Avaya IP Office DevLink3 Tutorial, Issue 1.0

Contents

Introduction

With the release of IPOffice 10, a new interface was introduced - DevLink3. DevLink3 provides unsolicited real time events to a client application. For example, the client could register to receive events pertaining to SIP phone activity (calls made, answered, etc). The purpose of this document is to provide details on the necessary programming steps required to communicate with IPOffice using the DevLink3 protocol. The DevLink3 client code examples referred to in this document were written in C# (using Visual Studio 2015 on Windows 10). The code was tested against IP Office 10.0. Please also refer to the DevLink3 API reference document available on the DevConnect portal. The code is sample code only and it is expected that the API consumers will add error checking and exception handling as appropriate.

Establishing a connection

Communication between the IPOffice and client application using the DevLink3 protocol is over TCP or TLS. This example uses TCP.

The following code will initialize a connection to IPOffice and also start a separate receive thread to listen over any incoming traffic from IPOffice:

```
client = new TcpClient(IPOaddress.Text, ipoport);
stream = client.GetStream();
background = new Thread(Receive);
background.Start();
```

The **Receive** function will constantly listen for any traffic from IPOffice and then parse any data received.

This code is wrapped in a function which does some validation and checks if the background thread is already running:

```
private bool StartComms()
{
   bool retvalue = true;
   if(background == null)
   {
      retvalue = false;
      if (string.IsNullOrWhiteSpace(IPOaddress.Text))
      {
         this.SetText("Need IPO address \r\n");
      }
      else
      {
         this.SetText("Attempting establish connection \r\n");
         try
         {
            client = new TcpClient(IPOaddress.Text, ipoport);
            stream = client.GetStream();
            background = new Thread(Receive);
            background.Start();
            retvalue = true;
            this.SetText("Connected \r\n");
         }
         catch
         {
            this.SetText("Could not connect \r\n");
            retvalue= false;
         }
      }
   }
   return retvalue;
}
```

Before sending any data to IPOffice the **StartComms** function is called.

DevLink3 packets

Packet encoding

The DevLink3 packet is composed of a byte array of hexadecimal values. The string representation of hexadecimal octet values are encoded as follows:

```
private byte[] HexOcttoByte(string input)
{
   var length = input.Length / 2;
   var output = new byte[length];
   for (var i = 0; i < length; i++)
   {
      output[i] = Convert.ToByte(input.Substring((i * 2), 2), 16);
   }
   return output;
}
```

Packet structure

Each DevLink3 packet is made up of a Header, a unique packet **requestid** and a packet body:

```
class Packet
{
   private byte[] Pbytes, Headerbytes, RIDBodybytes;
   public string hexheader, requestid, body, pktlength;
```

The **requestid** is generated by the client application when required as follows:

```
private string RequestID()
{
   Random seed = new Random();
   int arb = seed.Next();
   requestid = arb.ToString("D8");
   if (requestid.Length > 8)
   {
      requestid = requestid.Substring(0, 8);
   }
```

```
        return requestid;
    }
```

The header (**hexheader**) and body will vary according to the type of packet being sent or received:

```
    public static class PacketTypes
    {
        public static string Test { get { return "002A0001"; } }
        public static string TestR { get { return "802A0001"; } }
        public static string Authenticate { get { return "00300001"; } }
        public static string AuthenticateR { get { return "80300001"; } }
        public static string EventRequest { get { return "00300011"; } }
        public static string EventRequestR { get { return "80300011"; } }
        public static string Event { get { return "10300011"; } }
    }

    public static class Response
    {
        public static string Pass {  get { return "00000000"; } }
        public static string Fail { get { return "80000041"; } }
        public static string Challenge { get { return "00000002"; } }
        public static string UnknownFlag { get { return "80000021"; } }
    }
```

Before sending any packet the relevant packet elements (attributes) are all copied in to a single byte array **Pbytes** as follows:

```
    public void BuildBuffer()
    {
        RIDBodybytes = HexOcttoByte(RequestID() + body);
        pktlength = (3 + (hexheader.Length / 2) + (requestid.Length / 2) + (body.Length /
2)).ToString("X4");
        Headerbytes = HexOcttoByte("49" + pktlength + hexheader);
        Pbytes = new byte[Headerbytes.Length + RIDBodybytes.Length];
        Buffer.BlockCopy(Headerbytes, 0, Pbytes, 0, Headerbytes.Length);
        Buffer.BlockCopy(RIDBodybytes, 0, Pbytes, Headerbytes.Length, RIDBodybytes.Length);
    }
```

The byte array can then be accessed as an attribute of the packet:

```
public byte[] Bytes { get { return Pbytes; } }
```

