XML/A STREAM COMPRESSION – CLIENT PERSPECTIVE

By Alex Zivkovic – Intellimerce Inc. & Nikola Petrović – Morena Engineering

Published: 2004-04-28

One of the often-cited reasons for not adopting XML/A as the protocol for client/server business intelligence communication is the fact that XML and web services are considered verbose. The payload for XML/A web service is an XML structure. Indeed the XML structures are very liberal with use of <> and even looking at the structures one can easily see means of compressing it. Fortunately the process of compressing and decompressing web service calls and therefore XML/A is understood and accounted for through configuration options of most web servers. Both Apache and IIS 6.0 have capabilities that provide for compression of http streams and consequently the web services and XML/A that might sit on top of it. In this paper we describe how one can write a client that consumes the said compressed XML/A stream that originates on Microsoft XML/A SDK sitting on top of an IIS 6.0 server. While we discuss the XML/A compression specifically the same methodology can be used for any web service that is configured for compression and the code is not specific to XML/A.

What Is HTTP Compression?

HTTP Compression is process where the http stream served from the Web server is compressed on the server side and decompressed on client side. Web server does not blindly compress the stream Instead it checks to see if the consumer (client application or browser) is able to process a compressed response. Compression enabled consumers add special "Accept-Encoding" header in the request. If the compression is enabled the web server serves a compressed version of the registered type of files for compression.

For static content the server checks to see if the compressed file exists in a temporary compressed cache and serves it from there or if it is the first time that the file is being served compresses it, serves it and stores it in the cache. For dynamic content such as asp.net forms the server simply compresses the response and sends it to the requestor.

Compressing XML/A SDK Response on IIS

One of the improvements in IIS 6.0 is excellent compression implementation, so there's no need any more for third-party compression add-ins.

To compress the response from Microsoft XML/A SDK one simply needs to configure the IIS 6.0 virtual directory (typically called xmla) that contains the msxisapi.dll. To enable HTTP compression on your Windows 2003 Server just follow the step by step examples provided in following excellent article by Donnie Mack:

http://www.dotnetjunkies.com/HowTo/16267D49-4C6E-4063-AB12-853761D31E66.dcik

Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

Decompressing XML/A SDK Response in Client Code

Modern browsers, like IE and Netscape, automatically send compression request header, recognize the compressed streams and apply the appropriate decompression using the built-in compression libraries. However, if web service consumer is not a browser but your .Net code, dealing with compression on client side is your responsibility. In this section we describe how the decompression is achieved using the open source (LGPL) ICSharpCode Zip library (http://www.ICSharpCode.com/Zip) and .net.

Requesting compression

In order for the server to know that we are requesting the compressed stream we must modify the http request by adding the notification for compressed encoding. The cleanest cut point for modifying the http request when calling a SOAP service in .net is in the References.cs file that you will find hides behind the

`Web Services/XMLAInterface/References.map` (assuming that you have named your import of XML/A wsdl as XMLAInterface). Note that this file gets regenerated whenever you import a web service so make sure that your changes are stored safely so that you can reapply them if there is a change to the WSDL for the service that you are calling.

The first step is to modify the GetWebRequest function for compressed case. Here is the code that performs this function:

```
protected override System.Net.WebRequest GetWebRequest(System.Uri uri)
{
	System.Net.WebRequest request = base.GetWebRequest(uri);
	if (wantCompression)
		request.Headers.Add("Accept-Encoding", "gzip");
	return request;
}
```

As you can see for the compressed case (replace wantCompression with your variable) we add the header with the indication that we accept “gzip” encoding. Now the web server will know that we can process compressed web service streams.

Processing the compressed response

In order to process the compressed web response from the web service we modified the References.cs GetWebResponse method as shown:

```
protected override System.Net.WebResponse GetWebResponse(System.Net.WebRequest request)
{
	HttpWebResponseDecompressed response = new HttpWebResponseDecompressed(request);
	return response;
}
```

All the magic occurs in the HttpWebResponseDecompressed class which is described next.

