread.Sleep(1024 * 2)
End Sub
Public Sub Service()
    Console.WriteLine("[ENTER] to terminate")
    Console.WriteLine()
    Console.ReadLine()
    Environment.Exit(0)
End Sub
End Module
```

6. Validate the console application is working. Run the application.

If successful the console will open:

```
[Enter] to terminate
```

7. Stop the service.

### 2.3.1 Endpoint

This WS-BD .NET Reference Implementation uses a Windows Communication Foundation (WCF) service model. As such, it is associated with at a minimum one unique URI address, called an **endpoint**. A WCF endpoint exists on the service side of the client/service architecture. It enables communication between a service and client. An endpoint can be declared in a configuration file or programmatically in source code, as is done in this demonstration. An endpoint consists of four properties<sup>2</sup>:

- An **address** that indicates where the endpoint can be found.
- A **binding** that specifies how a client can communicate with the endpoint.
- A **contract** that identifies the operations available.
- A set of **behaviors** that specify local implementation details of the endpoint.

An endpoint also defines the location of the service. The location is given as a URL. It may or may not specify a port number. The port number selection is directly related to the protocol. For example, the endpoint can be defined to use the HTTP protocol. If left unspecified the port number portion of the address would default to port 80 which is the default port for HTTP traffic. However, we have arbitrarily specified port 8003 to handle HTTP requests made to this service. A firewall rule must be defined to allow access to a particular endpoint. See Section 3 Errors, of this document, for additional information on defining a firewall rule for this endpoint. Additionally in this example, *localhost* is used but the host's IP address may also be used. (i.e., <http://127.0.0.1:8003/Service>)

WS-BD gives guidance regarding endpoints in relation to target sensors ([Section 2.4.4 Sensor Identity](#)). Two endpoints are defined for this service; one for streaming and one for all other operations. Each endpoint has its own binding. Having separate endpoints allows for different configurations for the streaming operation vs. other service operations. The WebHttpBinding for streaming (StreamingBinding in the code below) has its transfermode property set to *streamed*. If the WebHttpBinding transfermode is not explicitly set it uses a default value of *buffered*.

1. To the **EntryPoint.vb** module, add the line:

```
Public Const ServiceAddress As String = "http://localhost:8003/Service"  
Public Sub Main()
```

2. Inside the Service subroutine, add the code to establish the endpoints and host the service.

```
Public Sub Service()  
Try  
    ' Initialize a service host for the WebCameraService class to be created.  
    Dim Host As New WebServiceHost(GetType(WebCameraService))  
  
    ' Set WebHttpBinding values. It is important to set Transfermode to Streamed.  
    Dim StreamingBinding As New WebHttpBinding  
        With StreamingBinding  
  
            ' Set the service configured with this binding to use the  
            ' streamed mode of message transfer.  
            .TransferMode = TransferMode.Streamed  
  
        End With  
  
    End With  
Catch  
End Try  
End Sub
```

<sup>2</sup> <https://msdn.microsoft.com/en-us/library/ms733107%28v=vs.110%29.aspx>

```

        ' Set the maximum amount of memory, in bytes, to be allocated for message '
        ' buffers that receive messages from the channel.
        .MaxBufferSize = 4096

        ' Set the maximum size, in bytes, for a message that can be processed by
        the binding.
        .MaxReceivedMessageSize = 4096

        ' Set the interval of time provided for a write operation to
        ' complete before the transport raises an exception.
        .SendTimeout = TimeSpan.FromHours(24)
    End With

    ' Create the endpoint for the stream.
    Host.AddServiceEndpoint(GetType(IMixedReplaceable), StreamingBinding, ServiceAddress +
"/Stream")

    ' Create the endpoint for all other operations of service.
    Dim RestEndPoint As ServiceEndpoint = Host.AddServiceEndpoint(GetType(ISensorService),
New WebHttpBinding(), ServiceAddress)

    ' Create and set values on a behavior to enable JSON support.
    Dim Behavior As WebHttpBehavior = New WebHttpBehavior()
    With Behavior
        .DefaultOutgoingResponseFormat = WebMessageFormat.Xml
        .AutomaticFormatSelectionEnabled = True
    End With

    RestEndPoint.Behaviors.Add(Behavior)
    Host.Open()

    For Each EndPoint As ServiceEndpoint In Host.Description.Endpoints
        Console.WriteLine("{0}", EndPoint.Address.ToString())
    Next

    Console.WriteLine("[ENTER] to terminate")
    Console.WriteLine()
    Console.ReadLine()
    Environment.Exit(0)

Catch ex As Exception
    Console.WriteLine(ex.StackTrace)
    Console.WriteLine(ex.Message)
    Debugger.Break()
End Try
End Sub

```

3. From the **File** menu, click **Save All**.

## 2.4 Step 4: Create the WebCameraService Class

The WS-BD service class contains a uniform set of method stubs that support data acquisition from any sensor. Developers provide the implementation of these methods based on the sensor(s) used and guidance from the WS-BD specification. WS-BD specifies that the method response returned must be contained in the WS-BD defined “Result” type. For this exercise, before adding any code related to the actual web camera, we will insert code that simply returns “success” as the service response.

1. In the Solution Explorer, right click on the WebCameraService project.
2. Point to **Add**, then click **Class**.
3. Under **Installed Templates**, click **Common Items**.
4. In the **Name** box, type: **WebCameraService.vb**
5. Click **Add**.
6. Add the statements below to the top of the page.

```
Option Strict On

Imports Nist.Bcl.Wsbd
Imports System.ServiceModel
Imports System.Runtime.Serialization
Imports AForge.Video.DirectShow
Imports Nist.Bcl.Wsbd.Streaming
Imports System.Drawing
Imports System.IO
Imports System.Text.RegularExpressions
Imports System.Threading
Imports System.Drawing.Imaging
Imports System.ServiceModel.Web
Imports System.Net

Public Class WebCameraService

End Class
```

7. Add the following line of code, then press **Enter**.

This automatically populates the class with stubs of the core WS-BD methods.