Creating a test packet to send

A test packet can be sent once to validate the connection before attempting to authenticate as follows:

```
private void TestPkt_Click(object sender, EventArgs e)
{
   if (StartComms())
   {
      Packet STest = null;
      STest = new Packet(PacketTypes.Test);
      STest.BuildBuffer();
      stream.Write(STest.Bytes, 0, STest.Bytes.Length);
      this.SetText("\r\nTestS: " + BitConverter.ToString(STest.Bytes, 0).Replace("-", string.Empty));
   }
}
```

The Packet constructor is overloaded according to the number of parameters. When invoking **new Packet(PacketTypes.Test)** the 'base' constructor is called as follows:

```
public Packet(string hexvalue)
{
   hexheader = hexvalue;
   if (hexvalue == DevLink3.PacketTypes.Test)
   {
      body = "00000000";
   }
}
```

Authenticating with IPOffice to receive events

In order to authenticate with IPOffice, in this example we will demonstrate a TCP connection using **SHA1**. Initially we create an authentication request packet as follows:

```
    private void Connect_Click(object sender, EventArgs e)
    {
      if (StartComms())
      {
        Packet STest = null;
        STest = new Packet(PacketTypes.Authenticate, UserName.Text);
        STest.BuildBuffer();
        pendingRequest = STest.requestid;
        stream.Write(STest.Bytes, 0, STest.Bytes.Length);
        this.SetText("\r\nConnectS: " + BitConverter.ToString(STest.Bytes, 0).Replace("-",
string.Empty));
      }
    }
```

As mentioned previously the **Packet** constructor is overloaded as follows:

```
    public Packet(string hexvalue, string txt) : this(hexvalue)
    {
      if (hexheader == DevLink3.PacketTypes.Authenticate)
      {
        body = "00000001" + BitConverter.ToString(Encoding.UTF8.GetBytes(txt.Trim() +
Char.MinValue)).Replace("-", string.Empty);
      }
      if (hexheader == DevLink3.PacketTypes.EventRequest)
      {
        txt = txt + Char.MinValue;
        body = (txt.Length / 2).ToString("X4") +
BitConverter.ToString(Encoding.UTF8.GetBytes(txt.Trim())).Replace("-", string.Empty);
      }
    }
```

Initially the **:this(hexvalue)** invokes the 'base' constructor which does nothing as the hexvalue (**Packet Type**) is not Test. Note as the hexvalue (**Packet Type**) supplied is **Authenticate**, the username is encoded to a UTF8 string and a null terminator is added. This is then appended to

00000001 to provide the packet body. The **BuildBuffer** function will then eventually convert each pair of string values in the string to an array of individual hexadecimal values using the **HexOcttoByte** function which will then be sent to IPOffice.

In sending the authentication request a global variable **pendingRequest** is set, so that any response from IPOffice can be correlated with the appropriate request.

The response to the initial authentication request is processed in the background thread function **Receive**. The **Receive** function will parse any incoming data to evaluate the type and content of the packet being received:

```
private string ParseHeader(byte[] Data)
{
   return (BitConverter.ToString(Data, 0).Replace("-", string.Empty).Substring(6, 8));
}

private string ParseRequestID(byte[] Data)
{
   return (BitConverter.ToString(Data, 0).Replace("-", string.Empty).Substring(14, 8));
}

private string ParseResponse(byte[] Data)
{
   return (BitConverter.ToString(Data, 0).Replace("-", string.Empty).Substring(22, 8));
}

private byte[] GetChallenge(byte[] Data)
{
   int size = Convert.ToInt16(Data[18]);
   byte [] result = new byte[size];
   Buffer.BlockCopy(Data, 19, result, 0, size);
   return result;
}
```

Once the appropriate criteria are met, the **Receive** function will automatically respond to the SHA1 handshake by generating a new packet and sending it as follows:

```
Packet RTest = null;
RTest = new Packet(PacketTypes.Authenticate, GetChallenge(bytes), Password.Text);
RTest.BuildBuffer();
```

```
            pendingRequest = RTest.requestid;
            stream.Write(RTest.Bytes, 0, RTest.Bytes.Length);
            this.SetText("\r\nChallengeS: " + BitConverter.ToString(RTest.Bytes, 0).Replace("-",
string.Empty));
```

