bwin Feed Documentation

API Description, Integration, Configuration
Version 4

Entain
For the good of entertainment

Entain

bwin Feed Documentation

API Description, Integration, Configuration

Published in Vienna by Entain Services Austria GmbH (“bwin”).

Preface

Welcome to the bwin Feed Guide. This document describes the *bwin Feed*, the bet content delivery service for our sports betting partners. You will learn how the bwin Feed works, how to integrate it into your application landscape, and how to configure it properly to your needs.

The target audience for this document are Entain sports betting data integration partners. Consequently, this guide does not explain general sports betting terms and procedures. Furthermore, we assume that you have the necessary knowledge to work with REST APIs.

If you have questions above and beyond the information provided here, do not hesitate to contact the Entain Product Management Team (d.All.PM.Sports.Backend@entaingroup.com).

Vienna, November 2021
The bwin Feed Team

Contents

1 Introduction

The bwin Feed provides sports betting content via an API. The available content includes event snapshots, translations, and the possibility to retrieve older data that may be needed for recovery use cases. The bwin Feed provides:

- Fixtures (“events”)
- Odds
- Event information
- Resulting
- Event calendar
- Translations in different languages

> **TIP:** See Appendix B for a glossary of important terms and definitions.

Depending on your requirements, the sports betting content retrieved can easily be customized to your needs.

> **NOTICE:** The base language of the betting content is English.

1.1 Overview

The bwin Feed collects the information coming in from the Entain trading system, aggregates it, and then sends this information as a snapshot of each fixture to the partners.

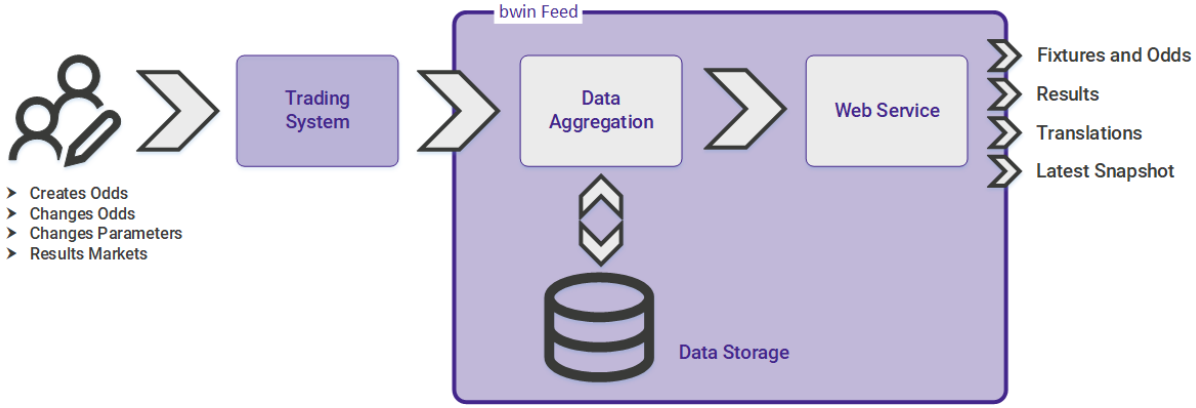


Fig. 1: bwin Feed system overview

2 Technical set-up

The bwin Feed is a Web API that has been implemented using HTTP and REST[1] principles. This allows for simple implementation while being platform-independent, because most programming languages provide built-in functionality to perform HTTP requests.

The currently supported HTTP version is HTTP/1.1.

2.1 REST principles and the bwin Feed

The REST architectural style emphasizes principles that apply for Web services such as the bwin Feed. These points are the most important ones for the bwin Feed:

- **Resources:** The key abstraction within REST is the resource. The API exposes all data elements that can be identified via URLs as resources.
- **Statelessness:** A server does not maintain any partner state across HTTP requests. State is always provided via single requests. Hence, the bwin Feed does not distinguish between initial and subsequent calls. However, global data may persist across calls.
- **Idempotency:** HTTP GET requests to the same resource shall always return the same data. However, data within the gaming domain quickly gets stale due to continuous updates of the betting content.

[1] See https://restfulapi.net/ for details.

2.2 Response format

The bwin Feed supports different media types for the response:

- **XML** (application/xml):
  The data is formatted as XML. The corresponding XSD for the deserialization into objects is provided by us. Be aware that we distribute a distinct XSD for every bwin Feed version. The XML format is the best human-readable format, but with the biggest data size.
- **JSON** (application/json):
  The data is formatted as JSON. We also distribute a documentation of the JSON structure with every new bwin Feed version.
  JSON data size is smaller than XML, but not as well-readable.

You must specify the response media type you want to get as `Accept` in the header of your HTTP request (HTTP 1.1 standard). You can define multiple media types as comma-separated values:

```
GET bwinFeedUrl
HTTP/1.1
Accept: application/xml,application/json

HTTP/1.1 200 OK
Content-Type: application/json
```

The `Content-Type` in the HTTP response header defines the actual format of the response. If you have defined multiple media types in the request, the bwin Feed selects the most performant one.

2.3 Error codes

The bwin Feed uses standard HTTP status codes when returning errors:

```
HTTP/1.1 StatusCode StatusDescription
Content-Type: ContentType
```

Any status code different to `200` indicates an error. In addition to the HTTP status code, the response header also contains the status description, for instance `OK` for status code `200`:

```
HTTP/1.1 200 OK
Content-Type: application/xml
```

For most errors, the API also provides the available additional error information in the HTTP response body:

```
HTTP/1.1 StatusCode StatusDescription
Content-Type: application/xml

<string>Event ID not found</string>
<string>Event ID is invalid</string>
…
```

2.4 Error handling

An error message containing a string message with additional details is returned after sending a request that failed to meet a business rule. Typical examples are sending event requests with invalid or non-existing Event IDs or sending a request with an improper Access ID.