HttpWebResponseDecompressed Class

Here is the code for the HttpWebResponseDecompressed class (adaptation of Jacek Chmiel code http://www.dotnetjunkies.com/Tutorial/90D3B3E0-6544-4594-B3BA-E41D8F381324.dcik):

Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

```
using System;
using System.IO;
using System.Net;
using ICSharpCode.SharpZipLib.GZip;

namespace SnowflakeNS
{
	internal class HttpWebResponseDecompressed:WebResponse {
		private System.Net.HttpWebResponse response;
		public HttpWebResponseDecompressed(WebRequest request) {
			response=(HttpWebResponse)request.GetResponse();
		}
		public override Stream GetResponseStream(){
			Stream compressedStream = null;
			if (response.ContentEncoding=="gzip"){
				compressedStream = new
GZipInputStream(response.GetResponseStream());
			}
			else if (response.ContentEncoding=="deflate"){
			}
			if (compressedStream != null){
				// decompress
				MemoryStream decompressedStream = new MemoryStream();

				int totalSize=0;
				int size = 2048;
				byte[] writeData = new byte[2048];
				while (true) {
					size = compressedStream.Read(writeData, 0, size);
					totalSize+=size;
					if (size > 0) {
						decompressedStream.Write(writeData, 0, size);
					}
					else {
						break;
					}
				}
				decompressedStream.Seek(0, SeekOrigin.Begin);
				response.Close();// Jacek, you have missed this line
				return decompressedStream;
			}
			else
				return response.GetResponseStream();
		}
		public override long ContentLength {
			get{return response.ContentLength;}
		}
		public override string ContentType {
			get {return response.ContentType;}
		}
		public override System.Net.WebHeaderCollection Headers{
			get{return response.Headers;}
		}
		public override System.Uri ResponseUri{
			get{return response.ResponseUri;}
		}
	}
}
```


Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

The constructor does nothing more than call the GetResponseStream

```
public HttpWebResponseDecompressed(WebRequest request) {
	response=(HttpWebResponse)request.GetResponse();
}
```

All the heavy lifting is really done by the ICSharpCode SharpZipLib library. At this point we really must thank the guys from ICSharpCode for a wonderful open source LGPL library. For more details visit their site at http://www.icshaprcode.net.

So lets look at the GetResponseStream() few line at a time. First we check to see if we have to do any processing at all:

```
Stream compressedStream = null;
if (response.ContentEncoding=="gzip"){
	compressedStream = new
GZipInputStream(response.GetResponseStream());
}
else if (response.ContentEncoding=="deflate"){
}
```

If you remember we have set the ContentEncoding by requesting the gzip encoding in our request. The server returns this type of encoding in the response. If the encoding is gzip we create a GzipInputStream from the response stream otherwise we do nothing. The rest of the code is skipped if the encoding is not set.

So here comes the decompression part:

```
if (compressedStream != null){
	// decompress
	MemoryStream decompressedStream = new MemoryStream();

	int totalSize=0;
	int size = 2048;
	byte[] writeData = new byte[2048];
	while (true) {
		size = compressedStream.Read(writeData, 0, size);
		totalSize+=size;
		if (size > 0) {
			decompressedStream.Write(writeData, 0, size);
		}
		else {
			break;
		}
	}
	decompressedStream.Seek(0, SeekOrigin.Begin);
	response.Close();
	return decompressedStream;
}
else
	return response.GetResponseStream();
```

This is the regular code that you would write for decompressing any gzip stream. If we don't have to decompress we just return the original stream.

The rest of the functions are just accessors.

Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

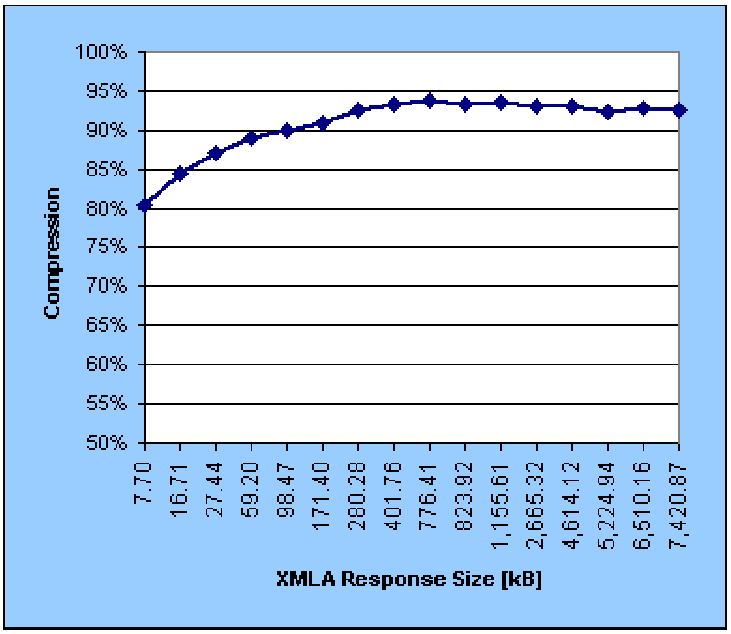
Compression Results

With all this work it is interesting to see what are the results of the compression are. To that effect we have run various EXECUTE and DISCOVER methods with different size of responses. Here are the charts showing that the compression effectiveness grows to an enviable 93 and 95 % respectively for responses.