```
Public Class WebCameraService
    Inherits SensorService

    Protected Overrides Function Initialize() As Nist.Bcl.Wsbd.Result
    End Function

    Protected Overrides Function GetConfiguration() As Nist.Bcl.Wsbd.Result
    End Function

    Protected Overrides Function SetConfiguration(ByVal configuration As
    Nist.Bcl.Wsbd.Configuration) As Nist.Bcl.Wsbd.Result
    End Function

    Public Overrides Function GetServiceInfo() As Nist.Bcl.Wsbd.Result
    End Function

    Protected Overrides Function Capture() As Nist.Bcl.Wsbd.Result
    End Function

    Public Overrides Function GetDownloadInfo(captureId As System.Guid) As Nist.Bcl.Wsbd.Result
    End Function

    Public Overrides Function Download(ByVal captureId As System.Guid) As Nist.Bcl.Wsbd.Result
    End Function

    Public Overrides Function ThriftyDownload(ByVal captureId As System.Guid, ByVal maxSize As
    Integer) As Nist.Bcl.Wsbd.Result
    End Function
```

8. Validate the web service is active and listening for client requests. Run the service.

If successful, the console output will be:

```
http://localhost:8003/Service/Stream
http://localhost:8003/Service
[ENTER] to terminate
```

While the service console is open verify the service at your endpoint is active.

Open a Command Prompt in Administrator mode

9. Click **Start**, point to **All Programs**, and then click **Accessories**.
10. Right click **Command Prompt** and select **Run As Administrator**.
11. Type: `Netsh http show servicestate`
12. Press **Enter**

If your service is active it will be listed in this output.

It can be identified by its address `HTTP://localhost:8003/Service/`.

```
URL group ID: FB0000014000004F
State: Active
Request queue name: Request queue is unnamed.
Properties:
  Max bandwidth: inherited
  Max connections: inherited
Timeouts:
  Timeout values inherited
Number of registered URLs: 1
Registered URLs:
  HTTP://+:8003/SERVICE/
```

13. Stop the service.

### 2.4.1 A Working Service Framework

Established so far is an active listening service at the URI locations listed. The solution contains references to the WS-BD .NET libraries, the code to host the service, and the shell of the new service class. At this point this solution could be used as the starting point for developing a WS-BD service for any biometric sensor, with one change. In the code of the EntryPoint.vb module, the `WebServiceHost` is declared for “WebCameraService”. However if creating a different service, replace “WebCameraService” with the name of the service being created, for example “FingerprintService” (the name of your new service class).

```
Dim Host As New WebServiceHost(GetType(FingerprintService))
```

The WS-BD method stubs would contain device specific code for a different biometric sensor (e.g., fingerprint scanner).

### 2.4.2 Service Behavior

Service behaviors have a direct relation to the performance of the service and provide various possible ways to configure key run time behaviors related to concurrency, instance context, and service throttling. Instancing refers to how objects are created and to the life time of the service object. Whenever clients make a request runtime will create service objects to provide the response. Through instancing we control how long the service instance wants to be retained. To enable a new service instance to be created for every thread request, “PerCall” is used.

When configuring a WCF service as per call, new service instances are created for every method call made via a client. In order to specify the instancing mode, provide the `InstanceContextMode` value in the `ServiceBehavior` attribute as shown below. This attribute needs to be specified on the `Service` class. Therefore the service behavior statement *must* be on the line directly above the opening class statement.

```
Imports System.ServiceModel.Web
Imports System.Net

<ServiceBehavior(InstanceContextMode:=InstanceContextMode.PerCall)>
Public Class WebCameraService
.
.
End Class
```

### 2.4.3 WS-BD Result

WS-BD requires each method stub be implemented and return an object of type `Result`. Here the `Result` type is a WS-BD defined type in the `Nist.Bcl.Wsbd` namespace. As the demonstration progresses each method will be implemented with device specific code and return a value of success. To ensure the console application runs and demonstrates which functions still need implementation we will temporarily set each function to return a value of success.

1. Inside each method stub, add the statements:

```
Protected Overrides Function Initialize() As Nist.Bcl.Wsbd.Result
Dim Result As New Result()
Result.Status = Status.Success
Return Result
End Function
```

2. Validate the service still compiles. Run the service.

If successful, console output will be:

```
http://localhost:8003/Service/Stream
http://localhost:8003/Service
[ENTER] to terminate
```

3. Stop the service.

## 2.5 Step 5: Add Device Specific Requirements

A WebCamera class will be constructed to contain device specific requirements such as device application programming interfaces (APIs) used to identify and control the biometric sensor. The WS-BD defined streaming interface (IStreamable), which provides support for Live Preview, is also implemented here so as to encapsulate all the device specific code into one class.

1. Right click on the **WebCameraService** project, point to **Add**, and then click **Class**.
2. In the **Name** box, type: **WebCamera.vb**
3. Click **Add**.
4. Add the following at the top of the WebCamera.vb class:

```
Option Strict On
Option Explicit On

Imports Nist.Bcl.Wsbd.Streaming
Imports AForge.Video.DirectShow
Imports System.Text.RegularExpressions
Imports System.Threading
Imports System.Drawing

Public Class WebCamera
```

5. Add the following variables and properties:

```
Public Class WebCamera

    Protected Property VideoSource As VideoCaptureDevice
    Protected Property IsStreaming As Boolean = False

    ' A StreamPool object handles transmitting images from a device to each client.
    Protected Property TargetPool As StreamPool

    Public Property SnapshotTimeout As Integer = 1024 * 30      'milliseconds

    Private Shared m_LatestImage As Bitmap = Nothing
    Public Property LatestImage As Bitmap
        Get
            Return m_LatestImage
        End Get
        Protected Set(value As Bitmap)
            m_LatestImage = value
        End Set
    End Property
```

5. Add the following constructor:

```
End Property
' Set the identifier of the video camera in the VideoSource object.
Public Sub New(CaptureDevice As VideoCaptureDevice)
    VideoSource = CaptureDevice