- When the bwin Feed returns such an error, re-check your request. Do not retry the service call with the same arguments. Try different arguments.
- In other scenarios with errors, where the type of error cannot be inferred from the response code and error message, retry the service call for limited number of times with the same arguments, but with delays between the calls.

2.5 Access

2.5.1 Available URLs

The bwin Feed is available in two environments via the following URLs:

- Integration: http://v4.customerintegration.bwinfeed.com
- Production: http://v4.bwinfeed.com

For your requests, combine one of these URLs with a resource URL (see section 3).

Integration environment

Entain maintains the Integration environment for customer integration and testing purposes. Although only for testing, the environment provides up-to-date production content.

Connect to this environment during the implementation, integration, and testing phases.

Production environment

Switch to this environment when going live after the testing phase for revenue-generating sports betting activities.

2.5.2 Interfaces

The bwin Feed provides fixtures for pre-match betting (before the event starts) and fixtures for in-play (“live”) betting via dedicated endpoints. Furthermore, a unified (common) endpoint is available that can deliver both types of fixtures.

Live

```
GET /api/Event/getLiveEvent/[…]
```

Pre-match

```
GET /api/Event/getPreMatchEvent/[…]
```

Unified/common

```
GET /api/Event/getEvent/[…]
```

2.5.3 Authorization

All API calls require authorization. bwin Feed authorization is based on the following factors:

- **Access identifier:** You have got an access identifier (“Customer ID”) from your Entain partner representative.
- **Whitelisted IP address:** You have sent us the IP address we shall whitelist to enable you to connect.

  > **HINT:** If this has not happened yet, contact your Entain partner representative.

Include the access identifier in the request URL like in this example:

```
GET /api/Index/liveindex?customerId=customerId
HTTP/1.1
Host: v4.bwinfeed.com/
Accept: application/xml
```

2.6 Push technology

The bwin Feed API uses *HTTP Long Polling* as its main push technology.

> **NOTICE:** HTTP Long Polling is not a real push technology. It is a variant of the traditional polling technique that allows emulating a push mechanism.

You request information from the server similar to normal polling. The main difference comes into play when the server does not have any information available:

- With normal polling, the server would send an empty response.
- With long polling, the server holds the request for up to 15 seconds. If the requested information becomes available, it is returned. After 15 seconds without the requested information, the latest available information is returned.

The following sequence diagram displays how long polling works in the bwin Feed:

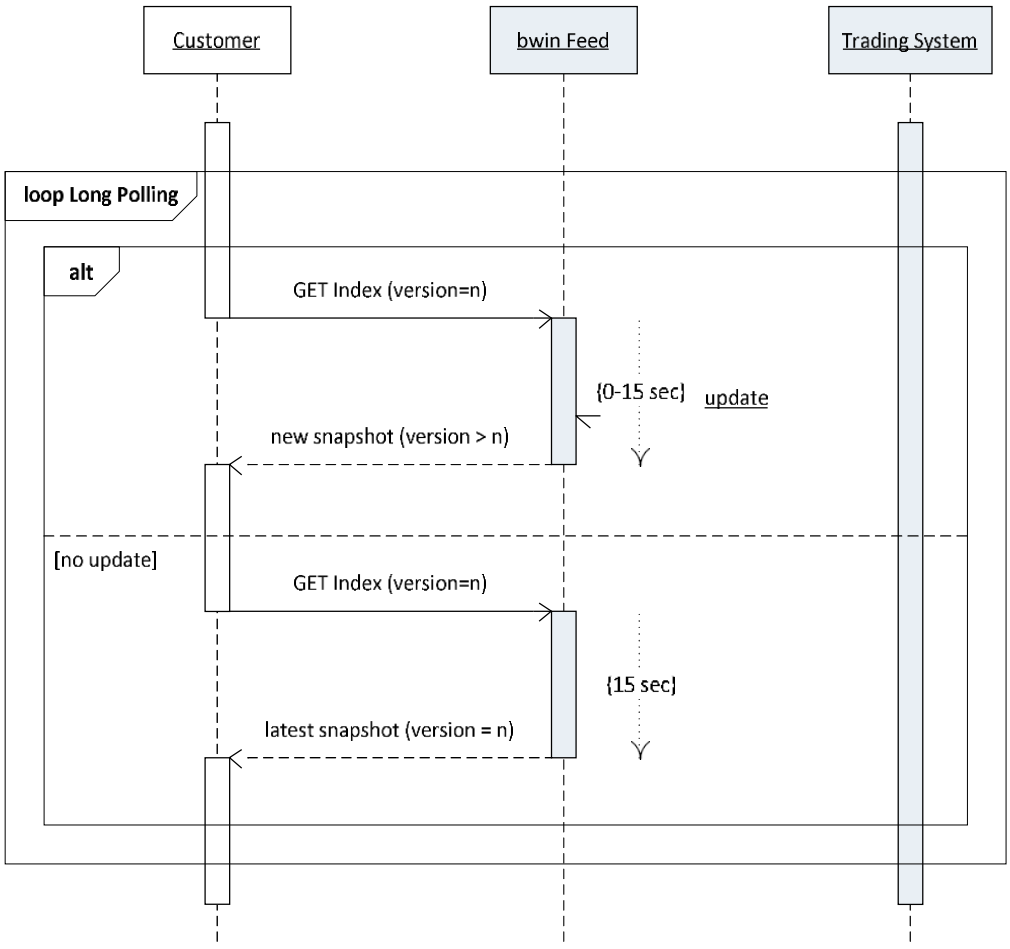


Fig. 2: HTTP Long Polling in the bwin Feed

3 Getting started

This part of the documentation tells you the necessary steps to get started with connecting to and requesting data from the live interface (see section 3.1) and from the pre-match interface (see section 3.2).

