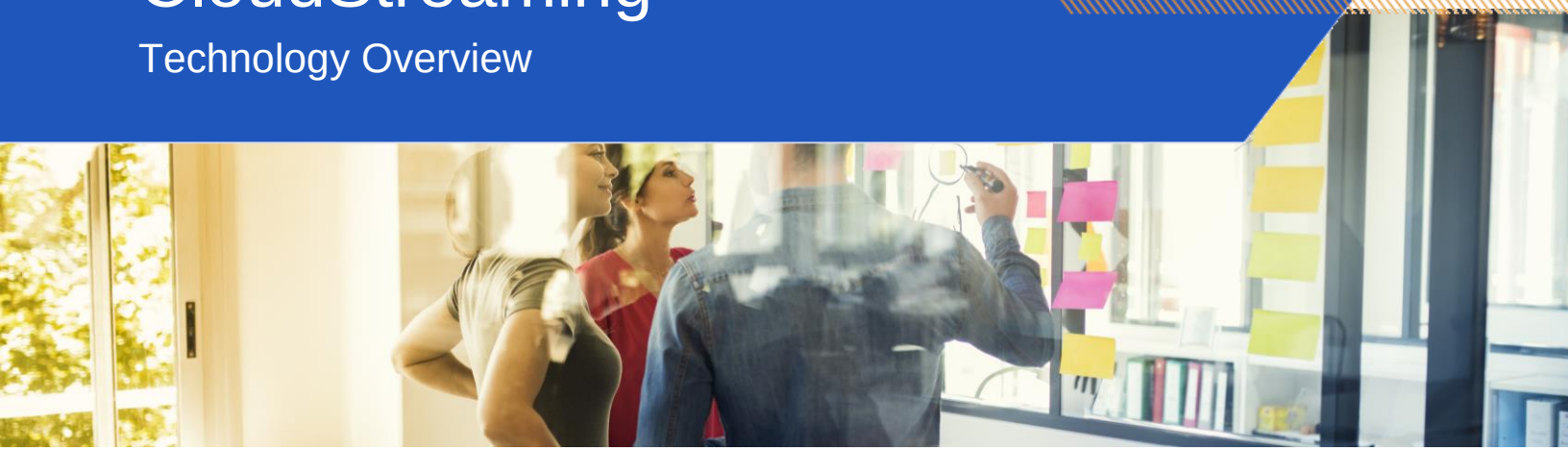
xignite

CloudStreaming

Technology Overview



Objectives

Xignite CloudStreaming is a zero-footprint solution for streaming financial market data directly to end-user devices such as mobile and web apps. Customers can easily write and deploy streaming financial apps using open technology standards. Unlike traditional streaming solutions, Xignite CloudStreaming eliminates the need for customers to build, implement, deploy, and operate a fan-out production infrastructure required to ingest, process, and distribute data to their end-user apps.

Xignite CloudStreaming leverages Xignite's Market Data Cloud, a high-performance, scalable, and performant production cloud infrastructure that serves a broad array of financial market data including real-time and delayed prices across equities, funds, bonds, futures, options, currencies, and metals.

Terminology

We use these terms to describe Xignite CloudStreaming:

Term	Definition	Example
Customer	An Xignite customer. A company that has a business relationship with Xignite.	Personal Capital
End-user	An end-user of a customer. An end-user has a business relationship with a customer and not Xignite, which includes end-users of a customer's public website.	A Personal Capital end-user

Zero Footprint

Xignite CloudStreaming delivers an API that customers can call directly from a client app to stream financial market data. By leveraging the Xignite Market Data Cloud, customers can stream data with minimal costs. No need to purchase and deploy hardware, build data feed handlers, implement a streaming infrastructure, or hire the necessary engineering and operations staff.

The Xignite Market Data Cloud is based on Amazon Web Services (AWS) and is a culmination of several years of engineering investment in creating and optimizing a financial market data distribution platform on AWS. The Xignite Market Data Cloud delivers the necessary performance, scalability, and redundancy that mission-critical financial applications require and can reliably count on: typical response times of less than 100 msec, serving over 50B hits per month, and with an uptime exceeding 99.99%.

Xignite CloudStreaming runs on the Xignite Market Data Cloud and benefits from the same performance, scalability, and redundancy that powers our Xignite CloudAPIs, a broad library of easy-to-use financial market data REST APIs. The same broad array of market data currently available through Xignite CloudAPIs can now be made available through Xignite CloudStreaming.

Easy to Implement

As with Xignite CloudAPIs, the primary objective of Xignite CloudStreaming is ease of use. In addition to providing a zero-footprint streaming solution, ease-of-use means making it as simple as possible for an application developer to display financial data.

Xignite CloudStreaming is as simple as Xignite CloudAPIs; both use HTTP GET, the same URLs, and the same JSON output. For example, you can use Xignite CloudAPIs or Xignite CloudStreaming to retrieve the latest currency rates using the following URLs. Note that the only difference is the subdomain.

