


# Getting RAW Soap Data from a Web Reference Client running in ASP.net


Asked 12 years ago   Active 11 months ago   Viewed 117k times


  
95

I'm trying to trouble shoot a web service client in my current project. I'm not sure of the platform of the Service Server (Most likely LAMP). I believe there is a fault on their side of the fence as i have eliminated the potential issues with my client. The client is a standard ASMX type web reference proxy auto generated from the service WSDL.

What I need to get to is the RAW SOAP Messages (Request and Responses)

What is the best way to go about this?

  
63



asp.net

soap

proxy

asmx

web-reference

edited Sep 4 '12 at 14:19



John Saunders

157k   23   222   381

asked Nov 19 '08 at 0:38



Andrew Harry

13.3k   16   64   101

## 9 Answers

| Active | Oldest | Votes |
|--------|--------|-------|
|--------|--------|-------|

  
135

I made following changes in `web.config` to get the SOAP (Request/Response) Envelope. This will output all of the raw SOAP information to the file `trace.log`.

```
<system.diagnostics>
  <trace autoflush="true"/>
  <sources>
    <source name="System.Net" maxdatasize="1024">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>
    <source name="System.Net.Sockets" maxdatasize="1024">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>
  </sources>
```







```

</sharedListeners>
<switches>
  <add name="System.Net" value="Verbose"/>
  <add name="System.Net.Sockets" value="Verbose"/>
</switches>
</system.diagnostics>

```

edited May 9 '12 at 22:31



Keltex

25.3k

11

73

108

answered Jan 9 '09 at 14:29

Nadeem Abbasi

- 2 This is not working for me in 02/2012 using VS 2010. Anyone know a more up-to-date solution? – [qxotk](#) Feb 28 '12 at 21:12
- 2 This is helpful. However, in my case the responses are Gzipped and pulling either the ASCII or the hex codes out of the combined output is a pain. Are their alternative output methods which are either binary or text only? – [user565869](#) Jan 14 '13 at 22:33
- 2 @Dedicated - you can set the path to the trace.log file by adjusting the value of the initializeData attribute in the example above. – [DougCouto](#) May 24 '13 at 15:10
- 3 It's of no use anymore, I used wireshark to see the raw SOAP data. But thanks anyway! – [Dedicated](#) May 24 '13 at 16:53
- 7 Good. But the XML for me is like : System.Net Verbose: 0 : [14088] 00000000 : 3C 3F 78 6D 6C 20 76 65-72 73 69 6F 6E 3D 22 31 : <?xml version="1 System.Net Verbose: 0 : [14088] 00000010 : 2E 30 22 20 65 6E 63 6F-64 69 6E 67 3D 22 75 74 : .0" encoding="ut ..... Any idea how to format it or have only xml data.? – [Habeeb](#) Nov 5 '15 at 12:40 ✎

35

You can implement a SoapExtension that logs the full request and response to a log file. You can then enable the SoapExtension in the web.config, which makes it easy to turn on/off for debugging purposes. Here is an example that I have found and modified for my own use, in my case the logging was done by log4net but you can replace the log methods with your own.

```

public class SoapLoggerExtension : SoapExtension
{
    private static readonly ILog log =
    LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
    private Stream oldStream;
    private Stream newStream;

    public override object GetInitializer(LogicalMethodInfo methodInfo,
    SoapExtensionAttribute attribute)
    {
        return null;
    }

    public override object GetInitializer(Type serviceType)
    {

```



```

public override void Initialize(object initializer)
{

}

public override System.IO.Stream ChainStream(System.IO.Stream stream)
{
    oldStream = stream;
    newStream = new MemoryStream();
    return newStream;
}

public override void ProcessMessage(SoapMessage message)
{
    switch (message.Stage)
    {
        case SoapMessageStage.BeforeSerialize:
            break;
        case SoapMessageStage.AfterSerialize:
            Log(message, "AfterSerialize");
            CopyStream(newStream, oldStream);
            newStream.Position = 0;
            break;
        case SoapMessageStage.BeforeDeserialize:
            CopyStream(oldStream, newStream);
            Log(message, "BeforeDeserialize");
            break;
        case SoapMessageStage.AfterDeserialize:
            break;
    }
}

public void Log(SoapMessage message, string stage)
{
    newStream.Position = 0;
    string contents = (message is SoapServerMessage) ? "SoapRequest " :
"SoapResponse ";
    contents += stage + ";";

    StreamReader reader = new StreamReader(newStream);

    contents += reader.ReadToEnd();

    newStream.Position = 0;

    log.Debug(contents);
}

void ReturnStream()