> **NOTICE:** The following sections shall give you an overview. The technical details are explained in subsequent parts. Refer to section 10 for concrete implementation strategies and their advantages and pitfalls.

3.1 Getting started with the live interface

Retrieving data from the live interface of the bwin Feed includes the following steps:

1. Long-poll the index operation (see section 3.1.1).
2. Long-poll the bet content for selected fixtures (see section 3.1.2).
3. Process the bet content (see section 3.1.3).

3.1.1 Long-poll the index operation on the live interface

The first step in connecting to the bwin Feed is to long-poll the **`GET Live Index`** operation. This returns a list of all active in-play fixtures, including their identifier and version number.

Request

```
GET /api/Index/liveindex?customerid=12345678-1234-123456789ABCD&version=0
HTTP/1.1
Host: v4.customerintegration.bwinfeed.com
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<CustomVersionIndex>
   <version>0</version>
   <events>
      <id>0</id>
      <version>0</version>
      <eventDateUtc>2021-10-13T09:36:24.689Z</eventDateUtc>
      <eventCutOffDateUtc>2021-10-13T09:36:24.689Z</eventCutOffDateUtc>
```

```
      <willBeOfferedLive>true</willBeOfferedLive>
      <sportId>0</sportId>
      <tradingVersion>1</tradingVersion>
      <internalId>0</internalId>
   </events>
</CustomVersionIndex>
```

3.1.2 Long-poll bet content via the live interface

The second step is to request the bet content for selected in-play fixtures by long-polling the **`GET Live Event`** operation. You receive a snapshot of each requested fixture.

Each fixture snapshot always contains the complete information for the fixture, including all traded markets, as well as the entire scoreboard. There are no delta update snapshot messages that only contain parts of the fixture information. The advantage is that losing one or a small number of snapshot messages does not significantly impact content usage models.

Request

```
GET api/Event/getLiveEvent?customerId=12345678-1234-123456789ABCD&id=12345&version=1
HTTP/1.1
Host: v4.customerintegration.bwinfeed.com
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Event Version=”1”Id=”12345”Result=””>
  <Info ContractVersion=”v4_0” TimestampUTC=”2021-10-30T09:33:14.4392546Z”>
  […]
</Event>
```

3.1.3 Process the live bet content

For every response you get, you need to process the information received according to the data definition (in an XSD or another source). Extract the information relevant for you.

3.2 Getting started with the pre-match interface

Retrieving data from the pre-match interface of the bwin Feed includes the following steps:

1. Long-poll the index operation (see section 3.2.1).
2. Request the bet content for selected fixtures (see section 3.2.2).
3. Process the bet content (see section 3.2.3).

3.2.1 Long-poll the index operation on the pre-match interface

The first step in connecting to the bwin Feed is to long-poll the **`GET Prematch Index`** operation. This returns a list of all active pre-match fixtures, including their identifier.

Request

```
GET /api/Index/prematchindex?customerid=12345678-1234-123456789ABCD&version=0
HTTP/1.1
Host: v4.customerintegration.bwinfeed.com
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<CustomVersionIndex>
   <version>0</version>
   <events>
      <id>0</id>
      <version>0</version>
      <eventDateUtc>2021-10-13T11:32:21.418Z</eventDateUtc>
      <eventCutOffDateUtc>2021-10-13T11:32:21.418Z</eventCutOffDateUtc>
      <willBeOfferedLive>true</willBeOfferedLive>
      <sportId>0</sportId>
      <tradingVersion>1</tradingVersion>
      <internalId>0</internalId>
   </events>
</CustomVersionIndex>
```

3.2.2 Request bet content via the pre-match interface

The second step is to request the bet content for selected fixtures by calling the **`GET PreMatch Event`** operation with the minimum available version of the fixture. You receive a snapshot of each requested fixture.

Each single fixture snapshot always contains all information for the fixture, including all traded markets. There are no delta update snapshot messages that only contain part of the fixture information. The advantage is that losing one or a small number of snapshot messages does not significantly impact content usage models.

Request

```
GET /api/Event/getPreMatchEvent?customerId=12345678-1234-123456789ABCD&id=23456&version=0
HTTP/1.1
Host: v4.customerintegration.bwinfeed.com
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Event Version=”1”Id=”23456”Result=””>
  <Info ContractVersion=”v4_0” TimestampUTC=”2021-09-30T09:33:14.4392546Z”>
  […]
</Event>
```

3.2.3 Process the pre-match bet content

For every response you get you need to process the information received according to the data definition (in an XSD or another source). Extract the relevant information for your needs.

4 Working with in-play content

You can access the in-play ("live") content of the bwin Feed using the following available primary operations:

- **GET Live Calendar:** The list of fixtures announced for in-play trading (see section 4.1)
- **GET Live Index:** The list of currently live traded fixtures (see section 4.2)
- **GET Live Event:** The latest snapshot for one fixture (see section 4.3)

4.1 GET Live Calendar

The `GET Live Calendar` operation returns a full list of announced live fixtures.

`GET Live Calendar` is implemented using *HTTP Long Polling* (see section 2.6).

Request syntax

```
GET /api/Index/livecalendar?customerId=customerID
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<CalendarEvents […]>
  <Event Version="26496019" Id="315448" Result="">
    <Info ContractVersion="v4_0" TimestampUTC="2021-09-30T09:33:14.4392546Z">
    […]
  </Event>
  <Event Version="635076592576173996" Id="328022">
    <Info ContractVersion="v4_0" TimestampUTC="2021-09-30T09:35:14.4392546Z">
    […]
  </Event>
</CalendarEvents>
```

Response elements

The elements in the response are:

Element	Description
`CalendarEvents`	The container for all fixture information. **Type:** Container
`Event`	The container for one fixture information. **Type:** Container
`(others)`	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`406`	*Not acceptable*. The requested response format is not supported.
`500`	*Internal server error*. A non-recoverable error occurred.

