

Introduction

- Overview
- Target Readership
- Assumptions
- Scope of Document
- Terms of Use
- Terminology

Document Conventions

- HTTP URLs
- HTTP POST Request Formats
- XML Response Formats

REST Resources (Concepts)

- Implementation Example
- Resource Overview
- Users
- Session Tokens
- Channels

Data Management

- Subscriptions and Channels
- Constructing Channel Filters
- Price Data Filters

Streaming Data

- Summary
- Usage
- Reconnecting and Sequence Numbers
- 'Heartbeat Message'
- HTTP Chunked Transfer Encoding
- Data Streaming Implementation Example

Available Services (Code Examples)

- Help Resource
- User Resources
- Channel Resources
- Fixture Resource

Status and Error Codes

Implementation Examples (.NET / C#)

- Introduction
- User Resources Example
- Channel Resources Example
- Streaming Channel Resources Example

Further Help

- Frequently-asked Questions (FAQs)
- Support Information

Introduction

Overview

This document describes the basic operation of the Sporting Solutions HTTP Pull API service ("the API"). It introduces the concepts of Users, Channels and Streaming and gives code implementation examples for each.

Target Readership

This document is for programmers, web developers, project managers and other IT personnel who are implementing the API.

Assumptions

This document assumes that you have a good working knowledge of:

- XML;
- HTTP (including verbs, status codes and error codes);

- HTML

You can create all the HTTP requests described this document programmatically using any language that can generate HTTP/1.1-compliant requests. Sporting Solutions expects any customers or other third parties integrating with its API to use this approach and to be proficient in their selected programming language(s).

We assume you have a Sporting Solutions client account. If you do not, please contact your Sporting Solutions account manager for further information.

Scope of Document

This document describes several key concepts and gives a summary of the HTTP specification of the REST resources that are available. Customers can access these resources using various tools:

- HTTP GET requests against a resource URL using any browser; or
- HTTP PUT, DELETE and POST requests using a Web debugging tool.

The document also provides guidelines on how to incorporate the API into your own website and retrieve, filter and stream sports-related data. Please note that we do not make any suggestions or recommendations regarding generic web-page design.

This service has been developed according to [HTTP/1.1 Specifications](#) and uses a REST approach. (For details of REST practices, please consult any search engine or other online resource.)

The document demonstrates how to:

- log in and out as a User and view User Profile information;
- modify Channels;
- stream Channels

Terms of Use

This document and the material contained herein are confidential to the intended recipients. This document may not be used for any other purposes, reproduced in whole or in part, nor passed to any organization or person without the specific permission in writing of the Managing Director of Sporting Solutions.

Terminology

Document Conventions

HTTP URLs

Since the API is a REST interface all methods are expressed as a component of a URL together with an HTTP verb. This section demonstrates the conventional notation:

Request Type	GET
URL	<i>http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/{UserName}/Token</i>
Required Headers	X-Authentication-Token:{SessionToken}

You should interpret this section as

- an HTTP GET request on the URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/{**UserName**}/Token*

- requiring the HTTP header X-Authentication-Token:{SessionToken}.

In this example the parameters enclosed in braces are:

{UserName} Username provided by Sporting Solutions
{SessionToken} Session identifier provided by a Login request

Please note that the base URL may vary from that given in the example above.

HTTP POST Request Formats

HTTP POSTed data should be in XML format (similar to the examples in this document). All POST statements require a Content-Type header of "text/xml".

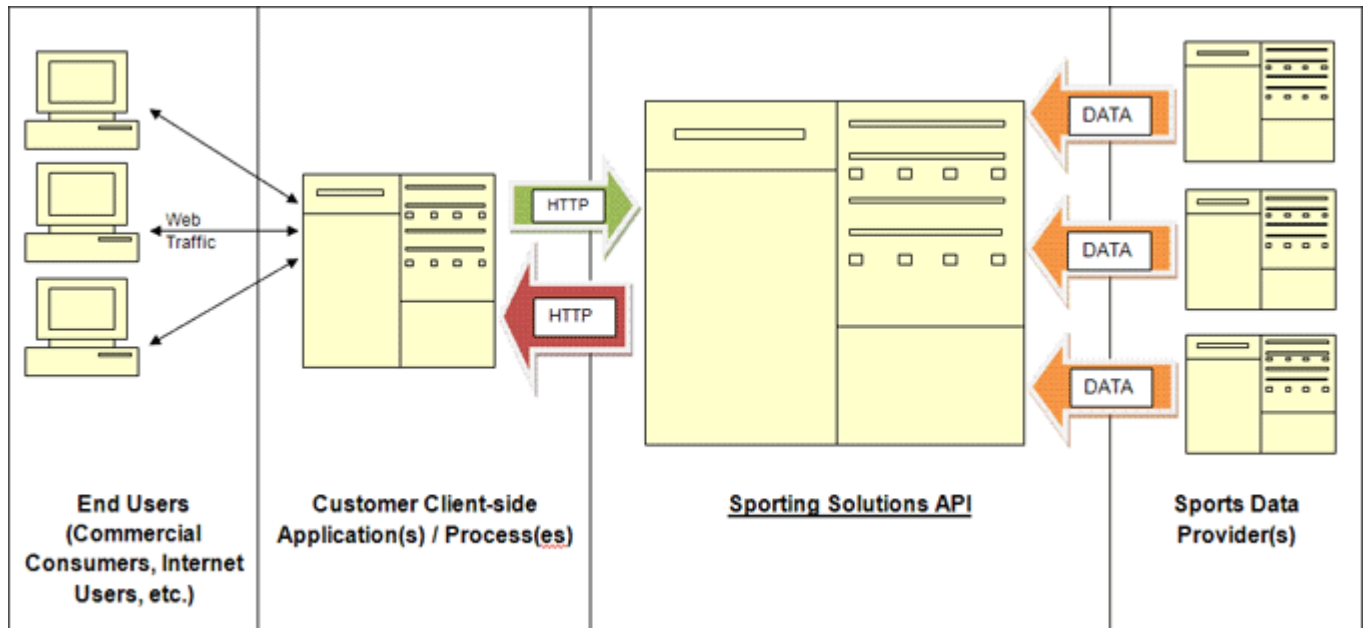
XML Response Formats

The API currently provides responses only in XML format. For examples, please see the section on Available Services (Code Examples).

REST Resources (Concepts)

Implementation Example

This is a diagrammatic representation of a User's interaction with the API:



Resource Overview

The following resources are available via the API:

- Users;
- Session Tokens;
- Channels.