    ' Associate the new frame event with the event handler CaptureCompleteHandler.
    If VideoSource IsNot Nothing Then
        AddHandler VideoSource.NewFrame, AddressOf CaptureCompleteHandler
    End If
End Sub
```

6. Add the following subroutine:

```
End Sub
' This method is triggered by new frame event, it captures the image and continually
' supplies the image for streaming; this will happen rapidly, eg. about 30 times a
' second.
Protected Sub CaptureCompleteHandler(ByVal sender As Object, ByVal args As
AForge.Video.NewFrameEventArgs)

    Monitor.Enter(VideoSource)

    ' Save a new frame as a bitmap to return as a snapshot or for streaming or both.
    LatestImage = CType(args.Frame.Clone(), Bitmap)

    Dim rs As New System.IO.MemoryStream

    ' Save bitmap to a stream as a jpeg.
    If (LatestImage IsNot Nothing) Then
        LatestImage.Save(rs, System.Drawing.Imaging.ImageFormat.Jpeg)
    End If

    ' Continually supply the image byte array to the StreamPool - enabling live preview.
    TargetPool.UpdateLatestFrame(rs.ToArray, "image/jpeg")

    ' If not streaming stop the web camera.
    If Not IsStreaming Then
        VideoSource.SignalToStop()
    End If

    Monitor.PulseAll(VideoSource)
    Monitor.Exit(VideoSource)
End Sub
End Class
```

### 2.5.1 IStreamable

IStreamable is an interface provided in the WS-BD StreamingInfrastructure project/library that must be implemented to enable the Live Preview capability. It provides a consistent means for communication between the service application and the biometric device.

1. Add the following line to the WebCamera class, Press Enter.

```
Public Class WebCamera  
    Implements IStreamable
```

It will populate the class with four subroutine stubs needed for streaming, seen below:

```
Public Sub SignalStartStream () Implements IStreamable.SignalStartStream  
  
End Sub  
Public Sub SignalStopStream () Implements IStreamable.SignalStopStream  
  
End Sub  
Public Sub RegisterTargetPool(TargetPool As StreamPool) Implements  
IStreamable.RegisterTargetPool  
  
End Sub  
Public Sub UnregisterTargetPool() Implements IStreamable.UnregisterTargetPool  
  
End Sub
```

2. Add the following implementation details to the IStreamable method stubs:

```
Public Sub SignalStartStream () Implements IStreamable.SignalStartStream  
    IsStreaming = True  
  
    ' Start the camera.  
    VideoSource.Start()  
End Sub  
Public Sub SignalStopStream () Implements IStreamable.SignalStopStream  
    Monitor.Enter(VideoSource)  
  
    ' Stop the camera.  
    VideoSource.SignalToStop()  
    LatestImage = Nothing  
    IsStreaming = False  
    Monitor.Exit(VideoSource)  
End Sub  
Public Sub RegisterTargetPool(TargetPool As StreamPool) Implements  
IStreamable.RegisterTargetPool  
  
    ' Pair the targetPool to the device object.  
    Me.TargetPool = TargetPool  
End Sub  
Public Sub UnregisterTargetPool() Implements IStreamable.UnregisterTargetPool  
  
    ' Unpair the targetPool from the device object.  
    Me.TargetPool = Nothing  
End Sub
```

3. Add the following public method to the WebCamera class:

```
End Sub
' Called by the WebCameraService.Capture function, this function waits until at least one
' frame is returned from the CaptureCompleteHandler then saves and returns the image.
Public Function TakeSnapshot() As Bitmap
    Monitor.Enter(VideoSource)

    ' If latestImage is empty, start the camera, and wait for an image from
    ' CaptureCompleteHandler.
    If LatestImage Is Nothing Then
        VideoSource.Start()
        Monitor.Wait(VideoSource, SnapshotTimeout)
    End If

    Dim Result As Bitmap = CType(LatestImage.Clone, Bitmap)

    LatestImage = Nothing
    Monitor.Exit(VideoSource)

    Return Result
End Function
```

4. Add the following properties and shared methods:

```
' Set configurable parameters of the web camera.
Public Property FrameRate As Integer
    Get
        Return VideoSource.DesiredFrameRate
    End Get
    Set(value As Integer)
        VideoSource.DesiredFrameRate = value
    End Set
End Property

Public Property ImageWidth As Integer
    Get
        Return ImageSize.Width
    End Get
    Set(value As Integer)
        Dim ns As New Drawing.Size(value, ImageHeight)
        ImageSize = ns
    End Set
End Property

Public Property ImageHeight As Integer
    Get
        Return ImageSize.Height
    End Get
    Set(value As Integer)
        Dim ns As New Drawing.Size(ImageWidth, value)
    End Set
End Property

Public Property ImageSize As Drawing.Size
    Get
        Return VideoSource.DesiredFrameSize
```

```
End Get
Set(value As Drawing.Size)
    VideoSource.DesiredFrameSize = value
End Set
End Property
End Class
```

A list of constants to be used by the service class is stored in the module called ParameterName.vb.

5. Right click **WebCameraService** project, point to **Add**, then click **Module**.
6. Type: **ParameterName.vb**
7. Click **Add**.
8. Add the following code to the ParameterName.vb module.
9. Click **Save All**.