4.2 GET Live Index

The `GET Live Index` operation returns a list of events in the in-play ("live") phase.

`GET Live Index` is implemented using *HTTP Long Polling* (see section 2.6).

Request syntax

```
GET api/Index/liveindex?customerId=customerId
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
`customerId`	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Index […]>
  <Version>16334</Version>
  <Events>
    <IndexEvent>
      […]
    </IndexEvent>
  </Events>
</Index>
```

Response elements

The elements in the response are:

Element	Description
`Index`	The container for the whole index. **Type:** Container
`Version`	The version number of the index. **Type:** Long
`Events`	The container for all fixture information in the index. **Type:** Container
`IndexEvent`	The container for one fixture information. **Type:** Event
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`406`	*Not acceptable*. The requested response format is not supported.
`500`	*Internal server error*. A non-recoverable error occurred.

Entain

4.3 GET Live Event

The `GET Live Event` operation checks for updates of an event based on the version number provided and returns the latest snapshot for the event.

`GET Live Event` is implemented using *HTTP Long Polling* (see section 2.6).

Request syntax

```
GET /api/Event/getLiveEvent?customerId=customerId&id=id&version=version
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
id	The unique identifier of the fixture. **Type:** Integer **Default:** None	Yes
version	The version number of the fixture. If the fixture version provided is greater or equal to the version of the last fixture snapshot, then the event operation holds the request until a new fixture snapshot is available. If there is no new snapshot available within 15 seconds, the last event snapshot is returned. **Type:** Long **Default:** The version of the last event snapshot.	No
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Event […] Version=”63” Id=”12345” Result=”8:2”>
  <Info ContractVersion=”4_0” TimestampUTC=”2021-09-30T09:33:14.4392546Z”>
  […]
</Event>
```

Response elements

The elements in the response are:

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The contract version and timestamp of the response. **Type:** Info
`EventMetaInfo`	The container for the fixture-specific data. **Type:** Container
`Markets`	The container for the markets of the fixture. **Type:** Container
`Scoreboard`	The container for the score updates of the fixture. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`404`	*Not found*. The requested resource was not found.
`406`	*Not acceptable*. The requested response format is not supported.
`500`	*Internal server error*. A non-recoverable error occurred.

5 Working with pre-match content

You can access the pre-match content of the bwin Feed using the following available primary operations:

- **GET Prematch Index:** A list of pre-match traded fixtures (see section 5.1)
- **GET Prematch Event:** The latest snapshot for one fixture (see section 5.2)

5.1 GET Prematch Index

The `GET Prematch Index` operation returns a list of currently traded pre-match events.

`GET Prematch Index` is implemented using *HTTP Long Polling* (see section 2.6).

Request syntax

```
GET /api/Index/prematchindex?customerid=customerId&version=version
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
version	The version number of the index. If the index version provided is greater or equal to the index version of the last index snapshot, then the index operation holds the request until a new index snapshot is available. If there is no new snapshot available within 15 seconds, the latest index snapshot is returned. **Type:** Long **Default:** The version of the last index snapshot.	No
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Index>
  <Version>635077558334252997</Version>
  <Events>
    <IndexEvent>
      […]
    </IndexEvent>
  </Events>
</Index>
```

Response elements

The elements in the response are:

Element	Description
`Index`	The container for the index information. **Type:** Container
`Version`	The version number of the index. **Type:** Long
`Events`	The container for all fixture information. **Type:** Container
`IndexEvent`	The container for one fixture information. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`406`	*Not acceptable*. The requested response format is not supported.
`500`	*Internal server error*. A non-recoverable error occurred.

5.2 GET Prematch Event

The `GET Prematch Event` operation gets the event with the provided version number or a more recent version, if available.

Request syntax

```
GET /api/Event/getPreMatchEvent?customerId=customerId&id=id&version=version
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
id	The unique identifier of the fixture. **Type:** Integer **Default:** None	Yes
version	The minimum version number of the fixture. The request is only successful, if a version number equal to or greater than *version* is available. If the version provided is greater or equal to the version of the last fixture snapshot, then the event operation holds the request until a new fixture snapshot is available. If there is no new snapshot available within 15 seconds, an error code is returned. **Type:** Long **Default:** The version of the last event snapshot.	No
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Event […] Version=”6350765” Id=”12345”>
  <Info ContractVersion=”v4_0” TimestampUTC=”2021-09-30T09:33:14.4392546Z”>
  <EventMetaInfo Id=”11337611” Name=”Season 2021/2022 - World Cup” […]>
    […]
  </EventMetaInfo>
  <Markets>
    […]
  </Markets>
</Event>
```

Response elements

The elements in the response are:

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The contract version and timestamp of the response. **Type:** Info
`EventMetaInfo`	The container for the fixture-specific data. **Type:** Container
`Markets`	The container for the markets of the fixture. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`404`	*Not found.* The requested resource was not found.
`406`	*Not acceptable*. The requested response format is not supported.
`500`	*Internal server error*. A non-recoverable error occurred.

6 Working with all fixtures

If you do not want to distinguish between pre-match and in-play content when working with the bwin Feed, you can use the following available primary operation:

- **GET Event:** The latest snapshot for one fixture (see section 6.1)

6.1 GET Event

The `GET Event` operation checks for updates of an event based on the version number provided and returns the latest snapshot for the event. Using `GET Event`, you can request both pre-match and in-play fixture information.

> **NOTICE:** While this is one endpoint for both pre-match and in-play fixtures, make sure to use the proper implementation strategy for each type (see section 10). For instance, use long-polling only for requesting in-play fixtures.