Compression Test – Execute Method

XMLA Response Size [kB]

Without compression	Compressed	Difference	Compression rate
7.70	1.51	6.20	**80.46%**
16.71	2.58	14.13	**84.56%**
27.44	3.56	23.87	**87.01%**
59.20	6.53	52.67	**88.96%**
98.47	9.88	88.58	**89.96%**
171.40	15.55	155.85	**90.93%**
280.28	20.94	259.33	**92.53%**
401.76	27.13	374.63	**93.25%**
776.41	47.83	728.58	**93.84%**
823.92	55.22	768.70	**93.30%**
1,155.61	75.65	1,079.97	**93.45%**
2,665.32	187.74	2,477.58	**92.96%**
4,614.12	314.64	4,299.48	**93.18%**
5,224.94	400.03	4,824.91	**92.34%**
6,510.16	473.23	6,036.93	**92.73%**
7,420.87	553.93	6,866.94	**92.54%**

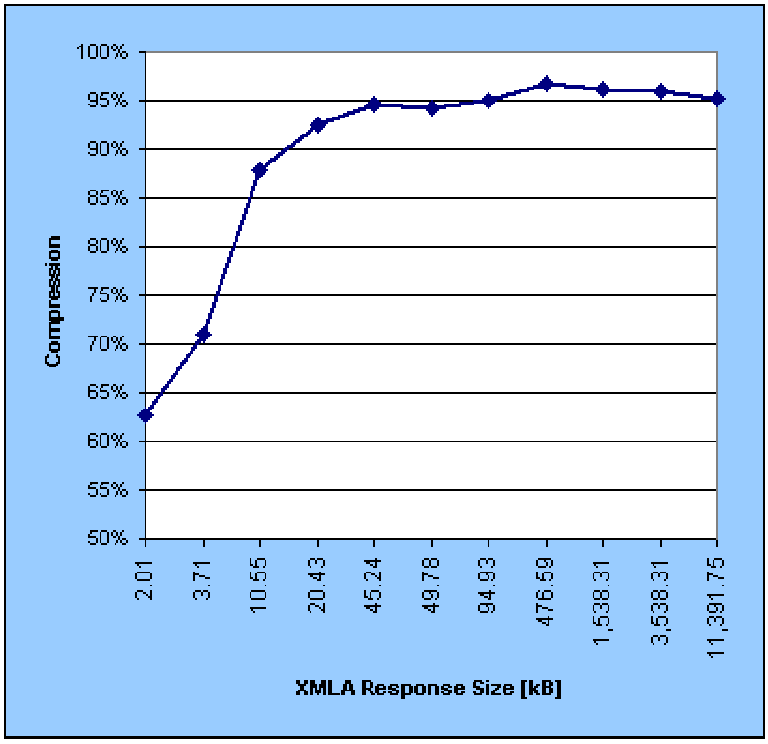


Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

Compression Test – Discover Method

XMLA Response Size [kB]

Without compression	Compressed	Difference	Compression rate
2.01	0.75	1.26	**62.74%**
3.71	1.08	2.63	**70.88%**
10.55	1.29	9.26	**87.82%**
20.43	1.54	18.89	**92.47%**
45.24	2.44	42.80	**94.60%**
49.78	2.92	46.86	**94.14%**
94.93	4.74	90.18	**95.00%**
476.59	15.83	460.76	**96.68%**
1,538.31	59.21	1,479.10	**96.15%**
3,538.31	141.13	3,397.18	**96.01%**
11,391.75	551.74	10,840.01	**95.16%**



Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com

That's All Folks

That is all there is to it. In a few lines of code and with the IIS or Apache compressed configuration you can greatly reduce network bandwidth for XML/A. Note that this methodology is not specific to XML/A. You can do this for any web service that is running on top of http compression. The compression levels for XML/A are staggering: in most cases 90% and higher yet the server overhead is low thanks to efficient compression libraries. For heavy-loaded servers you can consider a fast CPU or a web farm. In addition one should not forget that there is a processing overhead on both sides – web service (compression) and web service consumer (decompression). You could modify this example to access the built-in windows gzip library if you know that such is available i.e. assuming that you are running on a Windows server. (note: alternative might be mono project running .net on linux)

Intellimerce Inc.
439 King St. West, 4th Floor
Toronto, Ontario, M5V1K4
Tel.416.644.8755 Fax.416.644.8801
http://www.intellimerce.com