All the REST resources that this document describes are accessible through specific URLs and return XML responses.

Users

A User Resource denotes the account or set of actions of anyone who logs in to the API. User Resources have the following properties:

- Username;
- Password;
- User Profile.

The User Profile contains any information relating to the User's access to API data. This is currently limited to details of the User's Subscription Channel. (Future versions of the API will expand on this User metadata.)

For code examples of how the API interacts with User data, please see the section on User Resources.

Session Tokens

A Session Token is a resource that identifies a Session with the API uniquely:

- performing a login creates a Session and returns a Session Token Resource;
- deleting this resource logs the User out by removing the Session.

Channels

All data that is available via the API is accessible through Channels.

In the context of the API, a Channel is an aggregation of multiple data feeds, typically price updates or match-related metadata such as:

- goals scored;
- wickets taken in cricket;
- yellow cards awarded in football.

The API considers Channels to be resources and can therefore use REST operations to:

- create;
- modify;
- delete them.

The API creates Channels by specifying the data that a User wishes to retrieve through a filter-based mechanism. A User can have multiple Channels in existence at any time. For code examples of how the API interacts with Channel data, please see the section on Channel Resources.

Channel usage example

In this example we assume that a customer can access all the football prices that Sporting Solutions publishes. To manage feeds more easily, the customer sets up multiple Channels, one for each football match or group of matches.

During the course of a match between the teams Arsenal and Chelsea, the customer is interested only in price updates for that match to populate part of their website.

To achieve this, the customer specifies a Channel whose filters include only the Meeting identifier for that match, using the method outlined in the section on Constructing Channel Filters. That Channel then includes price updates only for that match, reducing the amount of both network traffic and data that the customer needs to process.

The same customer can also create another Channel to filter data for the Meeting identifiers of both the "Liverpool vs. Everton" and "Portsmouth vs. Southampton" matches into a single Channel.

For further details of how the API interacts with Channel data, please see the section on Data Management.

Polling or streaming?

After creating a Channel, a customer has two options for retrieving data.

Name	Description	Advantage(s)	Disadvantage(s)
Polling	The User can poll a Channel repeatedly, using a standard HTTP GET request to retrieve the underlying data structure.	This is a simple approach. The customer decides when and how frequently to request data.	The customer has no control of when price updates are published and may receive updates late on occasion. Redundant requests occur when no updates have been made, increasing overall network traffic.
Streaming	The customer can open an HTTP GET streaming connection to a Channel; for further details, please see the section on Streaming Data	The customer receives updates as they occur. Only updates that affect the Channel cause data transfer, keeping network traffic lower.	Client-end software needs to be more complex, needs to be in operation constantly and requires management of Sequence Numbers. Data transfer occurs by 'pushes' controlled by the server: the client application has no control

over incoming traffic.

For an example of how to overcome the difficulties, please see the section that describes a Data Streaming Implementation Example.

For further details of how the API interacts with streamed Channel data, please see the section on Streaming Data.

Data Management

The API provides access to large amounts of rapidly updating data (such as price changes) and match metadata (such as live updates of sports Markets).

To allow Users to access smaller subsets of this data, the API uses Channel Resources to grant Users access only to the data that interests them.

When created, a Channel becomes accessible as a REST resource. Additionally, a customer can open an HTTP stream to a Channel, which then remains open and provided updates periodically. If a client application drops its HTTP connection to a specific Channel Resource (for example, due to a network failure or a crashed system process) it can reconnect to the specific Channel and resume its Session.

For code examples of how the API interacts with Channel data, please see the section on Channel Resources.

Subscriptions and Channels

Every User has a set of data, or Subscription Channel, which the User cannot modify. Any User-created Channel must be a subset of the existing Subscription Channel.

Sporting Solutions sets up a Subscription Channel as part of the User creation process.

Constructing Channel Filters

Channels, including the Subscription Channel, are defined by whatever filtering mechanism is appropriate to the data; the example in the next section is relevant to fixed-odds price data.

Price Data Filters

Price Data is classified in a cascading structure according to Sporting Solutions Address System:

•SportId -> MeetingId -> SubTypeId -> Market ID

(Future versions of the API may change the structure of this data and, therefore, the requisite classification and filtering mechanism.)

Channel identifiers (IDs) are alphanumeric. For details of their current and possible values, please contact your Sporting Solutions account manager or our Customer Services Team.

A Channel filter is constructed in a similar manner to an IP address, with 'wildcard' characters permitted. Examples are:

T123-*	Everything in Sport ID 123
T123-456-*	Everything in Sport ID 123 where the Meeting ID is 456
T123-456-*-	Everything in Sport ID 123 where the Meeting ID is 456 and Market ID is 789
T789	but no SubType is specified

Streaming Data

Summary

One of the key features of the API is the ability for a User to open an HTTP streaming connection and receive updates as they are published.

This is implemented as an HTTP server 'push' using *HTTP Chunked Transfer Encoding* semantics: the API server does not terminate its connection after making its first response to the client application. The API then sends updates over the established HTTP connection. Additionally, the server may specify a 'heartbeat message' to inform the client application that the connection remains open.

For code examples of how the API interacts with streamed Channel data, please see the section on Channel Resources.

Usage

This is an example of a Session that uses streaming.

1. Login - The User logs in using the correct Username and Password.
2. User Profile Request [Optional] - The client application requests the User Profile to enumerate existing Channels.
3. Create or Modify Channel [Optional] - The client application creates a new Channel to filter the data to be streamed, or modifies an existing one.
4. Get Data by Channel [Optional] - The User retrieves data from an existing Channel using a single request (to obtain, for example, a single view of the available data prior to opening a streaming connection).
5. Open Stream to Channel - The User opens an HTTP connection and receives data immediately. The server sends subsequent 'heartbeat messages' (if requested) and updates via the open HTTP connection.
6. Close the Stream - The User can terminate the streaming connection at any time by closing the HTTP connection.

Reconnecting and Sequence Numbers

A streaming request must contain a Sequence Number in its HTTP header. This allows the service to determine at what point in the sequence of updates the data returned will begin.

If the connection is dropped, the client application needs to reopen the stream using the same parameters and the most-recently-received Sequence Number. The API then sends:

- its own current Sequence Number; and
- the subset of data that the server has issued between the client's most-recently-received Sequence Number and the server's current Sequence Number.

'Heartbeat Message'

A client application can specify a preferred 'heartbeat message' frequency in its HTTP headers. The server then sends a 'heartbeat message' at this frequency, regardless of data changes, which consists of a simple XML document containing the most-recently-transmitted Sequence Number.