Request syntax

```
GET /api/Event/getEvent?customerId=customerId&id=id&version=version
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
id	The unique identifier of the fixture. **Type:** Integer **Default:** None	Yes
version	The minimum version number of the fixture. The request is only successful, if a version number equal to or greater than *version* is available. If the version provided is greater or equal to the version of the last fixture snapshot, then the event operation holds the request until a new fixture snapshot is available. If there is no new snapshot available, the current snapshot is returned. **Type:** Long **Default:** The version of the last event snapshot.	No
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<Event […] Version=”3085” Id=”12345”>
  <Info ContractVersion=”v4_0” TimestampUTC=”2021-09-30T09:33:14.4392546Z”>
  <EventMetaInfo Id=”11337611” Name=”Season 2021/2022 - World Cup” […]>
    […]
  </EventMetaInfo>
  <Markets>
    […]
  </Markets>
</Event>
```

Response elements

The elements in the response are:

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The contract version and timestamp of the response. **Type:** Info
`EventMetaInfo`	The container for the fixture-specific data. **Type:** Container
`Markets`	The container for the markets of the fixture. **Type:** Container
`Scoreboard`	The container for the score updates of the fixture. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`404`	*Not found.* The requested resource was not found.
`406`	*Not acceptable*. The requested response format is not supported.
`409`	*Conflict*. The request could not be completed because Long Polling was aborted.
`500`	*Internal server error*. A non-recoverable error occurred.

7 Content recovery

If the cut-off date for a fixture has been reached and it does not get any updates for an hour, then the event will no longer be accessible in the `GET Event` endpoints. This applies both to pre-match and in-play ("live") fixtures.

However, the latest snapshot will be available in the `GET Latest Snapshot` endpoint for recovery purposes. The retention time is 10 days in the Production system and 3 days in the Customer Integration system:

- **GET Latest Snapshot:** The latest snapshot for one event (see section 7.1)
- **GET Event Log:** The event history for a single pre-match or in-play fixture (see section 7.2)

7.1 GET Latest Snapshot

The `GET Latest Snapshot` operation returns the latest snapshot for a single pre-match or in-play fixture.

Request syntax

```
GET api/Logs/getLatestSnapshot?customerId=customerId&id=id
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
id	The unique identifier of the fixture. **Type:** Integer **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
<Event […] Version=”6350765” Id=”12345” Result=”8:2”>
  <Info ContractVersion=”v4_0” TimestampUTC=”2021-09-30T09:33:14.4392546Z”>
  <EventMetaInfo Id=”11337611” Name=”SK Rapid Wien – Manchester United” […]>
    […]
  </EventMetaInfo>
  <Markets>
    […]
  </Markets>
</Event>
```

Response elements

The elements in the response are:

Element	Description
Event	The container for one fixture information. **Type:** Event
Info	The contract version and timestamp of the response. **Type:** Info
EventMetaInfo	The container for the fixture-specific data. **Type:** Container
Markets	The container for the markets of the fixture. **Type:** Container
Scoreboard	The container for the score updates of the fixture. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
400	*Bad request*. The input is invalid.
401	*Unauthorized*. The access has been denied.
404	*Not found.* The requested resource was not found.
406	*Not acceptable*. The requested response format is not supported.
409	*Conflict*. The request could not be completed because Long Polling was aborted.
500	*Internal server error*. A non-recoverable error occurred.

7.2 GET Event Log

The `GET Logs` operation returns the event history for a single pre-match or in-play fixture.

Request syntax

```
GET api/Logs/getLogs?customerId=customerId&id=id
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
id	The unique identifier of the fixture. **Type:** Integer **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
<string>Logs were saved in file(s): Event log file path directory</string>
```

Response elements

The elements in the response are:

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The contract version and timestamp of the response. **Type:** Info
`EventMetaInfo`	The container for the fixture-specific data. **Type:** Container
`Markets`	The container for the markets of the fixture. **Type:** Container
`Scoreboard`	The container for the score updates of the fixture. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`400`	*Bad request*. The input is invalid.
`401`	*Unauthorized*. The access has been denied.
`404`	*Not found.* The requested resource was not found.
`406`	*Not acceptable*. The requested response format is not supported.
`409`	*Conflict*. The request could not be completed because Long Polling was aborted.
`500`	*Internal server error*. A non-recoverable error occurred.

8 Working with translations

You can retrieve available content translations using the following operations:

- **GET Competition Translations:** Translations for competition names (see section 8.1)
- **GET Fixture Translations:** Translations for fixture names (see section 8.2)
- **GET Team Translations:** Translations for team names (see section 8.3)
- **GET Region Translations:** Translations for region names (see section 8.4)
- **GET Sport Translations:** Translations for sport names (see section 8.5)

The translations are updated every 30 minutes and are available in more than 20 different languages.

Entain

8.1 GET Competition Translations

The `GET Competition Translations` operation returns translations for competition names in selected sports and in one language.

Request syntax

```
GET
api/Translation/getCompetitionTranslations?customerId=customerId&languageCode=languageCode
&sportIds=sportsIds
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
languageCode	The ISO 639-1 language code. Supported ISO 639-1 language codes are: BG, CA, CS, DA, DE, EL, EN, ES, FR, HR, HU, IT, NL, NO, PL, PT, RO, RU, SK, SL, SV, TR. **Type:** String **Default:** None	Yes
sportsIds	A comma-separated list of sport IDs. **Type:** String **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<LocalizationResource […]>
  <Language LanguageCode="DE">
    <ResourceGroup GroupId="Competition">
      <Items Key="4630" UIDs="1:4630">Australian Open, Frauen</Items>
      […]
    </ResourceGroup>
  </Language>
</LocalizationResource>
```

Response elements

The elements in the response are:

Element	Description
`LocalizationResource`	The container for the localization information. **Type:** Container
`Language`	The container for the language information. **Type:** Container
`ResourceGroup`	The container for the translation items. **Type:** Container
`Items`	One translation string, including a key specific to the resource group. The key is used in conjunction with the identifiers returned for all translatable content within an event snapshot. **Type:** String

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`404`	*Not found.* The specified language was not found.
`500`	*Internal server error.* A non-recoverable error occurred.