```
Option Strict On

Public Module ParameterName
    Public Const Width As String = "width"
    Public Const Height As String = "height"
    Public Const FrameRate As String = "frameRate"

    Public Const CaptureTime As String = "captureTime"
    Public Const ContentType As String = "contentType"
    Public Const Modality As String = "modality"
    Public Const Submodality As String = "submodality"
    Public Const Stream As String = "streamUrl"
End Module
```

10. Validate the service still compiles. Run the service.

If successful, console output will be:

```
http://localhost:8003/Service/Stream
http://localhost:8003/Service
[ENTER] to terminate
```

11. Stop the service.

## 2.6 Step 6: Implement WS-BD Methods

The WS-BD methods interact with the target sensor and determine client responses. To verify the code implemented, client requests will be made to the service. The requests can be made by any HTTP client. Curl, a command line client, will be used for this demonstration.

In the WS-BD model, a client identifies itself to a service via the use of a session—a collection of operations that originate from the same logical endpoint. To initiate a session, a client performs a *registration* operation and obtains a *session identifier* (or “session id”). During subsequent operations, a client uses this identifier as a parameter to uniquely identify itself to a server (Section 2.4.3 Client Identity). A WS-BD service must have exclusive, sovereign control over sensor hardware to perform a sensor operation such as initialization, configuration or capture. After a registration is performed a *lock* operation must be performed (Section 2.4.5 Locking). For every session performed registration and lock must be performed to initiate the session.

In the WebCameraService class we temporarily filled in the WS-BD methods with code that replies *success* to client requests. In the following steps we will replace the simple code with the actual code that enables the functionality appropriate for each method.

1. Open the WebCameraService class.
2. Add the following variables:

```
<ServiceBehavior(InstanceContextMode:=InstanceContextMode.PerCall)>
Public Class WebCameraService
    Inherits SensorService

    Private Const DEBUG As Boolean = True
    Protected Shared Property VideoSource As VideoCaptureDevice
    Protected Shared Property WebCamera As WebCamera
    Protected Shared Property ServiceInformation As New Dictionary()
    Protected Shared Property DataStore As New SortedDictionary(Of Guid, Bitmap)()
    Protected Shared Property MetadataStore As New SortedDictionary(Of Guid, Dictionary)
    Protected Shared WebCameraPool As StreamPool

    Protected Shared CaptureGuids As New GuidArray()
```

### 2.6.1 Service Information

Service information (Section 4.1 Service Information) is a list of characteristics of the service as a whole. In response to the GetServiceInfo HTTP request, the service responds with these characteristics and their possible values, to inform clients of the service's capabilities. The client uses this information to form future HTTP service requests. The parameters can be read-only and service related, such as *lastupdated* and *modality* or they can be configurable and sensor related, such as *height* and *width*. Sensor parameters are usually those that affect a capture and will differ with different target sensors. Those parameters that *must* be included in any WS-BD service are listed in Appendix A of the WS-BD specification.

The information can be supplied programmatically. However, in this implementation it is stored in an XML file and referenced in the LoadServiceInformation method. LoadServiceInformation converts the ServiceInformation.xml file into a WS-BD defined dictionary structure that is accessed by the GetServiceInfo method. Add the XML file to the solution.

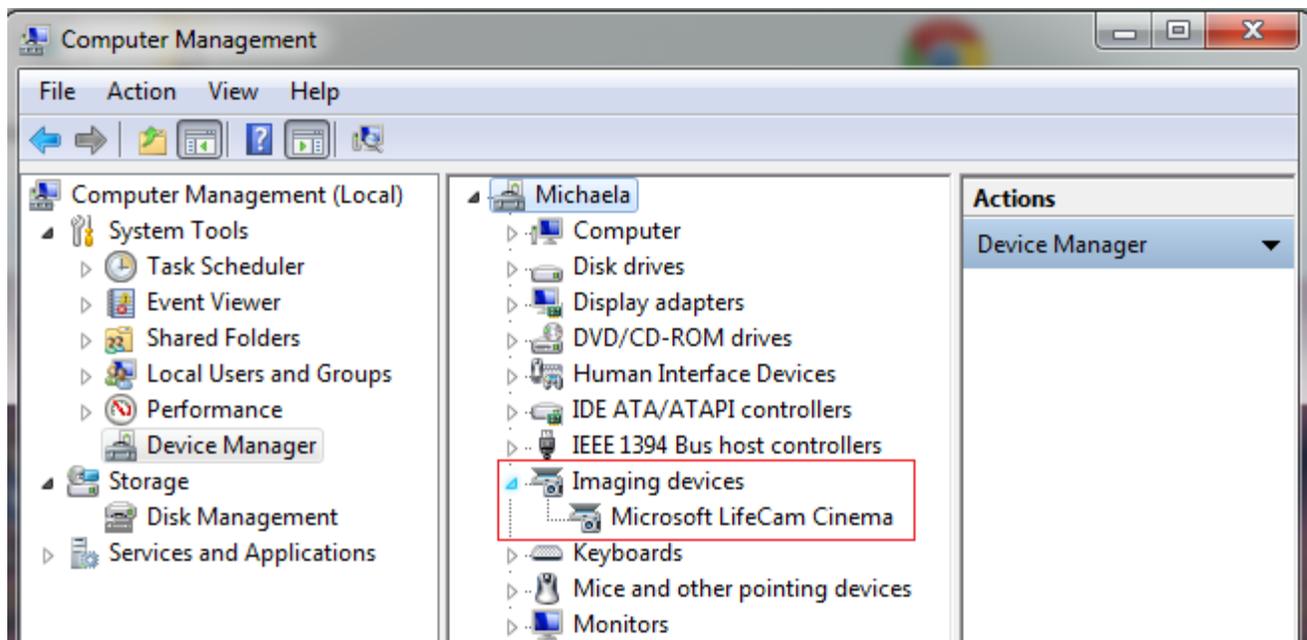
1. In Windows Explorer, copy the ServiceInformation.xml file to the WebCameraService project folder.
2. Right click **WebCameraService** Project, click **Properties**, then click the **Resources Tab**.
3. Make the **Access Modifier: Public**
4. Expand the **Add Resource** menu, then click **Add Existing File**.
5. Navigate to the project folder and select the **ServiceInformation.xml** file.
6. Click **Open**.

### 2.6.2 A Different Web Camera

If you have a different web camera than the one used in this demonstration, you will need to retrieve the device name value of the connected web camera to use in the Initialize method in the next section. The name of your camera can be found in the *Device Manager* of your Windows machine.

#### Find the device name of your camera

1. Ensure your camera is installed and connected.
2. Click **Start**, right click **Computer**, and then point to **Manage**.
3. Click **Device Manager**, Expand **Imaging devices**.
4. Copy the name of the connected web camera.



### 2.6.3 Implement the Initialize() Method

The job of the Initialize method ([Section 6.9 Initialize](#)) is to put the sensor into the ready state in preparation for capture. If the sensor has no explicit initialization API or task, the service should return a *success* result.

The TargetDeviceName variable of the Initialize method should contain the name of the web camera sensor used. Replace the name value used for TargetDeviceName variable, with the name of the connected camera.

1. In the WebCameraService.vb class, add the method LoadServiceInformation.

