Audience Insight Client

September 18, 2013

# Introduction

The Audience Insight client is a component that accepts data in the form of Log objects from various sources, groups the logs into batches and sends a serialized version of the batch to the Audience Insight server.

**Log** – The core data unit in the Audience Insight system is a log. A log is a collection of fields, similar to a database row, representing a snapshot of data usually relating to a single subject (i.e. video quality, an advertising event, etc). A log typically has a type identifier and timestamp along with other relevant fields.

**Batch** – A batch is a group of logs. Logs are queued up on the client as they are generated, and then sent as a group, or batch, to the server. The batch is serialized to JSON before being transmitted.

Client App

Log

Audience Insight

Batch

Log

Log

…

Server

Batch

Log

Log

…

Serialized Batch

**Config/mapping file** – The Audience Insight client uses a config file to determine *how and which* log fields are mapped to serialized fields.

**Mapping Example:**

|  |  |  |
| --- | --- | --- |
| **Log Data** | **Mapping configuration** | **Mapped and serialized data** |
| MyNumber: 12345  MyFirstName: "john"  MyLastName: "doe" | <Log Type="CustomLog" Id="CustomLog">  <Value Name="MyNumber" Id="num" />  <Value Name="MyFirstName" Id="first" />  </Log> | {  num: 12345,  first: "john"  }  (Note that the field names were mapped and the field MyLastName was not included because it was not configured.) |

# Usage – Non-Player Framework Context

[See demo apps for full implementation]

Client App

Log

Audience Insight

Batch

Log

Log

…

Server

Batch

Log

Log

…

Serialized Batch

1. **Include:** Microsoft.AudienceInsight (via vsix, NuGet package, or source)
2. **Configure:**  
   <Log Type="CustomLog" Id="CustomLog">  
    <Value Name="Type" Id="typ" />  
    <Value Name="LogId" Id="lid" />  
    <Value Name="TimeStamp" Id="tik" />  
    <Value Name="CustomProperty" Id="s01" />  
    <Value Name="CustomPropertyNumber" Id="s02" />  
    <Value Name="CustomPropertyBool" Id="s03" />  
   </Log>
3. **Initialize:**

// Audience Insight config

var batchingConfig = await BatchingConfigFactory.Load(new Uri("ms-appx:///AudienceInsightConfig.xml"));

batchingLogAgent = new BatchingLogAgent(batchingConfig);

1. **Log data:**

var customLog = new CustomLog()

{

CustomProperty = "testing",

CustomPropertyNumber = 3.14159,

CustomPropertyBool = true

};

batchingLogAgent.LogEntry(customLog);

# Usage – Player Framework Context

[See demo apps for full implementation]

[See separate architecture diagram]

Client App

Audience Insight

Batch

Log

Adapter:

Microsoft  
.VideoAnalytics  
.AudienceInsight

…

Server

Batch

Log

…

Serialized Batch

Player Framework

Video Analytics

Data Clients  
(Adaptive monitor, Edge server monitor, etc.)

1. **Include:**
   1. **Microsoft.AudienceInsight** (via vsix, NuGet package, or source)
   2. **Microsoft.VideoAnalytics.AudienceInsight** - Adapter that allows the Player Framework's VideoAnalytics library to be an Audience Insight log source. (via vsix, NuGet package, or source)
   3. **Relevant Player Framework libraries**  (via vsix, NuGet package, or source)
2. **Configure**: see example config file in demo app
3. **Initialize**

var configFileUrl = new Uri("ms-appx:///AudienceInsightConfig.xml");

// Audience Insight config

var batchingConfig = await BatchingConfigFactory.Load(configFileUrl);

var batchinglogAgent = new BatchingLogAgent(batchingConfig);

var aiLoggingTarget = new AudienceInsightLoggingTarget(batchinglogAgent);

Microsoft.VideoAnalytics.LoggingService.Current.LoggingTargets.Add(aiLoggingTarget);

// Player Framework analytics config

var analyticsConfig = await AnalyticsConfig.Load(configFileUrl);

var analyticsPlugin = new AnalyticsPlugin(analyticsConfig);

var adaptivePlugin = player.Plugins.OfType<AdaptivePlugin>().FirstOrDefault();

var adaptiveMonitorFactory = new AdaptiveMonitorFactory(adaptivePlugin.Manager);

var edgeServerMonitor = new EdgeServerMonitor();

analyticsPlugin.AdaptiveMonitor = adaptiveMonitorFactory.AdaptiveMonitor;

analyticsPlugin.EdgeServerMonitor = edgeServerMonitor;

player.Plugins.Add(analyticsPlugin);