```

```

void ReceiveStream()
{
    CopyAndReverse(newStream, oldStream);
}

public void ReverseIncomingStream()
{
    ReverseStream(newStream);
}

public void ReverseOutgoingStream()
{
    ReverseStream(newStream);
}

public void ReverseStream(Stream stream)
{
    TextReader tr = new StreamReader(stream);
    string str = tr.ReadToEnd();
    char[] data = str.ToCharArray();
    Array.Reverse(data);
    string strReversed = new string(data);

    TextWriter tw = new StreamWriter(stream);
    stream.Position = 0;
    tw.Write(strReversed);
    tw.Flush();
}

void CopyAndReverse(Stream from, Stream to)
{
    TextReader tr = new StreamReader(from);
    TextWriter tw = new StreamWriter(to);

    string str = tr.ReadToEnd();
    char[] data = str.ToCharArray();
    Array.Reverse(data);
    string strReversed = new string(data);
    tw.Write(strReversed);
    tw.Flush();
}

private void CopyStream(Stream fromStream, Stream toStream)
{
    try
    {
        StreamReader sr = new StreamReader(fromStream);
        StreamWriter sw = new StreamWriter(toStream);
        sw.WriteLine(sr.ReadToEnd());
        sw.Flush();
    }
}

```

```

ex.Message);
    log.Error(message, ex);
}
}
}

[AttributeUsage(AttributeTargets.Method)]
public class SoapLoggerExtensionAttribute : SoapExtensionAttribute
{
    private int priority = 1;

    public override int Priority
    {
        get { return priority; }
        set { priority = value; }
    }

    public override System.Type ExtensionType
    {
        get { return typeof (SoapLoggerExtension); }
    }
}

```

You then add the following section to your web.config where YourNamespace and YourAssembly point to the class and assembly of your SoapExtension:

```

<webServices>
  <soapExtensionTypes>
    <add type="YourNamespace.SoopLoggerExtension, YourAssembly"
        priority="1" group="0" />
  </soapExtensionTypes>
</webServices>

```

answered Nov 19 '08 at 2:12



**John Lemp**

4,880 3 25 36

I'll give it a go. I had looked into soap extentions, but it looked to me like it was more for hosting the service. Will give you the tick if it works :) – [Andrew Harry](#) Nov 19 '08 at 22:09

Yes, this will work to log the calls made from your web application through the generated proxy. – [John Lemp](#) Nov 20 '08 at 0:29

This is the route I went with, it's working really well, and I can use xpath to mask any sensitive data before logging it. – [Dave Baghdanov](#) Jun 4 '14 at 23:40

I needed to add headers to the soap request on the client. This helped me. [rhyous.com/2015/04/29/...](http://rhyous.com/2015/04/29/) – [Rhyous](#) Apr 29 '15 at 22:23



Not sure why all the fuss with web.config or a serializer class. The below code worked for me:

28

```
XmlSerializer xmlSerializer = new XmlSerializer(myEnvelope.GetType());

using (StringWriter textWriter = new StringWriter())
{
    xmlSerializer.Serialize(textWriter, myEnvelope);
    return textWriter.ToString();
}
```

answered Jan 22 '15 at 22:37



19 Where does myEnvelope come from? – ProfK Mar 16 '15 at 6:55

6 I dont understand why this answer doesnt have more upvotes, straight and to the point! Well done! – Batista Aug 6 '15 at 12:49

1 myEnvelope would be the object that you're sending over via the web service. I couldn't quite get this to work as is described (particularly the textWriter.ToString function) but it worked well enough for my purposes in debug mode in that you can see the raw xml after you call serialize. Very much appreciate this answer, as some of the others have worked with mixed results (logging using the web.config only returned part of the xml for example, and wasn't very easy to parse out either). – user2366842 Oct 9 '15 at 15:20

@Bimmerbound: would this work with secure soap messages sent over an encrypted VPN? – Our Man in Bananas Mar 9 '16 at 16:14

7 This answer does not produce soap envelope, it just serializes to xml. How to get soap envelope request? – Ali Karaca May 18 '18 at 14:20

Try [Fiddler2](#) it will let you inspect the requests and response. It might be worth noting that Fiddler works with both http and https traffic.