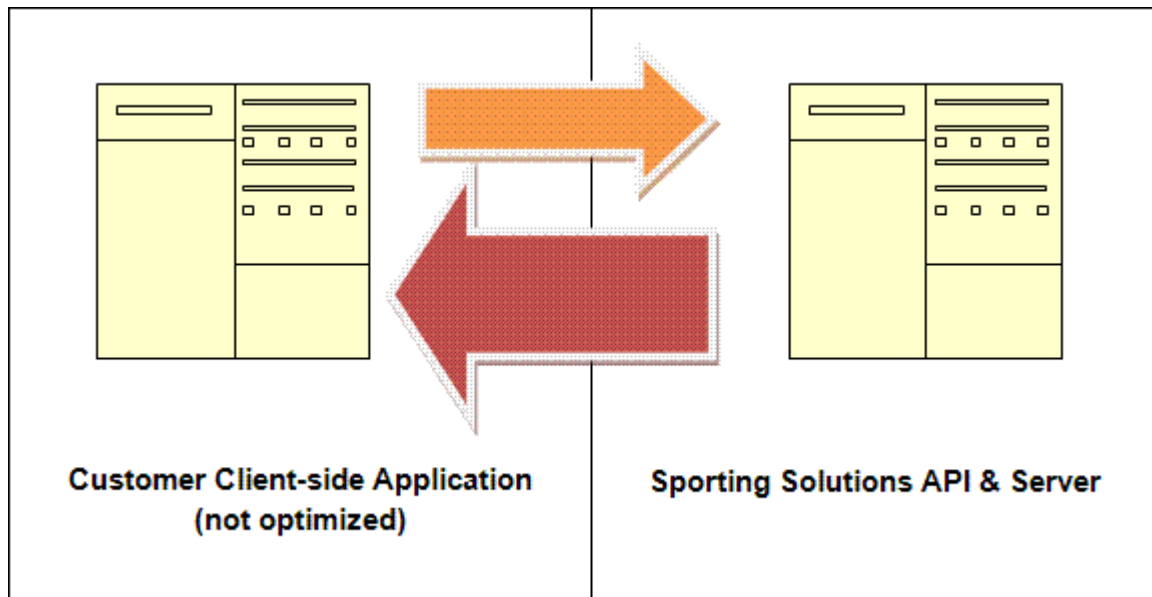
HTTP Chunked Transfer Encoding

The API uses *HTTP Chunked Transfer Encoding* to transmit discrete blocks of data via an open connection, as specified in the transfer-encoding header of the XML response.

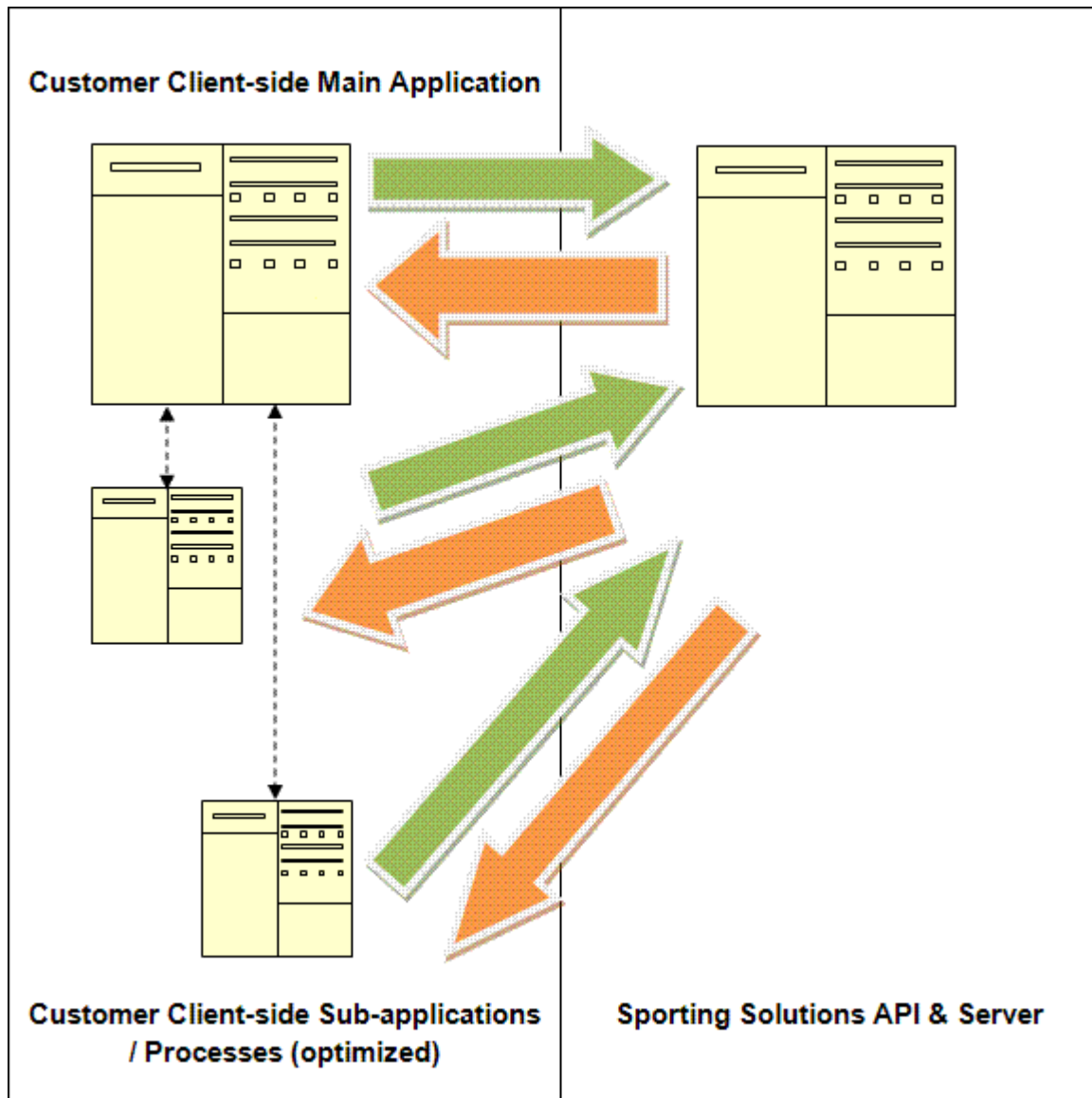
For further details of Chunked Transfer Encoding, please refer to online resources such as the [HTTP/1.1 Protocol Parameters specification](#).