8.2 GET Fixture Translations

The `GET Fixture Translations` operation returns translations for fixture names in selected sports and in one language.

Request syntax

```
GET api/Translation/getFixtureTranslations?customerId=customerId&languageCode=languageCode
&sportIds=sportsIds
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
languageCode	The ISO 639-1 language code. Supported ISO 639-1 language codes are: BG, CA, CS, DA, DE, EL, EN, ES, FR, HR, HU, IT, NL, NO, PL, PT, RO, RU, SK, SL, SV, TR. **Type:** String **Default:** None	Yes
sportsIds	A comma-separated list of sport IDs. **Type:** String **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<LocalizationResource […]>
  <Language LanguageCode="DE">
    <ResourceGroup GroupId="Fixture">
      <Items Key="11949982" UIDs="1:11949982">Boston Celtics bei New York Knicks</Items>
      […]
    </ResourceGroup>
  </Language>
</LocalizationResource>
```

Response elements

The elements in the response are:

Element	Description
LocalizationResource	The container for the localization information. **Type:** Container
Language	The container for the language information. **Type:** Container
ResourceGroup	The container for the translation items. **Type:** Container
Items	One translation string, including a key specific to the resource group. The key is used in conjunction with the identifiers returned for all translatable content within an event snapshot. **Type:** String

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
404	*Not found.* The specified language was not found.
500	*Internal server error.* A non-recoverable error occurred.

8.3 GET Team Translations

The `GET Team Translations` operation returns translations for team names in selected sports and in one language.

Request syntax

```
GET api/Translation/getTeamTranslations?customerId=customerId&languageCode=languageCode
&sportIds=sportsIds
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
languageCode	The ISO 639-1 language code. Supported ISO 639-1 language codes are: BG, CA, CS, DA, DE, EL, EN, ES, FR, HR, HU, IT, NL, NO, PL, PT, RO, RU, SK, SL, SV, TR. **Type:** String **Default:** None	Yes
sportsIds	A comma-separated list of sport IDs. **Type:** String **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<LocalizationResource […]>
  <Language LanguageCode="DE">
    <ResourceGroup GroupId="Team">
      <Items Key="2516287" UIDs="1:2516287">Rapid Wien</Items>
      […]
    </ResourceGroup>
  </Language>
</LocalizationResource>
```

Response elements

The elements in the response are:

Element	Description
`LocalizationResource`	The container for the localization information. **Type:** Container
`Language`	The container for the language information. **Type:** Container
`ResourceGroup`	The container for the translation items. **Type:** Container
`Items`	One translation string, including a key specific to the resource group. The key is used in conjunction with the identifiers returned for all translatable content within an event snapshot. **Type:** String

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`404`	*Not found.* The specified language was not found.
`500`	*Internal server error.* A non-recoverable error occurred.

8.4 GET Region Translations

The `GET Region Translations` operation returns translations for region names in one language.

Request syntax

```
GET api/Translation/getRegionTranslations?customerId=customerId&languageCode=languageCode
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
languageCode	The ISO 639-1 language code. Supported ISO 639-1 language codes are: BG, CA, CS, DA, DE, EL, EN, ES, FR, HR, HU, IT, NL, NO, PL, PT, RO, RU, SK, SL, SV, TR. **Type:** String **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<LocalizationResource […]>
  <Language LanguageCode="DE">
    <ResourceGroup GroupId="Region">
      <Items Key="29" UIDs="1:29">Schweden</Items>
      […]
    </ResourceGroup>
  </Language>
</LocalizationResource>
```

Entain

Response elements

The elements in the response are:

Element	Description
`LocalizationResource`	The container for the localization information. **Type:** Container
`Language`	The container for the language information. **Type:** Container
`ResourceGroup`	The container for the translation items. **Type:** Container
`Items`	One translation string, including a key specific to the resource group. The key is used in conjunction with the identifiers returned for all translatable content within an event snapshot. **Type:** String

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`404`	*Not found.* The specified language was not found.
`500`	*Internal server error.* A non-recoverable error occurred.

8.5 GET Sport Translations

The `GET Sport Translations` operation returns translations for sport names in one language.

Request syntax

```
GET api/Translation/getSportTranslations?customerId=customerId&languageCode=languageCode
HTTP/1.1
Host: host
Accept: application/xml
```

Request parameters

The available request parameters are:

Parameter	Description	Required
languageCode	The ISO 639-1 language code. Supported ISO 639-1 language codes are: BG, CA, CS, DA, DE, EL, EN, ES, FR, HR, HU, IT, NL, NO, PL, PT, RO, RU, SK, SL, SV, TR. **Type:** String **Default:** None	Yes
customerId	The customer access identifier to authorize the request. **Type:** String **Default:** None	Yes

Response syntax

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<LocalizationResource […]>
  <Language LanguageCode="DE">
    <ResourceGroup GroupId="Sport">
      <Items Key="45">Kampfsport</Items>
      […]
    </ResourceGroup>
  </Language>
</LocalizationResource>
```

Response elements

The elements in the response are:

Element	Description
LocalizationResource	The container for the localization information. **Type:** Container
Language	The container for the language information. **Type:** Container
ResourceGroup	The container for the translation items. **Type:** Container
Items	One translation string, including a key specific to the resource group. The key is used in conjunction with the identifiers returned for all translatable content within an event snapshot. **Type:** String

Error responses

If an error occurs, one of the following error codes is returned:

HTTP status code	Description
`404`	*Not found.* The specified language was not found.
`500`	*Internal server error*. A non-recoverable error occurred.

9 Content and lifecycles

This part of the documentation gives you a detailed reference of the available content types in the bwin Feed and of their lifecycles:

- Live fixtures content (see section 9.1)
- Live fixtures lifecycle (see section 9.2)
- Pre-match fixtures content (see section 9.3)
- Pre-match fixtures lifecycle (see section 9.4)
- Market resulting lifecycle (see section 9.5)
- Option lifecycle (see section 9.6)

> **NOTICE:** All examples in this part use XML as response format.

9.1 Live fixtures content

Request