1. **Log Data**
   1. The PlayerFramework Video Analytics component gathers data via several data clients including a PlayerMonitor, AdaptiveMonitor and an EdgeServerMonitor. Data from these sources is exposed via **numerous built-in log and report types** – see separate spreadsheed describing the available log types.
   2. **Custom logs and logging sources** may be created  
        
      analyticsPlugin.AnalyticsCollector.LoggingSources.Add(/\* custom ILoggingSource here \*/);  
      or  
      analyticsPlugin.AnalyticsCollector.SendLog(/\* custom ILog here \*/);  
      or  
      LoggingService.Current.LoggingSources.Add(/\* custom ILoggingSource here \*/);
   3. **Session-based values** can be included with every log:

var analyticsPlugin = new AnalyticsPlugin(analyticsConfig);

var geo = new Geolocator();

var location = await geo.GetGeopositionAsync();

analyticsPlugin.SessionData.Add("Latitude", location.Coordinate.Latitude);

analyticsPlugin.SessionData.Add("Longitude", location.Coordinate.Longitude);  
analyticsPlugin.SessionData.Add("Accuracy", location.Coordinate.Accuracy);

* 1. **Playlist-based values** can be included with every log:

var playlistData = new Dictionary<string, object>();

playlistData.Add("myKey", "myValue");

var playlistItem = new PlaylistItem()

{

Source = new Uri("http://maincontenturl")

};

playlistItem.SetValue(Analytics.AdditionalDataProperty, playlistData);

var playlistPlugin = player.Plugins.OfType<PlaylistPlugin>().First();

playlistPlugin.Playlist.Add(playlistItem);

playlistPlugin.GoToNextPlaylistItem();

# Usage – Logging Video Advertising Events

[See demo apps for full implementation]

[See separate architecture diagram]

Client App

Audience Insight

Batch

Log

Adapter:

Microsoft  
.VideoAnalytics  
.AudienceInsight

…

Server

Batch

Log

…

Serialized Batch

Player Framework

Video Analytics

Video Advertising

Adapter:  
Microsoft.  
VideoAnalytics.  
VideoAdvertising

1. **Include:**
   1. **Microsoft.VideoAnalytics.VideoAdvertising** – Adapter that allows the Player Framework's VideoAdvertising library to be a VideoAnalytics log source
2. **Configure**:   
     
   <Log Type="AdEvent" Id="AdEvent">

<Value Name="Type" Id="typ" />

<Value Name="LogId" Id="lid" />

<Value Name="TimeStamp" Id="tik" />

<Value Name="TrackingType" Id="s01" />

<Value Name="MediaSource" Id="s02" />

<Value Name="MediaSourceType" Id="s03" />

<Value Name="CreativeExtraInfo" Id="s04" />

<Value Name="CreativeSourceType" Id="s05" />

<Value Name="CreativeId" Id="s06" />

<Value Name="CurrentPosition" Id="s07" />

</Log>

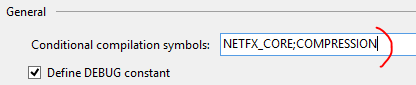
1. **Initialize**

// Audience Insight ad tracking config

analyticsPlugin.AnalyticsCollector.LoggingSources.Add(new AdvertisingLoggingSource(player.GetAdHandlerPlugin().AdHandlerController));

1. **Log Data** – Data is logged automatically, when an ad tracking even occurs.

# Development notes

1. **Demo applications**
   1. **Win8** – One solution with four projects: \Win8.AudienceInsight\Samples\AudienceInsightSamples.Win8.sln
      1. Xaml.Standalone
      2. Xaml.PlayerFramework
      3. JS.Standalone
      4. JS.PlayerFramework
   2. **WP8** – One solution with two projects:  
      \Phone.AudienceInsight\Samples.WP8\AudienceInsightSamples.WP8.sln
      1. StandaloneSample
      2. PlayerFrameworkSample
   3. **WP7** – One solution with two projects:  
      \Phone.AudienceInsight\Samples.WP7\AudienceInsightSamples.WP7.sln
      1. StandaloneSample
      2. PlayerFrameworkSample
2. **Server response**– The server can pass several control parameters back to the client via HTTP header in the response.
   1. **LoggingEnabled** – This flag allows the server to stop logging on the client. Possible values: 0, 1.  
        