```
' This method is called by the Initialize method. It converts the ServiceInformation.xml
' data into a WS-BD defined dictionary structure.
Public Sub LoadServiceInformation()
    Dim stream As New MemoryStream()
    Dim data() As Byte = System.Text.Encoding.UTF8.GetBytes(My.Resources.ServiceInformation)
    stream.Write(data, 0, data.Length)
    stream.Position = 0
    Dim dcs As New System.Runtime.Serialization.DataContractSerializer(GetType(Dictionary))
    ServiceInformation = CType(dcs.ReadObject(stream), Dictionary)
End Sub
```

2. Implement the Initialize method by replacing the simple code with the following:

```
' This method sets the specified camera device in the VideoSource object which will start and
' stop the camera. It creates the necessary objects needed for streaming and starts the
' streaming. It also makes the service information available to a client via the
' LoadServiceInformation method.
Protected Overrides Function Initialize() As Nist.Bcl.Wsbd.Result
    If DEBUG Then System.Console.WriteLine("Service.Initialize()")
    Dim Result As New Result()

    ' Create a list of all connected web cameras.
    Dim VideoDevices As New FilterInfoCollection(FilterCategory.VideoInputDevice)

    Dim TargetDeviceName As String = Nothing

    ' Store the name of the web camera being used.
    TargetDeviceName = "^Microsoft LifeCam Cinema$"

    For Each Device As FilterInfo In VideoDevices
        System.Console.WriteLine("Connected Camera: {0}", Device.Name)

        ' Set the VideoSource object to the actual web camera.
        If Regex.IsMatch(Device.Name, TargetDeviceName) Then
            VideoSource = New VideoCaptureDevice(Device.MonikerString)
            Exit For
        End If
    Next Device

    ' Create a device object to enable streaming.
    WebCamera = New WebCamera(VideoSource)

    ' Create a streamPool object to enable streaming.
    WebCameraPool = New StreamPool(WebCamera)

    ' Pair the streamPool with the streaming device.
    WebCamera.RegisterTargetPool(WebCameraPool)

    LoadServiceInformation()

    If VideoSource Is Nothing Then
        Result.Status = Status.Failure
        Result.Message = "Target device not found"
    Else
        WebCamera.SignalStartStream()' Start the web camera.
        Thread.Sleep(1024 * 5)
        WebCamera.SignalStopStream()' Stop the web camera.
    End If
End Function
```

```
Result.Status = Status.Success
End If
Return Result
```

```
End Function
```

3. Replace “^Microsoft LifeCam Cinema\$” with “^[YourWEBCAMName]\$” in the Initialize() code.
4. To validate that Initialize works, run the service and then the client requests as shown below: Register, Lock and Initialize.

A session must begin with Register to get a session id (ssid), followed by lock to obtain sovereign control over a sensor, followed by Initialize to prepare the sensor for sensor operations and then any other operations. Client requests that are part of the same session use the same ssid. Begin the session by executing the Register service request.

5. At a command prompt, execute the command:

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

Successful Register command output is shown below.

The ssid returned identifies operations connected to this client and this session.

```
HTTP/1.1 200 OK
Content-Length: 191
Content-Type: application/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 28 May 2015 17:36:30 GMT

<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"><status>success</status><sessionId
>1169ef2f-b942-4f82-9b2c-b94a7f233531</sessionId></result>
```

6. Replace {ssid} in the lock command, with the ssid for this session.

```
> curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
```

Successful Lock command output is shown below.

```
HTTP/1.1 200 OK
Content-Length: 132
Content-Type: application/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 28 May 2015 17:40:51 GMT

<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"><status>success</status></result>
```

7. Replace {ssid} in the Initialize command, with the ssid for this session.

```
> curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
```

Successful Initialize command output is shown below.

```
HTTP/1.1 200 OK
Content-Length: 132
Content-Type: application/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 28 May 2015 17:42:36 GMT

<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><status>success</status></result>
```

The console window will show:

```
http://localhost:8003/Service/Stream
http://localhost:8003/Service
[Enter] to terminate

Service.Initialize()
Connected Camera: Microsoft LifeCam Cinema
```

8. Stop the service.

#### 2.6.4 Implement the GetServiceInfo() Method

The GetServiceInfo method (Section 6.8 GetServiceInfo) retrieves metadata about the service that does not depend on session-specific information, or control of the target biometric sensor. A client uses this method to make inquiry to the service as to what kind of service it is and what capabilities are available? The service responds by returning the service information about itself. For example, in this demonstration service information includes parameters such as width, height, framerate, modality, lastupdated as well as the rest of the mandatory parameters. For any configurable parameters listed, their possible values will also be returned. The width parameter is returned along with all its possible values (eg. 1280, 960, 800, 600 etc).

1. Implement the GetServiceInfo method by replacing the simple code with the following:

```
' Retrieve the service information.
Public Overrides Function GetServiceInfo() As Nist.Bcl.Wsbd.Result
    Dim Result As New Result(Status.Success)
    Result.Metadata = ServiceInformation
    Return Result
End Function
```

2. To validate, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the Lock and Initialize commands, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}

>curl http://localhost:8003/Service/info
```