Data Streaming Implementation Example

As discussed in the section on Channels, the streaming of data can overload a client application if its owner does not take steps to manage the large amount of traffic that the service generates.



To avoid this problem, Sporting Solutions recommends its customers distribute their traffic by implementing their applications in a manner similar to the one shown below.



Available Services (Code Examples)

Help Resource

The API provides a basic online summary of this documentation:

Request Type GET
 URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/help/html*
 Required None
 Headers

The response type is in HTML format and this URL is accessible to any browser.

User Resources

The resources that this section describes are for managing User access. They currently allow customers to log in and out of the service and provide User Profile data.

Sporting Solutions provides Usernames and Passwords to access the API according to commercial

agreements with its customers.

For an example of a client application that interacts with this part of the API, please see the User Resources Example.

Log in

Submitting an HTTP PUT request to this resource logs in to the service with a Username and Password.

Request POST

Type

URL <http://fixedodds.api.sportingsolutions.com/PricingApi/SpinSecureWholesaleApi/Login>

Required Headers Content-Type:text/xml

The resource returns:

- Session Token;
- Session Expiry Date;
- URL to the relevant User Profile.

HTTP POST request format

```
<LoginRequest xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<UserName>{UserName}</UserName>
<Password>{YourPassword}</Password>
</LoginRequest>
```

XML response format

```
<?xml version="1.0" encoding="utf-8"?>
<LoginResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Token>{SessionToken}</Token>
<Expiry>{SessionExpiryDate}</Expiry>
<Uri>{UriToAccessUserProfile}</Uri>
</LoginResponse>
```

Notes on the code

Code

```
<Token>{SessionToken}
</Token>
```

```
<Expiry>{SessionExpiryDate}
</Expiry>
```

```
<Uri>{UriToAccessUserProfile}
</Uri>
```

Notes

This part of the XML response contains the User's current Session Token, which the customer's client application should include in subsequent HTTP requests via the API where specified in this document.

This contains the date and time at which the server has set the User's current API Session to expire.

This element contains the Internet address at which the current User Profile is visible.

User Profile

Accessing this resource returns available User Profile information. This currently includes a list of available Channel URLs.

Request Type GET
URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/{UserName}/Token*
Required Headers X-Authentication-Token: {SessionToken}

XML response format

```
<UserProfile xmlns="http://www.sportingindemarket.com/WholesaleApi"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
<Channels>
<ChannelDetails>
<Filters>
<Filter><Name>FILTER1</Name><Value>T338-*</Value></Filter>
</Filters>
<Id>f10cb6cd-6590-4c79-a31c-d14e80f5f914</Id>
<IsReadOnly>>false</IsReadOnly>
<Name>CHANNEL1</Name>
<Uri>http://web9005-
pullapi.oy.gb.sportingindex.com/PricingAPI/SpinSecureWholesaleApi.svc/Chann
el/f10cb6cd-6590-4c79-a31c-d14e80f5f9144
</Uri>
</ChannelDetails>
</Channels>
<UserName>{UserName}</UserName>
</UserProfile>
```

The example above describes one Channel - the User's Subscription Channel. The User cannot modify this; all User-created Channels are a subset of the User's existing Subscription Channel. Notes on the code

Code

```
<UserProfile
xmlns="http://www.sportingindemarket.com/WholesaleA
pi" xmlns:i="http://www.w3.org/2001/XMLSchema-
instance">
```

```
<Channels>
```

```
<ChannelDetails>
```

```
<Filters>
```

Notes

This part of the XML response contains the User Profile as determined by the Username and Session Token submitted in the User's HTTP request.

This XML element precedes a list of the Subscription Channel(s) available to the User. Each user currently has one Subscription Channel.

This element precedes a set of Subscription Channel details.

This element precedes a list of all filters that apply

```
<Filter><Name>FILTER1</Name><Value>T338-
* </Value></Filter>
```

```
</Filters>
```

```
<Id>f10cb6cd-6590-4c79-a31c-d14e80f5f914</Id>
```

```
<IsReadOnly>>false</IsReadOnly>
```

```
<Name>CHANNEL1</Name>
```

```
<Uri>http://web9005-
pullapi.oy.gb.sportingindex.com/PricingAPI/SpinSecu
reWholesaleApi.svc/Channel/f10cb6cd-6590-4c79-a31c-
d14e80f5f9144
</Uri>
```

```
</ChannelDetails>
```

```
</Channels>
```

```
<UserName>{UserName}</UserName>
```

```
</UserProfile>
```

to the User's Subscription Channel. A Channel may have multiple filters.

The XML response includes one instance of this type for each filter that applies to the User's Subscription Channel. In this example, the value "T338-*" indicates that this Subscription Channel returns data for sport T338 (Domestic Soccer- LIVE).

This tag indicates the end of the list of filters for the Subscription Channel.

This element contains the Channel's unique internal identifier.

This element contains a Boolean value ("true" or "false") that indicates whether the Channel is write-protected. This value is "true" for Subscription Channels.

This element contains the Channel's name, as visible to the User.

This element contains an Internet address for the Channel.

These lines close all open tags.

Log out

This request deletes the specified Session and logs the User out.

Request
Type

DELETE

URL

http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/
{UserName}/Token

Required
Headers

X-Authentication-Token: {SessionToken}

Channel Resources

For an example of a client application that interacts with this part of the API, please see the Channel Resources Example.

Create a Channel

This resource creates a Channel that contains Market entities in a price feed, filtered according to the filter string provided in the User's POST data.

The Channel is associated with the User whose Session Token that the HTTP request header has supplied. The XML response returns the relevant Channel identifier and filters.

Request POST

Type

URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel*

Required Content-Type: text/xml

Headers X-Authentication-Token: {SessionToken}

HTTP POST request format

This HTTP request creates a filter that returns all data for the sport whose Sport identifier is "T338":

```
<ChannelCreationRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Name>ChannelName</Name>
<Filters>
<Filter>
<Name>filter1</Name>
<Value>T338-*</Value>
</Filter>
</Filters>
</ChannelCreationRequest>
```

XML response format

```
<?xml version="1.0" encoding="utf-8"?>
<ChannelDetails xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Name>ChannelName</Name>
<Id>ef24e7fd-52d3-40d8-8c37-ee28871baad7</Id>
<IsReadOnly>>false</IsReadOnly>
<Filters>
<Filter><Name>filter1</Name>
<Value>T338-*</Value>
</Filter>
</Filters>
</ChannelDetails>
```

Notes on the code

Code

```
<Name>ChannelName</Name>
```

```
<Id>ef24e7fd-52d3-40d8-8c37-
ee28871baad7</Id>
```

```
<IsReadOnly>>false</IsReadOnly>
```

```
<Filters> <Filter>
```

Notes

This element contains the Channel's name, as determined by and visible to the User.