      var response = new HttpResponseMessage(HttpStatusCode.OK);  
      response.Headers.Add("LoggingEnabled", "0");
   2. **QueuePollingIntervalSeconds** – Sets the interval at which log batches are to be sent.  
        
      var response = new HttpResponseMessage(HttpStatusCode.OK);  
      response.Headers.Add("QueuePollingIntervalSeconds", "3");
   3. **ServerTime** – Provides the server time to the client so timestamps can be adjusted accordingly.  
        
      var response = new HttpResponseMessage(HttpStatusCode.OK);  
      response.Headers.Add("ServerTime", DateTimeOffset.Now.Ticks.ToString());
3. **Build** – Audience Insight components are built during the [main Player Framework build process](http://playerframework.codeplex.com/wikipage?title=Source%20Code%20and%20Build%20Scripts&referringTitle=Documentation).
   1. Generated **vsix packages (Win8, WP8)** located in \Build\Microsoft.PlayerFramework:
      1. Microsoft.AudienceInsight.Win8.vsix
      2. Microsoft.AudienceInsight.WP8.vsix
      3. Microsoft.PlayerFramework.Win8.Analytics.AudienceInsight.vsix
      4. Microsoft.PlayerFramework.WP8.Analytics.AudienceInsight.vsix
   2. Generated **NuGet packages (WP8 only)** located in \Build.NuGet:
      1. Microsoft.AudienceInsight.1.0.0.nupkg
      2. Microsoft.VideoAnalytics.AudienceInsight.1.3.0.nupkg
4. **Compression** – Compression can be used if necessary. To enable compression:
   1. **All platforms**: build Microsoft.AudienceInsight library with a “COMPRESSION” compiler symbol.  
      
   2. **All platforms**: enable via config file: <compression>1</compression>
   3. **Win8**: no further modifications.
   4. **WP8**: Install compression library in the Microsoft.WP8.AudienceInsight project via NuGet: <http://www.nuget.org/packages/Microsoft.Bcl.Compression/>
   5. **WP7**: Include SharpCompress.Portable.dll (Ms-PL) in the Microsoft.WP7.AudienceInsight project. <http://sharpcompress.codeplex.com/>
   6. **Server:** Add a message handler to WebApiConfig.Register:  
        
      config.MessageHandlers.Add(new GzipToJsonHandler());  
        
      See GzipToJsonHandler below.
5. **Local Azure setup for WP testing** – Since WP8 cannot use “localhost” to access a service running on the debugging PC, the service must be configured to be listening on the PC's IP address not just localhost. To configure the Azure emulator to do this, edit C:\Program Files\Microsoft SDKs\Windows Azure\Emulator\devfabric\DevFC.exe.config and restart the emulator.  
     
   <!--<add key="VipPoolStartIPAddress" value="127.0.0.1" />  
   <add key="VipPoolEndIPAddress" value="127.0.0.255" />-->  
   <add key="VipPoolStartIPAddress" value="your.ip.address.here" />  
   <add key="VipPoolEndIPAddress" value="your.ip.address.here" />
6. **Previous documentation**
   1. <http://playerframework.codeplex.com/wikipage?title=Analytics>
   2. <http://playerframework.codeplex.com/wikipage?title=Custom%20Logging%20Target>

## GzipToJsonHandler

using System;

using System.Collections.Generic;

using System.IO;

using System.IO.Compression;

using System.Linq;

using System.Net.Http;

using System.Threading;

using System.Threading.Tasks;

using System.Web;

namespace Microsoft.Evangelist.Mvc.Analytics

{

public class GzipToJsonHandler : DelegatingHandler

{

protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)

{

if (request.Content.Headers.ContentType == null || request.Content.Headers.ContentType.MediaType != "application/gzip")

return base.SendAsync(request, cancellationToken);

Stream outputStream = new MemoryStream();

request.Content.ReadAsStreamAsync().ContinueWith(t =>

{

var compressedStream = t.Result;

using (var uncompressedStream = new GZipStream(compressedStream, CompressionMode.Decompress, true))

{

var sr = new StreamReader(uncompressedStream, System.Text.Encoding.UTF8);

uncompressedStream.CopyTo(outputStream);

outputStream.Seek(0, SeekOrigin.Begin);

}

}).Wait();

HttpContent origContent = request.Content;

request.Content = new StreamContent(outputStream);

foreach (var header in origContent.Headers)

request.Content.Headers.Add(header.Key, header.Value);

request.Content.Headers.Remove("Content-Type");

request.Content.Headers.Add("Content-Type", "application/json");

return base.SendAsync(request, cancellationToken);

}

}

}