If GetServiceInfo is successful, the command prompt window will show:

```
<result xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oid:2.16.840.1.101.3.9.3.1"><status>success</status><metadata>
<item><key>lastUpdated</key>-<value i:type="Parameter"><name>lastUpdated</name>
<q:type xmlns:a="http://www.w3.org/2001/XMLSchema"
xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1">a:dateTime</q:type><readOnly>true</readOnly>
<defaultValue i:type="a:dateTime" xmlns:a="http://www.w3.org/2001/XMLSchema">2013-04-01T12:24:00-
04:00</defaultValue></value></item><item>
<key>width</key>-<value i:type="Parameter"><name>width</name>
<q:type xmlns:a="http://www.w3.org/2001/XMLSchema"
xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1">a:int</q:type>
<defaultValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">800</defaultValue>
<allowedValues>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">1280</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">960</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">800</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">640</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">424</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">416</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">352</allowedValue>
<allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">320</allowedValue>
</allowedValues>
</value>
</item>
.
.
.
</result>
```

4. Stop the service.

## GetServiceInfo Sample output

```
<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <status>success</status>
  <metadata>
    <item>
      <key>lastUpdated</key>
      <value i:type="Parameter">
        <name>lastUpdated</name>
        <q:type xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1"
          xmlns:a="http://www.w3.org/2001/XMLSchema">a:dateTime</q:type>
        <readOnly>true</readOnly>
        <defaultValue i:type="a:dateTime" xmlns:a="http://www.w3.org/2001/XMLSchema">2013-04-01T12:24:00-04:00</defaultValue>
      </value>
    </item>
    <item>
      <key>width</key>
      <value i:type="Parameter">
        <name>width</name>
        <q:type xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1"
          xmlns:a="http://www.w3.org/2001/XMLSchema">a:int</q:type>
        <defaultValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">800</defaultValue>
        <allowedValues>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">1280</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">960</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">800</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">640</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">424</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">416</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">352</allowedValue>
          <allowedValue i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">320</allowedValue>
        </allowedValues>
      </value>
    </item>
    <item>
      <key>modality</key>
      <value i:type="Parameter">
        <name>modality</name>
        <q:type xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1"
          xmlns:a="http://www.w3.org/2001/XMLSchema">a:string</q:type>
        <readOnly>true</readOnly>
        <defaultValue i:type="a:string" xmlns:a="http://www.w3.org/2001/XMLSchema">Face</defaultValue>
      </value>
    </item>
    <item>
      <key>submodality</key>
      <value i:type="Parameter">
        <name>submodality</name>
        <q:type xmlns:q="urn:oid:2.16.840.1.101.3.9.3.1"
          xmlns:a="http://www.w3.org/2001/XMLSchema">a:string</q:type>
        <readOnly>true</readOnly>
        <defaultValue i:type="a:string" xmlns:a="http://www.w3.org/2001/XMLSchema">Face2d</defaultValue>
      </value>
      .
      .
      .
    </item>
  </metadata>
</result>
```

## 2.6.5 Implement the SetConfiguration() Method

The SetConfiguration method (Section 6.11 SetConfiguration) sets the configurable sensor values that prepare for and affect a capture. It is worth noting that *configuration* is a specific WS-BD defined dictionary structure. In this demonstration, for a web camera capture, integer values are set for the height, width and framerate of the web camera. For a fingerprint capture one might set the submodality to LeftIndexFlat.

1. Implement the SetConfiguration method by replacing the simple code with the following:

```
' Set configurable parameters on the sensor (eg., modality-fingerprint, submodality-
' leftIndexFlat)
Protected Overrides Function SetConfiguration(ByVal configuration As
Nist.Bcl.Wsbd.Configuration) As Nist.Bcl.Wsbd.Result
If DEBUG Then System.Console.WriteLine("Service.SetConfiguration()")
    Dim Result As New Result(Status.Success)

    ' Get parameter values from the device.
    Dim Width As Integer = WebCamera.ImageWidth
    Dim Height As Integer = WebCamera.ImageHeight
    Dim FrameRate As Integer = WebCamera.FrameRate

    Dim Valid As Boolean = True
    Dim BadFields As New StringArray()

    ' Set configurable camera parameters to client requested configuration values.
    For Each Key As String In configuration.Keys
        Select Case Key
            Case ParameterName.Width
                Width = Integer.Parse(configuration(Key).ToString())
            Case ParameterName.Height
                Height = Integer.Parse(configuration(Key).ToString())
            Case ParameterName.FrameRate
                FrameRate = Integer.Parse(configuration(Key).ToString())
            Case ParameterName.CaptureTime
                'ReadOnly
            Case ParameterName.ContentType
                'ReadOnly
            Case ParameterName.Modality
                'ReadOnly
            Case ParameterName.Submodality
                'ReadOnly
            Case Else
                Valid = False
                BadFields.Add(Key)
        End Select
    Next Key

    If Valid Then
        Try
            ' Set a new image size on the device.
            WebCamera.ImageSize = New Size(Width, Height)
        Catch
            Result.Status = Status.BadValue
            BadFields.Add(ParameterName.Width)
            BadFields.Add(ParameterName.Height)
        End Try
    End Try

    Try
```

```

        ' Set a new framerate on device.
        WebCamera.FrameRate = FrameRate
    Catch
        Result.Status = Status.BadValue
        BadFields.Add(ParameterName.FrameRate)
    End Try
Else
    Result.Status = Status.NoSuchParameter
End If

If BadFields.Count > 0 Then
    Result.BadFields = BadFields
End If
Return Result
End Function

```

To test this method, a *configuration* (§4.2 Configuration) must be sent within the body of the SetConfiguration client request to set the desired values for width, height and framerate of the web camera in preparation for a capture. WS-BD defines a *configuration* as a dictionary structure containing parameters and their values. In a typical service a client would construct this dictionary sending the desired parameters and values to the service.

The XML form of this *configuration* is contained in the conf.xml file listed below. This file can be created in different ways. It can be copied and pasted into a notepad file. Alternatively, with minor adjustments to the opening and closing tags, the output returned by the GetConfiguration method provides the basic XML for creating this file.