21

edited Nov 20 '08 at 16:04

answered Nov 19 '08 at 1:54



It is a https encrypted service – Andrew Harry Nov 19 '08 at 22:07

8 Fiddler can unencrypt the https traffic – Aaron Fischer Nov 20 '08 at 16:03

4 I found this solution to be better than adding tracing to the web/app config. Thank! – Andrew Harry Nov 19 '08 at 22:07



6

It looks like Tim Carter's solution doesn't work if the call to the web reference throws an exception. I've been trying to get at the raw web response so I can examine it (in code) in the error handler once the exception is thrown. However, I'm finding that the response log written by Tim's method is blank when the call throws an exception. I don't completely understand the code, but it appears that Tim's method cuts into the process after the point where .Net has already invalidated and discarded the web response.

I'm working with a client that's developing a web service manually with low level coding. At this point, they are adding their own internal process error messages as HTML formatted messages into the response BEFORE the SOAP formatted response. Of course, the automagic .Net web reference blows up on this. If I could get at the raw HTTP response after an exception is thrown, I could look for and parse any SOAP response within the mixed returning HTTP response and know that they received my data OK or not.

Later ...

Here's a solution that does work, even after an exception (note that I'm only after the response - could get the request too):

```
namespace ChuckBevitt
{
    class GetRawResponseSoapExtension : SoapExtension
    {
        //must override these three methods
        public override object GetInitializer(LogicalMethodInfo methodInfo,
        SoapExtensionAttribute attribute)
        {
            return null;
        }
        public override object GetInitializer(Type serviceType)
        {
            return null;
        }
        public override void Initialize(object initializer)
        {
        }

        private bool IsResponse = false;

        public override void ProcessMessage(SoapMessage message)
        {
            //Note that ProcessMessage gets called AFTER ChainStream.
            //That's why I'm looking for AfterSerialize, rather than BeforeDeserialize
            if (message.Stage == SoapMessageStage.AfterSerialize)
                IsResponse = true;
        }
    }
}
```

```

public override Stream ChainStream(Stream stream)
{
    if (IsResponse)
    {
        StreamReader sr = new StreamReader(stream);
        string response = sr.ReadToEnd();
        sr.Close();
        sr.Dispose();

        File.WriteAllText(@"C:\test.txt", response);

        byte[] ResponseBytes = Encoding.ASCII.GetBytes(response);
        MemoryStream ms = new MemoryStream(ResponseBytes);
        return ms;
    }
    else
        return stream;
    }
}
}

```

Here's how you configure it in the config file:

```

<configuration>
...
<system.web>
  <webServices>
    <soapExtensionTypes>
      <add type="ChuckBevitt.GetRawResponseSoapExtension, TestCallWebService"
          priority="1" group="0" />
    </soapExtensionTypes>
  </webServices>
</system.web>
</configuration>

```

"TestCallWebService" should be replaced with the name of the library (that happened to be the name of the test console app I was working in).

You really shouldn't have to go to ChainStream; you should be able to do it more simply from ProcessMessage as:

```

public override void ProcessMessage(SoapMessage message)
{
    if (message.Stage == SoapMessageStage.BeforeDeserialize)
    {
        StreamReader sr = new StreamReader(message.Stream);
        File.WriteAllText(@"C:\test.txt", sr.ReadToEnd());
    }
}

```



```
}  
}
```

If you look up `SoapMessage.Stream`, it's supposed to be a read-only stream that you can use to inspect the data at this point. This is a screw-up 'cause if you do read the stream, subsequent processing bombs with no data found errors (stream was at end) and you can't reset the position to the beginning.

Interestingly, if you do both methods, the `ChainStream` and the `ProcessMessage` ways, the `ProcessMessage` method will work because you changed the stream type from `ConnectStream` to `MemoryStream` in `ChainStream`, and `MemoryStream` does allow seek operations. (I tried casting the `ConnectStream` to `MemoryStream` - wasn't allow.)