This element contains the Channel's unique internal identifier, generated by the server.

This element contains a Boolean value ("true" or "false") that indicates whether the Channel is write-protected. This value is "false" for User-created Channels.

This element precedes a list of all filters that apply

<Name>filter1</Name>

<Value>T338-*</Value>

</Filter> </Filters>

</ChannelDetails>

to the User's Subscription Channel. A Channel may have multiple filters.

This element contains the filter's name, as determined by and visible to the User.

This element contains the filter's selection criterion, as determined by and visible to the User.

These lines close all open tags.

Get data by Channel

This resource returns the data that is available to the Session identified by the Session Token provided in the HTTP request header, according to the Channel identifier in the URL and additional optional Boolean filters in the query string.

Request Type GET

URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel/{ChannelId}?Data={x}&Metadata={y}*

Required Headers X-Authentication-Token: {SessionToken}

Query string

Users can perform additional filtering in their query string: if these filters are omitted the default is to include both data and metadata.

Query String Field Value Outcome

Data	1	Returns live data
Data	0	Omits live data
Metadata	1	Returns metadata
Metadata	0	Omits metadata

Examples:

http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel/7e2cf78a-9f78-412c-960c-32e1ee118c69?Data=1&Metadata=0 Returns **only** live data for the channel.

http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel/7e2cf78a-9f78-412c-960c-32e1ee118c69?Data=0&Metadata=1 Returns **only** metadata for the channel.

http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel/7e2cf78a-9f78-412c-960c-32e1ee118c69?Data=0&Metadata=1

XML response format (including metadata)

```
<Channel Channel xmlns="http://www.sportingindex.com/Wholesale">
  <Meta>
    <Id>7e2cf78a-9f78-412c-960c-32e1ee118c69</Id>
    <Name>Channel</Name>
    <Filters>
```

```

<Filter>T338-43904-142366-*</Filter>
</Filters>
</Meta>
<Markets MaxSequence="4428864">
<Market Sequence="4428824">
<Tags>
<SportId>T338</SportId>
<MeetingId>43904</MeetingId>
<SubTypeId>142366</SubTypeId>
</Tags>
<Id>T338-43904-142366-7953403</Id>
<Name>Fifth Goalscorer Player 64</Name>
<AltName></AltName>
<BookPrice></BookPrice>
<DescSoFar>St: No, S: Yes, G: , Y: , R: , On: , Off:</DescSoFar>
<EndOfBettingTime></EndOfBettingTime>
<LastUpdate></LastUpdate>
<Live></Live>
<Price>7</Price>
<Result></Result>
<ShortName></ShortName>
<StartOfBettingTime></StartOfBettingTime>
<StartTime></StartTime>
<Status>0</Status>
<Tradability>1</Tradability>
</Market>
<Market Sequence="4428851">
<Tags>
<SportId>T338</SportId>
<MeetingId>43904</MeetingId>
<SubTypeId>142366</SubTypeId>
</Tags>
<Id>T338-43904-142366-7953430</Id>
<Name>Fifth Goalscorer Player 53</Name>
<AltName></AltName>
<BookPrice></BookPrice> <DescSoFar>St: No, S: Yes, G: , Y: , R: , On: ,
Off:</DescSoFar>
<EndOfBettingTime></EndOfBettingTime>
<LastUpdate></LastUpdate>
<Live></Live>
<Price>14</Price>
<Result></Result>
<ShortName></ShortName>
<StartOfBettingTime></StartOfBettingTime>
<StartTime></StartTime>
<Status>0</Status>
<Tradability>1</Tradability>
</Market>
</Markets>
</Channel>

```

Notes on the code

Code

```

<Meta>
<Id>7e2cf78a-9f78-412c-960c-
32e1ee118c69</Id>
<Name> Channel </Name>
<Filters>
<Filter> T338-43904-142366-

```

Notes

If the HTTP request included a Metadata value of "1", the XML response includes these elements which return details of the Channel's internal identifier, name and filter(s).

*</Filter>	
</Filters>	
</Meta>	
<Markets MaxSequence="4428864">	This tag precedes a list of all Markets that the Channel covers.
<Market Sequence="4428824">	This element specifies the Sequence Number of the current Market. (Every time the Market is updated, the Sequence Number increases.)
<Tags>	These elements specify the Sport, Meeting and SubType to which the current Market belongs.
<SportId>T338</SportId>	A list of all the Market's attributes follows.
<MeetingId>43904</MeetingId>	
<SubTypeId>142366</SubTypeId>	
</Tags>...	
... </Market>	These lines close all open tags.
</Markets>	
</Channel>	

Open a streaming connection to a Channel

This resource returns all the data that is available to the Session identified by the Session Token provided in the HTTP request header, according to the Channel identifier in the URL.

The HTTP connection remains open and the service streams data updates in real time.

If the User has specified a 'heartbeat message', the resource sends this message at a specified frequency.

For an example of a client application that interacts with this part of the API, please see the Streaming Channel Resources Example.

Request GET

Type

URL *http://fixedodds.api.sportingsolutions.com/PricingApi/streaming.ashx?channel={ChannelId}*

Require X-Authentication-Token: {SessionToken}

d X-Sequence: {SequenceNumber}

X-Heartbeat-Frequency: {Heartbeat-Frequency}

Headers

Notes:

- **Sequence Number** refers to the Sequence Number of the first price change you wish to receive. Sequence Numbers reside in the XML response.

- **Heartbeat Frequency** specifies the frequency (in milliseconds) at which you wish to receive 'heartbeat messages'. (If you set this to zero the API sends no 'heartbeat messages' at all.)