(The conf.xml file should be in the directory from which the curl commands are launched.)

conf.xml

```

<configuration xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:oid:2.16.840.1.101.3.9.3.1">
  <item>
    <key>width</key>
    <value xmlns:d3p1="http://www.w3.org/2001/XMLSchema" i:type="d3p1:int">800</value>
  </item>
  <item>
    <key>height</key>
    <value xmlns:d3p1="http://www.w3.org/2001/XMLSchema" i:type="d3p1:int">600</value>
  </item>
  <item>
    <key>frameRate</key>
    <value xmlns:d3p1="http://www.w3.org/2001/XMLSchema" i:type="d3p1:int">15</value>
  </item>
</configuration>

```

2. To validate SetConfiguration works, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the Lock, Initialize and Configure commands, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
```

```
>curl -X POST -H "Content-Type:text/xml" -d @conf.xml http://localhost:8003/Service/configure/{ssid}
```

If SetConfiguration is successful, the command prompt window will show:

```
C:\Users\demouser\Desktop>curl -X POST -H "Content-Type:text/xml" -d @conf.xml
http://localhost:8003/Service/configure/1169ef2f-b
942-4f82-9b2c-b94a7f233531
<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"><status>success</status></result>
```

4. Stop the service.

## 2.6.6 Implement the GetConfiguration() Method

The GetConfiguration method (Section 6.11 GetConfiguration) gets the configuration of the target sensor. This retrieves parameters and their corresponding values from the sensor. These values are loaded into a WS-BD defined Result structure and returned to the client. These are the same parameters and values that can be set by the SetConfiguration method. For the webcam in this demonstration, those include width, height and framerate.

1. Implement the GetConfiguration method by replacing the simple code with the following:

```
' Get the sensor configuration values and returns them to the client.
Protected Overrides Function GetConfiguration() As Nist.Bcl.Wsbd.Result
    If DEBUG Then System.Console.WriteLine("Service.GetConfiguration()")
    Dim Result As New Result(Status.Success)

    Result.Metadata = New Dictionary()
    Result.Metadata.Add(Width, WebCamera.ImageWidth)
    Result.Metadata.Add(Height, WebCamera.ImageHeight)
    Result.Metadata.Add(FrameRate, WebCamera.FrameRate)

    Return Result
End Function
```

2. To validate GetConfiguration works, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the Lock, Initialize and Configure commands, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
>curl -X POST -H "Content-Type:text/xml" -d @conf.xml http://localhost:8003/Service/configure/{ssid}

>curl http://localhost:8003/Service/configure/{ssid}
```

If GetConfiguration is successful, the command prompt window will show:

```
C:\Users\demouser\Desktop>curl http://localhost:8003/Service/configure/1169ef2f-b942-4f82-9b2c-b94a7f233531
<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><status>success</status><metadata><item><key>width</key><value i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">800</value></item><item><key>height</key><value i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">600</value></item><item><key>frameRate</key><value i:type="a:int" xmlns:a="http://www.w3.org/2001/XMLSchema">15</value></item></metadata></result>
```

4. Stop the service.

### 2.6.7 Implement the Capture() Method

The Capture function ([Section 6.12 Capture](#)) triggers the acquisition of biometric data. Download of data is not performed here. The function returns a handle that a client can use later to reference (or download) the captured data. Each element of data captured is assigned a unique capture Id.

Capture requires sovereign control (locking) of the target sensor. Only one client can control the sensor at any given time. Consequently, capture can only be done by one client at a time. Limiting this method to acquisition only allows the service to accommodate multiple clients with the least amount of delay. Data transfer which generally takes a bit more time, and does not require singular control of the sensor, is performed in the Download method.

1. Implement the Capture method by replacing the simple code with the following:

```
' Performs the capture and returns GUID identifiers for each captured object.
Protected Overrides Function Capture() As Nist.Bcl.Wsbd.Result
    If DEBUG Then System.Console.WriteLine("Service.Capture()")
    CaptureGuids.Clear()

    Dim Result As New Result(Status.Success)
    Dim CapturedImage As Bitmap = WebCamera.TakeSnapshot()

    Dim CaptureGuid As Guid = Guid.NewGuid()
    CaptureGuids.Add(CaptureGuid)

    ' Save the GUID identifier and its image to a DataStore.
    DataStore.Add(CaptureGuid, CapturedImage)

    ' Save the GUID and new dictionary to a MetadataStore.
    MetadataStore.Add(CaptureGuid, New Dictionary())

    ' Add the metadata values the MetadataStore.
    With MetadataStore(CaptureGuid)
        .Add(Width, DataStore(CaptureGuid).Width)
        .Add(Height, DataStore(CaptureGuid).Height)
        .Add(CaptureTime, DateTime.Now)
        .Add(ContentType, "image/bmp")
    End With

    ' Return the GUID identifiers (capture IDs) of captured data.
```

```
Result.CaptureIds = CaptureGuids

Return Result
End Function
```

2. To validate Capture works, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the Lock, Initialize, Configure and Capture command, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
```

```
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
```

```
>curl -X POST -H "Content-Type:text/xml" -d @conf.xml http://localhost:8003/Service/configure/{ssid}
```

```
>curl -i -X POST -d "" http://localhost:8003/Service/capture/{ssid}
```

Capture command output is shown below.

The captureId returned identifies an element of data captured by the sensor.

The captureId highlighted below will be used In the next section for the Download command.