```
GET api/Index/liveindex?customerId=12345678-1234-123456789ABCD&version=0
HTTP/1.1
Host: host
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<CustomVersionIndex>
   <version>0</version>
   <events>
      <id>0</id>
      <version>0</version>
      <eventDateUtc>2021-10-13T11:32:21.411Z</eventDateUtc>
      <eventCutOffDateUtc>2021-10-13T11:32:21.411Z</eventCutOffDateUtc>
      <willBeOfferedLive>true</willBeOfferedLive>
      <sportId>0</sportId>
      <tradingVersion>1</tradingVersion>
      <internalId>0</internalId>
   </events>
</CustomVersionIndex>
```

Response elements

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The container for the contract and version information. **Type:** Info
`EventMetaInfo`	The container for the meta information of the event. **Type:** Container
`Markets`	The container for the market information of the event. **Type:** Container
`Scoreboard`	The container for the scoreboard information of the event. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

9.2 Live fixtures lifecycle

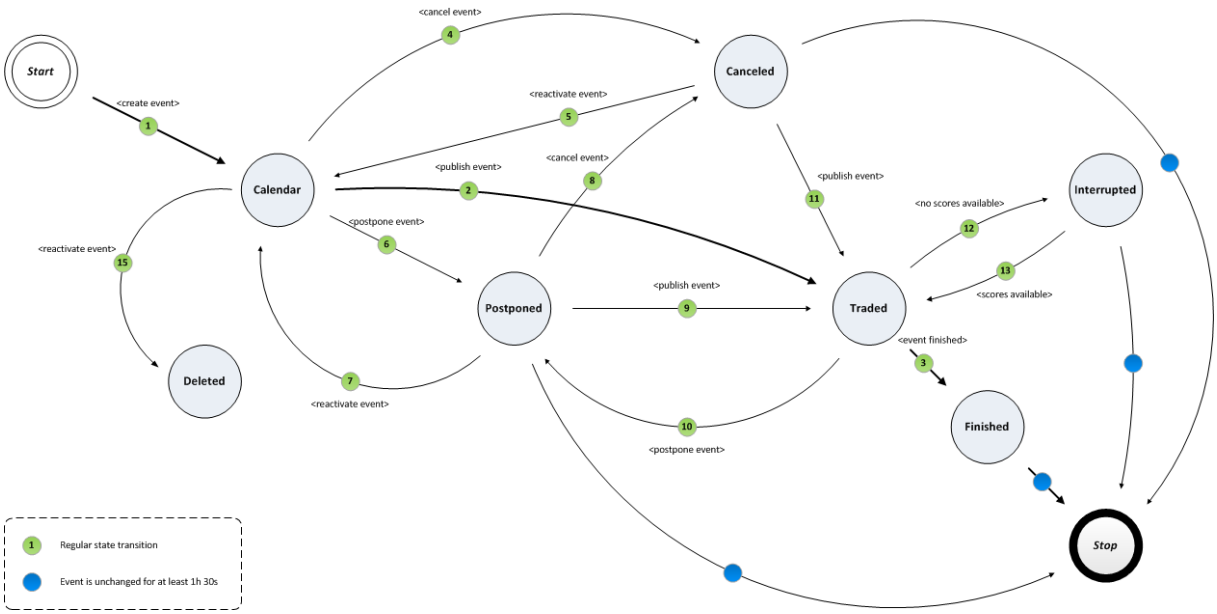


Fig. 3: Live fixtures lifecycle: states and transitions

States and transitions

The following state transitions happen during the lifecycle:

#	Reason
1	The fixture is created in the *Calendar*.
2, 9, 11	The fixture is set to *Traded*.
3	The fixture is set to *Finished*.
4, 8	The fixture is set to *Cancelled*.
5, 7	The fixture is reactivated In the *Calendar*.
6, 10	The fixture is set to *Postponed*.
12	No scores are currently available for the event. The fixture is set to *Interrupted*.
13	Scores are again available for the event. The fixture is set to *Traded*.
15	The fixture is set to *Deleted*.

Response

```
<EventMetaInfo Id="8061230" Name="Liverpool FC - Japan" Date="2021-09-30T22:00:00Z"
CutOffDate="2021-10-02T22:00:00Z" TradingType="Live" IsCanceled="false"
IsInterrupted="false" IsDeleted="false" IsPostponed="false" EventType="Tournament"
EventViewType="European" IsEnabled="false" IsDisplayed="true" TradingPartition="2"
UID="2:8061230">
  <LinkEventIds>
    <Link Id="2934129" Type="PreMatch"/>
  </LinkEventIds>
  <Teams>
    <TournamentTeams>
      <Team Id="2147190210" Name="Liverpool FC" ShirtColor="#000000" ShortsColor="#000000"
Order="Team01" HomeAway="Home" UID="2:293437"/>
      <Team Id="2147180711" Name="Japan" ShirtColor="#ffffff" ShortsColor="#ffffff"
Order="Team02" HomeAway="Away" UID="2:302936"/>
    </TournamentTeams>
  </Teams>
  <Sport Id="4" Name="Football"/>
  <Region Id="2147483641" Name="World" Code="WRL" UID="2:6"/>
  <League Id="2147481560" Name="Asian Cup" GroupId="0" GroupName="" StageId="0"
StageName="" UID="2:2087">
    <LinkLeagueIds>
      <Link Id="2147481560" UID="2:2087"/>
    </LinkLeagueIds>
  </League>
  <Broadcasters/>
  <MediaStreams/>
  <DataSupplier/>
</EventMetaInfo>
```

Entain

Status parameters in the meta info

The `EventMetaInfo` contains 4 status parameters. If one of these parameters is `true`, then the fixture is in the corresponding exception state:

- `IsCanceled`
- `IsInterrupted`
- `IsPostponed`
- `IsDeleted`

In the “happy path” states *Calendar*, *Traded*, and *Finished*, all 4 parameters are `false`.

9.3 Pre-match fixtures content

Request

