StreamhubAnalytics generic plugin for Silverlight web applications

For Silverlight web applications we provide a C# plugin and a sample video player application that uses the plugin.

[Download the plugin]

The plugin is composed of the following:

1. StreamhubAnalytics.cs which solely contains the API functions to call from your own Silverlight application's code.
2. NetworkUI.cs which interact with our remote Backend. You don't need to call its method directly.
3. Storage.cs which contains cookie management code

Getting Started

To get you started quickly, the downloadable archive contains a Silverlight basic video player application which integrates our plugin.

The application does not have external dependencies outside the .NET framework (>= 4.0.0) and Silverlight 5.

In the MainPage.xaml file, point the Source attribute of the MediaElement to a valid .wmv video file :

```
<MediaElement x:Name="playerView"
                      HorizontalAlignment="Left"
                      AutoPlay="True"
                      Source="C:/Users/user/Downloads/ELL_PART_5_768k.wmv"
                      Height="320" Width="480" VerticalAlignment="Top" />
```

And then Run/Debug the application. You should see logs starting with "StreamhubAnalytics..." in the output console window.

You can browse around the code of the MainPage.xaml.cs to get an overview of what needs to be implemented, which functions to call and in which order.

Integrate the plugin in your own application

The integration process is quite easy and straightforward, because the complexity of interacting with our Backend REST API has been abstracted in the plugin logic.

Add the plugin files

Locate the 3 plugin files StreamhubAnalytics.cs, NetworkUI.cs and Storage.cs in the archive and copy them into your Silverlight project.

Create an instance of the StreamhubAnalytics class

From your application code, create an instance of the StreamhubAnalytics class and save it into a class member.

C#

```
// 1) Initialize the StreamhubAnalytics plugin
            _sh = new StreamhubAnalytics("streamhub-5812d",
                                         "jwplatform", null, "silverlight-generic-plugin",
false, "");
```

Parameters details

name	type	comments
analyticsId	String	The main tracking code that has been provided to you by your Streamhub integration engineer.
partnerId	String	provided to you by your Streamhub integration engineer
endPoint	String	http://stats.streamhub.io is the production endPoint for submitting stats. However, you could choose to use your own mockup server during the plugin development phase.
playerId	String	unique identifier of the player playing the video. It is a unique identifier for the player in both your system and ours. If any clarification is needed, please contact your Streamhub integration engineer.
isLive	Boolean	specify if the video content is either a Live or a VoD program. If the content is Live, then, we usually require that you provide a channel identifier for the programs being played by your video player. See the API method **setChannelId** below.

		For any details, please contact your Streamhub integration engineer.
userId	String	is used if you are able to track your user via a user identifier. If so, you will be able to see user-specific analytics (How old is the public watching the program XYZ ? Which gender ? Which location ? so forth...)

Call the API methods

setChannelId (required for live only)

If your are playing Live video content, then you probably have EPG metadata relating to it. Usually, in the EPG metadata, a program belong to a channel. We require you to provide the channel identifier associated with the playing program if you have initialized the StreamhubAnalytics library with *isLive: true*.

C#
```
_sh.setChannelId("a-channel-unique-identifier-mapping-to-some-epg-content");
```

setDuration (optional, vod or live with time-shifting)

If your video content has a known duration, then you should provide it to the plugin so that it will be able to report stats with completion rates events. Completion rates will give you insight on how much of your videos are actually watched by your end users.

C#
```
_sh.setDuration(seconds);
```

onMediaReady (required)

onMediaReady must be called to provide the unique identifier of the media about to be played.

C#
```
_sh.onMediaReady("EducationalSpeech");
```

onMediaStart (required)

onMediaStart must be called the first time your video content starts to play.

C#
```
_sh.onMediaStart();
```

onTick (required)

onTick is a convenient method that abstract the logic of measuring accurately the video watching time.
The sample app create a thread to read the video playback current time at regular interval.
This is the perfect place to call the *onTick* api method.
You should call *onTick* during the whole duration of the video playback.
The unique parameter has to be provided in seconds.

C#
```
_sh.onTick(seconds);
```

setPaused (optional)

Call this method when media is paused or unpaused, providing the appropriate boolean value.

C#
```
_sh.setPaused( true );
```

onMediaComplete (required)

Call *onMediaComplete* to report a video completed event.
The state of the plugin is reset when you call *onMediaComplete*.
As a result, *onMediaReady, onMediaStart, setDuration* should be called again to process the next video if any.

C#
```
_sh.onMediaComplete()
```

Sample code

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Diagnostics;
using System.Windows.Threading;

using streamhub;
```

```
namespace player_plugin_silverlight_project
{
    public partial class MainPage : UserControl
    {
        DispatcherTimer _dispatcherTimer;
        StreamhubAnalytics _sh;

        public MainPage()
        {
            InitializeComponent();
            observe();
        }

        private void observe()
        {
            // 1) Initialize the StreamhubAnalytics plugin
            _sh = new StreamhubAnalytics("streamhub-5812d",
                                         "jwplatform", null, "silverlight-generic-plugin",
false, "");

            playerView.MediaOpened += new RoutedEventHandler(onMediaOpened);
            playerView.MediaEnded += new RoutedEventHandler(onMediaEnded);
        }

        private void onMediaOpened(object sender, RoutedEventArgs e)
        {
            int mediaDuration = (int)playerView.NaturalDuration.TimeSpan.TotalSeconds;

            // 2) Call onMediaReady() and provide the required publicId, then call
onMediaStart() and finally
            // call setDuration()  providing the media duration in seconds
            _sh.onMediaReady("EducationalSpeech");
            _sh.onMediaStart();
            _sh.setDuration(mediaDuration);

            // Monitor playhead's position
            _dispatcherTimer = new DispatcherTimer();
            _dispatcherTimer.Tick += new EventHandler(onTimerTick);
            _dispatcherTimer.Interval = new TimeSpan(0, 0, 1);
            _dispatcherTimer.Start();
        }

        private void onTimerTick(object sender, EventArgs e)
        {
            Debug.WriteLine("position: " + (int)playerView.Position.TotalSeconds);

            int position = (int)playerView.Position.TotalSeconds;

            // 3) call onTick() until the end of the media, providing the playhead's
position in seconds
            // since the beginning of the media
            _sh.onTick(position);
        }
```

```
        private void onMediaEnded(object sender, RoutedEventArgs e)
        {
            // 4) when the media is finished, call onMediaComplete()
            _sh.onMediaComplete();

            _dispatcherTimer.Stop();

            Debug.WriteLine("Media Ended");
        }

        private void pause(object sender, RoutedEventArgs e)
        {
            // Call setPaused() for complimentary reporting
            if (playerView.CurrentState == MediaElementState.Playing)
            {
                playerView.Pause();
                _sh.setPaused(true);
            }
            else
            {
                playerView.Play();
                _sh.setPaused(false);
            }
        }

        private void seek_btn_Click(object sender, RoutedEventArgs e)
        {
            playerView.Position = TimeSpan.FromSeconds(140);
        }
    }
}
```