So ..... Microsoft should either allow seek operations on the `ChainStream` type or make the `SoapMessage.Stream` truly a read-only copy as it's supposed to be. (Write your congressman, etc...)

One further point. After creating a way to retrieve the raw HTTP response after an exception, I still didn't get the full response (as determined by a HTTP sniffer). This was because when the development web service added the HTML error messages to the beginning of the response, it didn't adjust the `Content-Length` header, so the `Content-Length` value was less than the size of the actual response body. All I got was the `Content-Length` value number of characters - the rest were missing. Obviously, when .Net reads the response stream, it just reads the `Content-Length` number of characters and doesn't allow for the `Content-Length` value possibly being wrong. This is as it should be; but if the `Content-Length` header value is wrong, the only way you'll ever get the entire response body is with a HTTP sniffer (I use HTTP Analyzer from <http://www.ieinspector.com>).

edited Apr 10 '10 at 0:02

answered Apr 9 '10 at 18:41



Chuck Bevitt

199 3 3

2 I would prefer to have the framework do the logging for you by hooking in a logging stream which logs as the framework processes that underlying stream. The following isn't as clean as I would like it, since you can't decide between request and response in the `ChainStream` method. The following is how I handle it. With thanks to Jon Hanna for the overriding a stream idea

```
public class LoggerSoapExtension : SoapExtension  
{  
    private static readonly string LOG_DIRECTORY =  
        ConfigurationManager.AppSettings["LOG_DIRECTORY"];  
    private LogStream _logger;  
  
    public override object GetInitializer(LogicalMethodInfo methodInfo,
```

```

    }
    public override object GetInitializer(Type serviceType)
    {
        return null;
    }
    public override void Initialize(object initializer)
    {
    }
    public override System.IO.Stream ChainStream(System.IO.Stream stream)
    {
        _logger = new LogStream(stream);
        return _logger;
    }
    public override void ProcessMessage(SoapMessage message)
    {
        if (LOG_DIRECTORY != null)
        {
            switch (message.Stage)
            {
                case SoapMessageStage.BeforeSerialize:
                    _logger.Type = "request";
                    break;
                case SoapMessageStage.AfterSerialize:
                    break;
                case SoapMessageStage.BeforeDeserialize:
                    _logger.Type = "response";
                    break;
                case SoapMessageStage.AfterDeserialize:
                    break;
            }
        }
    }
}
internal class LogStream : Stream
{
    private Stream _source;
    private Stream _log;
    private bool _logSetup;
    private string _type;

    public LogStream(Stream source)
    {
        _source = source;
    }
    internal string Type
    {
        set { _type = value; }
    }
    private Stream Logger
    {
        get
        {

```

```

    {
        try
        {
            DateTime now = DateTime.Now;
            string folder = LOG_DIRECTORY + now.ToString("yyyyMMdd");
            string subfolder = folder + "\\ " + now.ToString("HH");
            string client = System.Web.HttpContext.Current != null &&
System.Web.HttpContext.Current.Request != null &&
System.Web.HttpContext.Current.Request.UserHostAddress != null ?
System.Web.HttpContext.Current.Request.UserHostAddress : string.Empty;
            string ticks = now.ToString("yyyyMMdd'T'HHmmss.ffffff");
            if (!Directory.Exists(folder))
                Directory.CreateDirectory(folder);
            if (!Directory.Exists(subfolder))
                Directory.CreateDirectory(subfolder);
            _log = new FileStream(new
System.Text.StringBuilder(subfolder).Append("\\ ").Append(client).Append('_').Append(ticks
            FileMode.Create);
        }
        catch
        {
            _log = null;
        }
    }
    _logSetup = true;
}
return _log;
}
}
public override bool CanRead
{
    get
    {
        return _source.CanRead;
    }
}
public override bool CanSeek
{
    get
    {
        return _source.CanSeek;
    }
}
public override bool CanWrite
{
    get
    {
        return _source.CanWrite;
    }
}
}