Xignite CloudAPI URL

```
https://globalcurrencies.xignite.com/xGlobalCurrencies.json/GetRealTimeRates\
     ?Symbols=EURUSD,USDJPY
```

Xignite CloudStreaming URL

```
https://stream.xignite.com/xGlobalCurrencies.json/GetRealTimeRates\
     ?Symbols=EURUSD,USDJPY
```

The data returned by Xignite CloudAPIs and Xignite CloudStreaming are the same as well. Note that the only difference is that Xignite CloudStreaming does not return a <delay> field, a field whose value reflects the time it takes to process an Xignite CloudAPI request.

Xignite CloudAPI JSON output


```
[{"Outcome":"Success","Message":null,"Identity":"Cookie","Delay":0.1844602,"BaseCur
rency":"EUR","QuoteCurrency":"USD","Symbol":"EURUSD","Date":"12/09/2015","Time":"2:
22:45 AM","QuoteType":"Spot","Bid":1.09013,"Mid":1.090155,...
```

Xignite CloudStreaming JSON output

```
[{"Outcome":"Success","Message":null,"Identity":"Cookie","BaseCurrency":"EUR","Quot
eCurrency":"USD","Symbol":"EURUSD","Date":"12/09/2015","Time":"2:22:45
AM","QuoteType":"Spot","Bid":1.09013,"Mid":1.090155,...
```

Open Standards

Xignite CloudStreaming streams data using Server Sent Events (SSE), also known as HTTP Streaming, which relies on the same network protocol (HTTP) for serving web pages over the Internet. SSE is an HTML5 standard, which the current version of all major browsers, with the exception of Internet Explorer, provide native support for SSE. For Internet Explorer, there are open source extensions available to add support for SSE.

Sample Code

Xignite CloudStreaming starts with making an HTTP GET request, which establishes an HTTP connection, over which SSE messages are received. These steps can be done using the HTML5 EventSource interface. The resulting JSON data can be interpreted using any open source library.

Simple javascript implementation

Here is some simple javascript code to demonstrate the two key concepts of Xignite CloudStreaming -- establishing the connection and receiving an SSE message. Note that in this example, only the first SSE message is received.

```
<script>
var eventSource = null
var data = [];

function connect()  {
 eventSource = new EventSource(\
   "https://stream.xignite.com/xGlobalCurrencies.json/GetRealTimeRates
 ?Symbols=EURUSD,USDJPY&_token=your_authentication_token&_token_userid
=you_userid");

 eventSource.addEventListener("data",
                              function(event) {data = event.data;} );
}

connect();
<\script>
```

This example:

- Streams EURUSD and USDJPY rates using XigniteGlobalCurrencies.
- Requires replacing “your_authentication_token” with a valid Xignite authentication token.
- Returns the data in JSON format.

Handling patch messages

SSE is an efficient messaging protocol for streaming, delivering smaller incremental messages with only field values that have changed with a message type of *patch*. A message type of *data* indicates a message with all field values.

This example builds on the first example to create a fully functional example that handles both *patch* and *data* message types.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/fast-json-patch/0.5.4/json-patch-duplex.min.js" />


<script>
var eventSource = null
var data = [];


function connect()  {
 eventSource = new EventSource(\
   "https://stream.xignite.com/xGlobalCurrencies.json/GetRealTimeRates\?
Symbols=EURUSD,USDJPY&_token=your_authentication_token&_token_userid=your_userid");


 eventSource.addEventListener("data",
                              function(event) {data = event.data;} );
}


 eventSource.addEventListener("patch",
                              function(event) {jsonpatch.apply(data,
                                               JSON.parse(event.data));} );
}
connect();
<\script>
```

This example:

- Introduces a second listener to receive *patch* messages.
- Includes an open source library for JSON patch messages.
- Interprets the field values from the *patch* message as JSON.
- Updates the data field values with those from the *patch* message.

Including error handling

A production implementation of Xignite CloudStreaming should include error handling. This example further builds on the example to include support for detecting errors and disconnecting.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/fast-json-patch/0.5.4/json-patch-duplex.min.js" />


<script>
var eventSource = null
var data = [];
```

```
function connect()  {
 eventSource = new EventSource(\
   "https://stream.xignite.com/xGlobalCurrencies.json/GetRealTimeRates
    ?Symbols=EURUSD,USDJPY&_token=your_authentication_token&_token_userid
=your_userid");

 eventSource.addEventListener("data",
                              function(event) {data = event.data;} );
}

 eventSource.addEventListener("patch",
                              function(event) {jsonpatch.apply(data,
                                               JSON.parse(event.data));} );

 eventSource.addEventListener("error",
                              function(event) {eventSource.close():} );
}

function disconnect() {
 if (eventSource != null {
   eventSource.close(); }
}

connect();
<\script>
```

This example:

- Introduces a disconnect function.
- Introduces a listener for an *error* message type.
- Disconnects in the event of receiving an error.

Using encrypted tokens

For security reasons, authentication tokens must not be exposed on client-side apps such as mobile and web apps. Authentication tokens need to be encrypted on the server side, with only encrypted tokens exposed on client-side apps.

For more information on how to generate an encrypted token, see *Client-side Xignite API request authentication*.