Note the SHA1 challenge data is extracted from the response using the **GetChallenge** function and then supplied as a parameter to the Packet constructor (again the 'base' constructor does nothing when invoked):

```
    public Packet(string hexvalue, byte [] challenge, string password) : this(hexvalue)
    {
      if (hexheader == DevLink3.PacketTypes.Authenticate)
      {
          string response;
          byte[] utf8pwd = new byte[16];
          byte[] HashBytes = new byte[challenge.Length + 16];

          Buffer.BlockCopy(Encoding.UTF8.GetBytes(password.Trim()), 0, utf8pwd, 0,
(Encoding.UTF8.GetBytes(password.Trim()).Length < 17) ?
Encoding.UTF8.GetBytes(password.Trim()).Length : 16);

          Buffer.BlockCopy(challenge, 0, HashBytes, 0, challenge.Length);
          Buffer.BlockCopy(utf8pwd, 0, HashBytes, challenge.Length, 16);

          SHA1 sha = SHA1.Create();
          byte[] HashOutp = sha.ComputeHash(HashBytes);

          response = BitConverter.ToString(HashOutp).Replace("-", string.Empty); ;
          body = "00000050" + (response.Length / 2).ToString("X8") + response;
      }
    }
```

The SHA1 requires the following namespace:

```
using System.Security.Cryptography;
```

Again, the password supplied to the constructor is converted to UTF8 with a null terminator. The resulting Hash value is converted to a string and appended to 00000050 to provide a packet

body. **Buildbuffer** and **HexOcttoByte** functions will then convert the string to hexadecimal octet values before sending.

If the **Receive** function then parses a message back from IPOffice to show success, the **Receive** function will automatically register for events with IPOffice (the sample code has an initial default Event flag of –SIPTrak which means the client application will be provided with any events related to SIP phone activity for example – make call, etc):

```
this.EventFlags.Text = "-SIPTrack";
```

This value is passed as a parameter in generating the Packet to register for events as follows:

```
this.SetText("\r\n Authenticate succeeded");
Packet ETest = null;
ETest = new Packet(PacketTypes.EventRequest, EventFlags.Text.Trim());
ETest.BuildBuffer();
pendingRequest = ETest.requestid;
stream.Write(ETest.Bytes, 0, ETest.Bytes.Length);
this.SetText("\r\nEventS: " + BitConverter.ToString(ETest.Bytes, 0).Replace("-",
string.Empty));
```

The flag is passed to the Packet constructor as follows:

```
public Packet(string hexvalue, string txt) : this(hexvalue)
{
  if (hexheader == DevLink3.PacketTypes.Authenticate)
  {
    body = "00000001" + BitConverter.ToString(Encoding.UTF8.GetBytes(txt.Trim() +
Char.MinValue)).Replace("-", string.Empty);
  }
  if (hexheader == DevLink3.PacketTypes.EventRequest)
  {
    txt = txt + Char.MinValue;
    body = (txt.Length / 2).ToString("X4") +
BitConverter.ToString(Encoding.UTF8.GetBytes(txt.Trim())).Replace("-", string.Empty);
  }
}
```

The full **Receive** function caters for any incoming traffic (and prints it to the dialog window):

```
    public void Receive()
    {
      byte[] bytes = new byte[1024];
      while (true)
      {
        int bytesRead = stream.Read(bytes, 0, bytes.Length);
        this.SetText("\r\nR: " + BitConverter.ToString(bytes, 0).Replace("-", string.Empty));
        if(ParseHeader(bytes) == PacketTypes.AuthenticateR)
        {
          if ((ParseRequestID(bytes) == pendingRequest) && (bytesRead>16))
          {
            if (ParseResponse(bytes) == Response.Challenge)
            {
              Packet RTest = null;
              RTest = new Packet(PacketTypes.Authenticate, GetChallenge(bytes), Password.Text);
              RTest.BuildBuffer();
              pendingRequest = RTest.requestid;
              stream.Write(RTest.Bytes, 0, RTest.Bytes.Length);
              this.SetText("\r\nChallengeS: " + BitConverter.ToString(RTest.Bytes, 0).Replace("-",
string.Empty));
            }
            else if (ParseResponse(bytes) == Response.Fail)
              this.SetText("\r\n Authentication failed");
            else if (ParseResponse(bytes) == Response.Pass)
            {
              this.SetText("\r\n Authenticate succeeded");
              Packet ETest = null;
              ETest = new Packet(PacketTypes.EventRequest, EventFlags.Text.Trim());
              ETest.BuildBuffer();
              pendingRequest = ETest.requestid;
              stream.Write(ETest.Bytes, 0, ETest.Bytes.Length);
              this.SetText("\r\nEventS: " + BitConverter.ToString(ETest.Bytes, 0).Replace("-",
string.Empty));
            }
          }
        }
        if (ParseHeader(bytes) == PacketTypes.TestR)
          this.SetText("\r\n Test responded");
        if (ParseHeader(bytes) == PacketTypes.EventRequestR)
        {
          if (ParseResponse(bytes) == Response.UnknownFlag)
```

```
            this.SetText("\r\n Event unknown flag string");
         else if (ParseResponse(bytes) == Response.Pass)
            this.SetText("\r\n Event register success");
      }
      if (ParseHeader(bytes) == PacketTypes.Event)
         this.SetText("\r\n Event received");
   }
}
```