```

```

        get
        {
            return _source.Length;
        }
    }

    public override long Position
    {
        get
        {
            return _source.Position;
        }
        set
        {
            _source.Position = value;
        }
    }

    public override void Flush()
    {
        _source.Flush();
        if (Logger != null)
            Logger.Flush();
    }

    public override long Seek(long offset, SeekOrigin origin)
    {
        return _source.Seek(offset, origin);
    }

    public override void SetLength(long value)
    {
        _source.SetLength(value);
    }

    public override int Read(byte[] buffer, int offset, int count)
    {
        count = _source.Read(buffer, offset, count);
        if (Logger != null)
            Logger.Write(buffer, offset, count);
        return count;
    }

    public override void Write(byte[] buffer, int offset, int count)
    {
        _source.Write(buffer, offset, count);
        if (Logger != null)
            Logger.Write(buffer, offset, count);
    }

    public override int ReadByte()
    {

```

```

        return ret;
    }
    public override void Close()
    {
        _source.Close();
        if (Logger != null)
            Logger.Close();
        base.Close();
    }
    public override int ReadTimeout
    {
        get { return _source.ReadTimeout; }
        set { _source.ReadTimeout = value; }
    }
    public override int WriteTimeout
    {
        get { return _source.WriteTimeout; }
        set { _source.WriteTimeout = value; }
    }
    }
}
[AttributeUsage(AttributeTargets.Method)]
public class LoggerSoapExtensionAttribute : SoapExtensionAttribute
{
    private int priority = 1;
    public override int Priority
    {
        get
        {
            return priority;
        }
        set
        {
            priority = value;
        }
    }
    public override System.Type ExtensionType
    {
        get
        {
            return typeof(LoggerSoapExtension);
        }
    }
}
}

```

answered Mar 9 '09 at 17:33

TimCarter



Here's a simplified version of the top answer. Add this to the `<configuration>` element of your `web.config` or `App.config` file. It will create a `trace.log` file in your project's `bin/Debug` folder. Or, you can specify an absolute path for the log file using the `initializeData` attribute.

1

```
<system.diagnostics>
  <trace autoflush="true"/>
  <sources>
    <source name="System.Net" maxdatasize="9999" tracemode="protocolonly">
      <listeners>
        <add name="TraceFile" type="System.Diagnostics.TextWriterTraceListener"
initializeData="trace.log"/>
      </listeners>
    </source>
  </sources>
  <switches>
    <add name="System.Net" value="Verbose"/>
  </switches>
</system.diagnostics>
```

It warns that the `maxdatasize` and `tracemode` attributes are not allowed, but they increase the amount of data that can be logged, and avoid logging everything in hex.

answered Jan 3 at 17:31



[Collin Anderson](#)

11.8k 6 52 50

You haven't specified what language you are using but assuming C# / .NET you could use [SOAP extensions](#).

0

Otherwise, use a sniffer such as [Wireshark](#)

edited Jan 13 '10 at 13:35



[Anton Gogolev](#)

106k 37 189 276

answered Nov 19 '08 at 1:43



[rbrayb](#)

39.4k 30 106 151

1 Yeah, tried wireshark, it can only show me the header information, the soap content is encrypted. – [Andrew Harry](#) Nov 19 '08 at 22:08



-1

I realize I'm quite late to the party, and since language wasn't actually specified, here's a VB.NET solution based on Bimmerbound's answer, in case anyone happens to stumble across this and needs a solution. Note: you need to have a reference to the stringBuilder class in your project, if you don't already.

```
Shared Function returnSerializedXML(ByVal obj As Object) As String
    Dim xmlSerializer As New System.Xml.Serialization.XmlSerializer(obj.GetType())
    Dim xmlSb As New StringBuilder
    Using textWriter As New IO.StringWriter(xmlSb)
        xmlSerializer.Serialize(textWriter, obj)
    End Using

    returnSerializedXML = xmlSb.ToString().Replace(vbCrLf, "")

End Function
```

Simply call the function and it will return a string with the serialized xml of the object you're attempting to pass to the web service (realistically, this should work for any object you care to throw at it too).

As a side note, the replace call in the function before returning the xml is to strip out vbCrLf characters from the output. Mine had a bunch of them within the generated xml however this will obviously vary depending on what you're trying to serialize, and i think they might be stripped out during the object being sent to the web service.

edited Oct 10 '15 at 5:06

answered Oct 9 '15 at 16:48

[user2366842](#)

1,213 14 23

To whoever slapped me with the downvote, please feel free to let me know what I'm missing/what can be improved. – [user2366842](#) Jul 31 '18 at 13:12