XML response format

```
<Channel xmlns="http://www.sportingsolutions.com/Wholesale">
  <Meta>
    <Id>7e2cf78a-9f78-412c-960c-32e1ee118c69</Id>
    <Name>Channel</Name>
    <Filters>
      <Filter>T338-43904-142366-*</Filter>
    </Filters>
  </Meta>
  <Markets MaxSequence="4428864">
    <Market Sequence="4428824">
      <Tags>
        <SportId>T338</SportId>
```

```

<MeetingId>43904</MeetingId>
<SubTypeId>142366</SubTypeId>
</Tags> <Id>T338-43904-142366-7953403</Id>
<Name>Fifth Goalscorer Player 64</Name>
<AltName></AltName>
<BookPrice></BookPrice>
<DescSoFar>St: No, S: Yes, G: , Y: , R: , On: , Off:</DescSoFar>
<EndOfBettingTime></EndOfBettingTime>
<LastUpdate></LastUpdate>
<Live></Live>
<Price>7</Price>
<Result></Result>
<ShortName></ShortName>
<StartOfBettingTime></StartOfBettingTime>
<StartTime></StartTime>
<Status>0</Status>
<Tradability>1</Tradability>
</Market>
<Market Sequence="4428851">
<Tags>
<SportId>T338</SportId>
<MeetingId>43904</MeetingId>
<SubTypeId>142366</SubTypeId>
</Tags>
<Id>T338-43904-142366-7953430</Id>
<Name>Fifth Goalscorer Player 53</Name>
<AltName></AltName>
<BookPrice></BookPrice>
<DescSoFar>St: No, S: Yes, G: , Y: , R: , On: , Off:</DescSoFar>
<EndOfBettingTime></EndOfBettingTime>
<LastUpdate></LastUpdate>
<Live></Live>
<Price>14</Price>
<Result></Result>
<ShortName></ShortName>
<StartOfBettingTime></StartOfBettingTime>
<StartTime></StartTime>
<Status>0</Status>
<Tradability>1</Tradability>
</Market>
</Markets>
</Channel>

```

'Heartbeat Message' XML response format

If the User has specified a 'heartbeat message' in an HTTP request, the XML response includes an extra line:

```
<heartbeat sequence='4428864'></heartbeat>
```

Modify a Channel

A PUT request to an existing User-created Channel Resource modifies it:

Request Type PUT

URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel*

Required Content-Type: text/xml

Headers X-Authentication-Token: {SessionToken}

HTTP PUT request format

This example modifies an existing Channel to include all data for the Sport whose Sport ID is "T338" and, additionally, data for one Market within another specific sports Meeting:

```

<ChannelDetails xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Name>ChannelName</Name>
<Id>ce756fa6-d9d3-42ef-867b-d89a333399f5</Id>
<IsReadOnly>>false</IsReadOnly>
<Filters>
<Filter>
<Name>filter1</Name>
<Value>T338-*</Value>
</Filter>
<Filter>
<Name>filter2</Name>
<Value>T418-43906-109175-7954964</Value>
</Filter>
</Filters>
</ChannelDetails>

```

The service responds to the request with an HTTP status code of "200".

Delete a Channel

Making an HTTP DELETE request to a specified User-created Channel Resource deletes it.

Request Type DELETE

URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Channel/{ChannelId}*

Required X-Authentication-Token: {SessionToken}

Headers

The service responds to the request with an HTTP status code of "200".

Fixtures Resource

This resource returns a list of current and upcoming sports Meetings; the list provides the Sport and Meeting identifiers required to create Channels.

This resource returns responses in *SportsML* format; for details, please see the [International Press Telecommunications Council \(IPTC\)](#) website (external link).

Requesting fixtures

Request Type GET

URL *http://fixedodds.api.sportingsolutions.com/PricingAPI/SpinSecureWholesaleapi/Fixtures/{SportCode}*

Required X-Authentication-Token: {SessionToken}

Headers

Notes:

- Sport IDs** reside in the code-key attribute of the sports-content-code element.
- The **Meeting IDs** reside in the event-key attribute of the event-metadata element.

Fixtures XML (*SportsML*) response format

This is an example of the *SportsML* format document that the Fixtures Resource returns.

```

<SportsMLEnvelopes MaxSequence="0"
xmlns="http://www.sportingindex.com/Wholesale">
  <Envelope Sequence="0">
    <Tags />

```

```

<Id />
<SportsContent>
  <sports-content xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://iptc.org/std/SportsML/2009-06-18/">
    <sports-metadata doc-id="b10c7884-5151-4e07-adf2-
79bb973c1766" publisher="Sporting Solutions" date-time="2010-09-
10T10:20:50.0434369+01:00">
      <sports-title />
      <sports-content-codes>
        <sports-content-code code-type="sport" code-
key="T338" code-source="iptc" code-name="" />
      </sports-content-codes>
    </sports-metadata>
    <sports-event>
      <event-metadata event-key="77457" event-
name="Newcastle v Blackpool (15.00, Saturday)" start-date-time="2010-09-
11T15:00:00+01:00" />
      <player>
        <player-metadata />
      </player>
    </sports-event>
    <sports-event>
      <event-metadata event-key="77453" event-
name="Birmingham v Liverpool (16.00, Sunday)" start-date-time="2010-09-
12T16:00:00+01:00" />
      <player>
        <player-metadata />
      </player>
    </sports-event>
  </sports-content>
</SportsContent>
</Envelope>
</SportsMLEnvelopes>

```

Notes on the code

<i>Code</i>	<i>Notes</i>
<pre> <sports-content xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance" xmlns="http://iptc.org/std/SportsML/2009-06- 18/"> <sports-metadata doc- id="b10c7884-5151-4e07-adf2-79bb973c1766" publisher="Sporting Solutions" date- time="2010-09-10T10:20:50.0434369+01:00"> <sports-title /> <sports-content- codes> <sports-content-code code-type="sport" code-key="T338" code- source="iptc" code-name="" /> </sports-content- codes> </sports-metadata> </pre>	<p>This section includes details of the relevant Sport's name, publisher, Sport ID and date and time broadcast.</p>
<pre> <sports-event> <event-metadata event-key="77457" event-name="Newcastle v Blackpool (15.00, Saturday)" start-date- time="2010-09-11T15:00:00+01:00" /> </pre>	<p>This section includes details of the relevant Meeting's name and Meeting ID.</p>

<pre><player> <player- metadata /> </player> </sports-event> </sports-content></pre>	<p>This section is empty in PricingAPI. In MetadataAPI it includes the identifier and full name of each player or participant in the relevant Meeting.</p>
	<p>These lines close all open tags.</p>

Status and Error Codes

Every HTTP request submitted to the server via the API generates a response.

The API returns standard HTTP status codes and XML error codes. Examples are:

Code Interpretation

200 Request successful
201 Item created
400 Bad request
500 Internal server error

For full details of these and other codes, please see any search engine or other online resource.

Implementation Examples (.NET / C#)

Introduction

This section of the document contains examples of a client implementation of the API using .NET technology and C# code.

- User Resources;
- Channel Resources; and
- Streaming Channel Resources.

You may base your own implementations on these examples, but please note that:

- these examples represent only a partial implementation of the API and do not incorporate all of its features; and
- Sporting Solutions does not offer support or advice for generic coding.