```
HTTP/1.1 200 OK
Content-Length: 212
Content-Type: application/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Thu, 28 May 2015 17:49:16 GMT

<result xmlns="urn:oid:2.16.840.1.101.3.9.3.1" xmlns:i="http://www.w3.org/2001/XMLSchema-
instance"><status>success</status><captureIds>
<element>ab1a5b6c-c69a-41d4-bd29-4c4745b1668e</element></captureIds></result>
```

4. Stop the service.

## 2.6.8 Implement the Download() Method

The Download function (Section 6.13 Download) performs the data transfer from the service to a client. Using the captureIds assigned in the Capture method, loads the data into a WS-BD Result structure and transfers the data to a client. It does not require control of the sensor and can be performed by multiple clients simultaneously.

1. Implement the Download method by replacing the simple code with the following:

```
' Download the image and its metadata from the sensor.
Public Overrides Function Download(ByVal captureId As System.Guid) As Nist.Bcl.Wsbd.Result
    If DEBUG Then System.Console.WriteLine("Service.Download()")
    Dim Result As New Result(Status.Success)

    ' If the GUID id is in both DataStore and MetadataStore.
    If DataStore.ContainsKey(captureId) AndAlso MetadataStore.ContainsKey(captureId) Then
        Dim Memory As New System.IO.MemoryStream()

        ' Save the image to a memory stream object.
```

```

    DataStore(captureId).Save(Memory, Imaging.ImageFormat.Bmp)

    ' Return the metadata and the image data in a WS-BD Result object.
    Result.Metadata = MetadataStore(captureId)
    Result.SensorData = Memory.ToArray()

    Memory.Close()
Else
    Result.Status = Status.InvalidId
    Result.BadFields = New StringArray()
    Result.BadFields.Add("captureId")
End If
Return Result
End Function

```

2. To validate Download works, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the Lock, Initialize, Configure, and Capture commands, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
```

```
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
```

```
>curl -X POST -H "Content-Type:text/xml" -d @conf.xml http://localhost:8003/Service/configure/{ssid}
```

```
> curl -i -X POST -d "" http://localhost:8003/Service/capture/{ssid}
```

**NOTE:** The parameter for the download command is **captureId** highlighted in the previous capture output.

```
>curl http://localhost:8003/Service/download/{captureId}
```



## 2.7 Step 7: Live Preview

The ability to provide Live Preview ([Section 5 Live Preview](#)) of a session gives feedback to the client on when to signal a capture and/or what is going on during a capture. The `MultipartMixedReplace` function streams data from the service to a client. It works in conjunction with other parts of the code to stream data.

1. Add the following code:

```
' Transmits a stream to a client.
Public Overrides Function MultipartMixedReplace() As System.IO.Stream
    Dim Boundary As String = Guid.NewGuid().ToString()
    System.Console.WriteLine("Stream requested")

    ' Set the message headers of the outgoing message.
    WebOperationContext.Current.OutgoingResponse.Headers.Add("Connection", "keep-alive")
    WebOperationContext.Current.OutgoingResponse.ContentType = "multipart/x-mixed-replace;
    boundary=" & Boundary

    ' Return a stream.
    Return New WsbdStream(WebCameraPool, Boundary)
End Function
End Class
```

2. To verify the service can stream data, run the service and the client requests.

```
>curl -i -X POST -d "" http://localhost:8003/Service/register
```

3. Replace {ssid} in the `Lock`, and `Initialize` commands, with the ssid for this session.

```
>curl -i -X POST -d "" http://localhost:8003/Service/lock/{ssid}
>curl -i -X POST -d "" http://localhost:8003/Service/initialize/{ssid}
```

4. In the **Safari** browser, enter the streaming endpoint address:

<http://localhost:8003/Service/Stream>

5. Press enter.

If successful, you will observe live feed in the browser from the webcam. The Safari browser is suggested for viewing the Live Preview feed. As of this writing, it is a quick and easily accessible option to view the feed. The browser should work without requiring any additional configuration.

## 3 Errors

### 3.1 Firewall disallows the service to run

When the service is run it attempts to listen for HTTP requests on the URL designated. At a minimum, a URL reservation allows the stated user account to run a process (the service) at the location specified. In other words, to assign the service to listen at this IP address and port number (this is altering the firewall settings), a URL reservation is needed for which administrator permission is necessary.

If the firewall is not configured to allow the service to run, or you are not running Visual Studio in Administrator mode, a likely error would be:

Command prompt window:

```
HTTP could not register URL http://+:8003/Service/Stream/. Your process does not have access rights to this namespace (see http://go.microsoft.com/fwlink/?LinkId=70353 for details).
```

This means the firewall is disallowing your process to run. The account the process is running as does not have permission to run this service at this location. Two possible ways to fix the problem:

#### Open Visual Studio in Administrator mode

1. Click **Save All** to save the solution.
2. Click **Exit** to close Visual Studio.
3. Click **Start**, then point to **Microsoft Visual Studio**.
4. Right click and select **Run As Administrator**.

OR

#### Make the URL reservation

1. Click **Start**, point to **All Programs**, and then click **Accessories**.
2. Right click **Command Prompt** and select **Run As Administrator**
3. Run the command below.

```
Netsh http add urlacl url=http://+:{port number}/ user={Domain\Username}
```

e.g: Netsh http add urlacl url=http://+:8003/ user=winGroup\Jane

**NOTE:** Use the machine name in place of the domain if the machine is not part of a domain.

If the reservation was successful, output will be:

```
Administrator: Command Prompt
C:\Windows\system32>netsh http add urlacl url=http://localhost:8003/Service
user=winGroup\Jane
```

```
URL reservation successfully added
```

```
C:\Windows\system32>
```

4. Validate the error is resolved. Run the service.

### 3.2 Unable to read data from conf.xml file

The SetConfiguration method requires a configuration to be sent in the body of the HTTP request made to the service. The conf.xml file contains the configuration information. You may send the request and the response returned is the error listed below:

```
Warning: Couldn't read data from file "conf.xml", this makes an empty POST.
```

```
␣␣␣␣<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
<title>Request Error</title>
```

This is an indication that the conf.xml file is not found. For this demonstration, the conf.xml file has to be in the same directory from which cURL launches the SetConfiguration HTTP request. To fix the problem:

1. Change directory to where the conf.xml file is located.
2. Resend the cURL command that sends the SetConfiguration HTTP request.

## 4 Bibliography

Micheals, R. e. (2015, June 18). *WS-Biometric Devices v1.0 - Candidate for CSD 03.doc Details*. Retrieved from OASIS Organization for the Advancing of Open Standards for the Information Society:  
[https://www.oasis-open.org/committees/document.php?document\\_id=54815&wg\\_abbrev=biometrics](https://www.oasis-open.org/committees/document.php?document_id=54815&wg_abbrev=biometrics)