User Resources Example

This (.NET / C#) implementation complements the HTTP code documented in the section on User Resources.

```
using System;
using System.IO;
using System.Net;
using System.Text;
using System.Xml;

namespace SportingSolutionsExample
{
    /// <summary>
    /// Provides access to API User Resources
    /// (example code only, not for production use).
    /// </summary>
    public class UserResourceExample
    {
        private readonly string _sportingSolutionsUrl;
        private const string LoginResource = @"/Login";
```

```

private const string TokenResource = @"/{0}/Token";
private const string ContentType = "text/xml";
private const string AuthenticationTokenHeader = "X-Authentication-Token";
private const string LoginPostData =
@"<LoginRequest><UserName>{0}</UserName>\<Password>{1}</Password></LoginReq
uest>";
/// <summary>
/// Creates a new UserResourceExample using the URL provided:
/// </summary>
/// <param name="url">Obtain URL from
{account_manager}@sportingsolutions.com</param>
public UserResourceExample(string url)
{
    _sportingSolutionsUrl = url;
}
/// <summary>
/// Logs in to the API and returns a Session ID:
/// </summary>
/// <param name="username">Obtain username from
{account_manager}@sportingsolutions.com</param>
/// <param name="password">Obtain password from
{account_manager}@sportingsolutions.com</param>
/// <returns></returns>
public Guid Login(string username, string password)
{
    string loginUrl = _sportingSolutionsUrl + LoginResource;

    // Set up the HTTP request:
    HttpRequest request = (HttpRequest)WebRequest.Create(loginUrl);
    string postData = string.Format(LoginPostData, username, password);
    byte[] postBytes = Encoding.ASCII.GetBytes(postData);
    request.Method = "POST";
    request.ContentType = ContentType;
    request.ContentLength = postBytes.Length;
    Stream requestStream = request.GetRequestStream();
    requestStream.Write(postBytes, 0, postBytes.Length);
    requestStream.Close();
    // Get the HTTP response and process the Session Token from it:
    HttpResponse response = (HttpResponse)request.GetResponse();
    XmlDocument xmlDocument = new XmlDocument();
    xmlDocument.Load(response.GetResponseStream());
    XmlNodeList nodes = xmlDocument.GetElementsByTagName("Token");
    string token = nodes[0].InnerText;
    return new Guid(token);
}
/// <summary>
/// Logs out of the API:
/// </summary>
/// <param name="username">Obtain username from
{account_manager}@sportingsolutions.com</param>
/// <param name="sessionToken">Obtain session token from Login
Method</param>
public void Logout(string username, Guid sessionToken)
{
    string tokenUrl = _sportingSolutionsUrl +
string.Format(TokenResource, username);
    HttpRequest request = (HttpRequest)WebRequest.Create(tokenUrl);
    request.Method = "DELETE";
    request.Headers.Add(AuthenticationTokenHeader, sessionToken.ToString());
    request.GetResponse();
}

```

```

/// <summary>
/// Gets the User Profile for the specified User (requires a valid session
token):
/// </summary>
/// <param name="username">Obtain username from
{account_manager}@sportingsolutions.com</param>
/// <param name="sessionToken">Obtain session token from Login
Method</param>
/// <returns></returns>
public XmlDocument GetUserProfile(string username, Guid sessionToken)
{
string tokenUrl = _sportingSolutionsUrl + string.Format(TokenResource,
username);
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(tokenUrl);
request.Method = "GET";
request.Headers.Add("X-Authentication-Token", sessionToken.ToString());
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.Load(response.GetResponseStream());
return xmlDocument;
}
}
}

```

Channel Resources Example

This (.NET / C#) implementation complements the HTTP code documented in the section on Channel Resources.

```

using System;
using System.IO;
using System.Net;
using System.Text;
using System.Xml;

namespace SportingSolutionsExample
{
/// <summary>
/// Provides access to API Channel Resources
/// (example code only, not for production use).
/// </summary>
public class ChannelResourceExample
{
private readonly string _sportingSolutionsUrl;
private const string ChannelResource = @"/Channel";
private const string ChannelDataResource = @"/Channel/{0}";
private const string ContentType = "text/xml";
private const string AuthenticationTokenHeader = "X-Authentication-Token";
private const string ChannelCreationPostData =
@"<ChannelCreationRequest><Name>{0}</Name><Filters><Filter><Name>{1}
</Name><Value>{2}</Value></Filter></Filters></ChannelCreationRequest>";
/// <summary>
/// Creates a new ChannelResourceExample using the URL provided:
/// </summary>
/// <param name="url">Obtain URL from
{account_manager}@sportingsolutions.com</param>
public ChannelResourceExample(string url)
{
_sportingSolutionsUrl = url;
}
/// <summary>
/// Creates a Channel with a single filter only. May be enhanced to add

```

```

multiple filters.
/// Filters must be valid for the User's subscription.
/// </summary>
/// <param name="sessionToken">Session token obtained from a Login via the
User Resource</param>
/// <param name="channelName">Required Name for the Channel
<example>SoccerChannel</example></param>
/// <param name="filterName">Required Name for a single Filter
<example>DomesticSoccer</example></param>
/// <param name="filterValue">Filter value <example>Domestic Soccer: "T338-
*"</example></param>
/// <returns>Channel Creation Response XML</returns>
public XmlDocument CreateChannel(Guid sessionToken, string channelName,
string filterName, string filterValue)
{
string channelUrl = _sportingSolutionsUrl + string.Format(ChannelResource);
HttpRequest request = (HttpRequest)WebRequest.Create(channelUrl);
string postData = string.Format(ChannelCreationPostData, channelName,
filterName, filterValue);
byte[] postBytes = Encoding.ASCII.GetBytes(postData);
request.Method = "POST";
request.ContentType = ContentType;
request.ContentLength = postBytes.Length;
request.Headers.Add(AuthenticationTokenHeader, sessionToken.ToString());
Stream requestStream = request.GetRequestStream();
requestStream.Write(postBytes, 0, postBytes.Length);
requestStream.Close();
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(response.GetResponseStream());
return xmlDoc;
}
/// <summary>
/// Returns the data for a specified Channel:
/// </summary>
/// <param name="sessionToken">Session token obtained from a Login via the
User Resource</param>
/// <param name="channelId">Channel ID obtained via a previous channel
creation. May be read from the user profile</param>
/// <returns>Channel Data XML</returns>
public XmlDocument GetDataForChannel(Guid sessionToken, Guid channelId)
{
string channelUrl = _sportingSolutionsUrl +
string.Format(ChannelDataResource, channelId);
HttpRequest request = (HttpRequest)WebRequest.Create(channelUrl);
request.Method = "GET";
request.Headers.Add(AuthenticationTokenHeader, sessionToken.ToString());
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(response.GetResponseStream());
return xmlDoc;
}
}
}
}

```

Streaming Channel Resources Example

This (.NET / C#) implementation complements the HTTP code documented in the section on Channel Resources.

```

using System;
using System.Diagnostics;

```

```

using System.IO;
using System.Net;
using System.Text;

namespace SportingSolutionsExample
{
    /// <summary>
    /// Provides access to API Streaming Channel Resources
    /// (example code only, not for production use).
    /// </summary>
    public class StreamingExample
    {
        private readonly string _sportingSolutionsUrl;
        private const string AuthenticationTokenHeader = "X-Authentication-Token";
        private const string SequenceHeader = "X-Sequence";
        private const string HeartbeatHeader = "X-Heartbeat-Frequency";
        private const string ChannelResource = "?channel={0}";
        public StreamingExample(string url)
        {
            _sportingSolutionsUrl = url;
        }
        /// <summary>
        /// Opens a Streaming data connection for a specific channel:
        /// </summary>
        /// <param name="sessionToken">Session token obtained from a Login via the
        User Resource</param>
        /// <param name="channelId">Channel identifier obtained via a previous
        channel creation. May be read from the user profile</param>
        public void StreamChannelData(Guid sessionToken, Guid channelId)
        {
            string channelUrl = _sportingSolutionsUrl + string.Format(ChannelResource,
            channelId);
            HttpRequest request = (HttpRequest)WebRequest.Create(channelUrl);
            request.Method = "GET";
            request.Headers.Add(AuthenticationTokenHeader, sessionToken.ToString());
            request.Headers.Add(SequenceHeader, "0");
            request.Headers.Add(HeartbeatHeader, "1000"); //One second heartbeat
            frequency
            RequestState requestState = new RequestState { Request = request };

            // Open the response, providing a callback delegate:
            request.BeginGetResponse(AsynchronousResponseCallback, requestState);
        }
        /// <summary>
        /// Handler for the streaming callback:
        /// </summary>
        /// <param name="asynchronousResult"></param>
        private static void AsynchronousResponseCallback(IAsyncResult
        asynchronousResult)
        {
            // State of request is asynchronous:
            var requestState = (RequestState)asynchronousResult.AsyncState;
            HttpRequest httpRequest = requestState.Request;
            requestState.Response =
            (HttpWebResponse)httpRequest.EndGetResponse(asynchronousResult);

            // Read the response into a Stream object:
            Stream responseStream = requestState.Response.GetResponseStream();
            requestState.StreamResponse = responseStream;

            // Begin reading from the stream, providing a callback method:

```

```

responseStream.BeginRead(requestState.Buffer, 0,
RequestState.BufferSize,
AsynchronousStreamReadCallback, requestState);
}
/// <summary>
/// Handler for the streaming reader:
/// </summary>
/// <param name="asynchronousResult"></param>
private static void AsynchronousStreamReadCallback(IAAsyncResult
asynchronousResult)
{
var requestState = (RequestState)asynchronousResult.AsyncState;
Stream responseStream = requestState.StreamResponse;
int read = responseStream.EndRead(asynchronousResult);
if (read > 0 )
{
string chunk = Encoding.UTF8.GetString(requestState.Buffer);
Trace.WriteLine(chunk); //Simply tracing out here. Production code will
process data here
requestState.Buffer = new byte[RequestState.BufferSize];
responseStream.BeginRead(requestState.Buffer, 0, RequestState.BufferSize,
AsynchronousStreamReadCallback, requestState);
}
else
{
responseStream.Close();
}
}
}
class RequestState
{
public const int BufferSize = 32768;
public byte[] Buffer { get; set; }
public HttpRequest Request { get; set; }
public HttpResponse Response { get; set; }
public Stream StreamResponse { get; set; }
public RequestState()
{
Buffer = new byte[BufferSize];
Request = null;
StreamResponse = null;
}
}
}

```

Further Help

Frequently-asked Questions (FAQs)

How do I...?

- [Log in and out](#)
- [See my User details](#)
- [View and manage my Channels](#)
- [Stream my data](#)
- [Diagnose errors and other problems](#)
- [Find help](#)

How do filters handle AND/OR/NOT logical combinations?

"Cricket" and another on the code for "Motor Racing - Japanese Grand Prix" results in a User-defined Channel that returns a combination of all cricket data and all data for the specified motor race.

Can I filter Channel data based on string values and keywords?

No. You need to obtain the relevant Sport and Meeting identifier codes from the [Fixtures Resource](#) and

select the subsets of SubType and Market identifiers from the initial result set.

How do I obtain identifiers for filtering my Channels?

Please see the section on the [Fixtures Resource](#).

Can I create or delete User accounts?

No. Only Sporting Solutions can create or delete User accounts. If you wish to change your list of Users, please contact your Sporting Solutions account manager.

Can I change User permissions and Subscription Channel settings?

No. Only Sporting Solutions can create or delete User accounts. If you wish to change your list of Users, please contact your Sporting Solutions account manager.

How do I obtain or view a User Session Token?

When you log in, the server includes your current Session Token in its XML response. For an example, please see the section on [User Resources](#).

Can I specify my own User-created Channel as my default Channel?

No. If you wish to change the data you receive via your Subscription Channel, please contact your Sporting Solutions account manager.

How can I edit Channels?

To edit and manage your User-defined Channels, please see the section on [Data Management](#).

Can Users share each other's User-created Channels?

The API does not currently allow separate Users to share a common User-defined Channel; if required, have each User create a separate Channel using identical filters.

Is the API available in other formats such as JSON?

The API accepts HTTP requests in JSON format if the HTTP Accept:header includes the value "application/json".

Why do I see unexpected values when I try to stream data?

If you try to stream data using an incorrect SportID or other identifier, you may see unexpected, corrupted or 'jumbled' results, or even an HTTP error such as "401".

Please ensure that you are using the correct identifiers for the Sport(s), Meeting(s) and Market(s) you wish to stream; for further details, please see the section on the [Fixtures Resource](#). If you have any further questions or problems, contact our Customer Services Team.

Support Information

If you require further help with your API service implementation, additional services or general account enquiries, please contact our Customer Services Team (available 24 hours a day, 7 days a week) by emailing sportingsolutionshelpdesk@sportingindex.com or calling 08000 96 96 05 (+44 207 840 4001 International).