```
GET /api/Index/prematchindex?customerid=12345678-1234-123456789ABCD&version=0
HTTP/1.1
Host: host
Accept: application/xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<CustomVersionIndex>
   <version>0</version>
   <events>
      <id>0</id>
      <version>0</version>
      <eventDateUtc>2021-10-13T11:32:21.418Z</eventDateUtc>
      <eventCutOffDateUtc>2021-10-13T11:32:21.418Z</eventCutOffDateUtc>
      <willBeOfferedLive>true</willBeOfferedLive>
      <sportId>0</sportId>
      <tradingVersion>1</tradingVersion>
      <internalId>0</internalId>
   </events>
</CustomVersionIndex>
```

Response elements

Element	Description
`Event`	The container for one fixture information. **Type:** Event
`Info`	The container for the contract and version information. **Type:** Info
`EventMetaInfo`	The container for the meta information of the event. **Type:** Container
`Markets`	The container for the market information of the event. **Type:** Container
(others)	Refer to the accompanying XSD file or other sources for the complete data structure definition.

9.4 Pre-match fixtures lifecycle

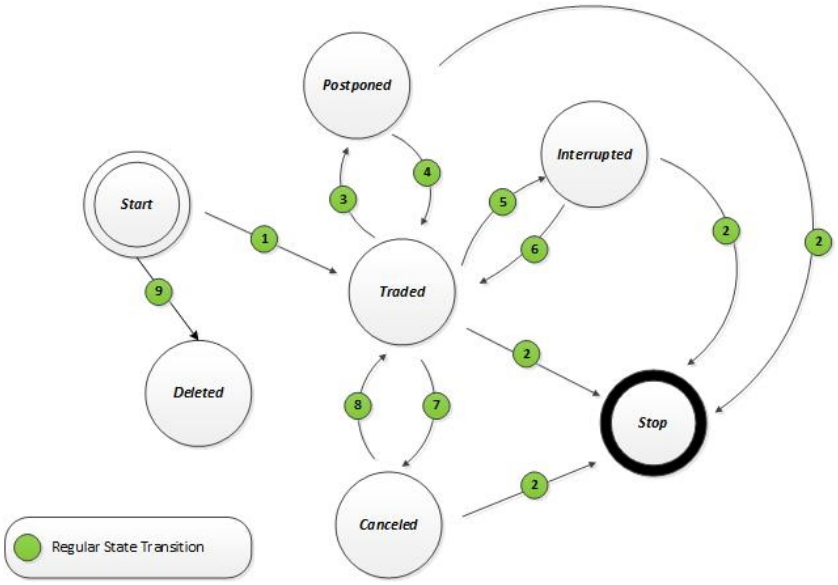


Fig. 4: Pre-match fixtures lifecycle: states and transitions

States and transitions

The following state transitions happen during the lifecycle:

#	Reason
1	The fixture is created and set to *Traded*.
2	The cut-off date of the fixture is reached. Trading is disabled.
3	The outside world event is rescheduled. The fixture is set to *Postponed*.
4	The open date of the fixture is reached. The fixture is set to *Traded*.
5	A trading system failure happens. The fixture is set to *Disabled/Interrupted*.
6	The trading system is available again. The fixture is set to *Traded*.
7	The outside world event is cancelled. The fixture is set to *Cancelled*.
8	The outside world event is reactivated (unlikely). The fixture is set to *Traded*.
9	The fixture is set to *Deleted*.

Response

```
<EventMetaInfo Id="8061230" Name="Liverpool FC - Japan" Date="2021-09-30T22:00:00Z"
CutOffDate="2021-10-02T22:00:00Z" TradingType="PreMatch" IsCanceled="false"
IsInterrupted="false" IsDeleted="false" IsPostponed="false" EventType="Tournament"
EventViewType="European" IsEnabled="true" IsDisplayed="true" TradingPartition="2"
UID="2:8061230">
  <LinkEventIds>
    <Link Id="2934129" Type="Live"/>
  </LinkEventIds>
  <Teams>
    <TournamentTeams>
      <Team Id="2147190210" Name="Liverpool FC" ShirtColor="#000000" ShortsColor="#000000"
Order="Team01" HomeAway="Home" UID="2:293437"/>
      <Team Id="2147180711" Name="Japan" ShirtColor="#ffffff" ShortsColor="#ffffff"
Order="Team02" HomeAway="Away" UID="2:302936"/>
    </TournamentTeams>
  </Teams>
  <Sport Id="4" Name="Football"/>
  <Region Id="2147483641" Name="World" Code="WRL" UID="2:6"/>
  <League Id="2147481560" Name="Asian Cup" GroupId="0" GroupName="" StageId="0"
StageName="" UID="2:2087">
    <LinkLeagueIds>
      <Link Id="2147481560" UID="2:2087"/>
    </LinkLeagueIds>
  </League>
  <Broadcasters/>
  <MediaStreams/>
  <DataSupplier/>
</EventMetaInfo>
```

Status parameters in the meta info

The `EventMetaInfo` contains 4 status parameters. If one of these parameters is `true`, then the fixture is in the corresponding exception state:

- `IsCanceled`
- `IsInterrupted`
- `IsPostponed`
- `IsDeleted`

In the state *Traded*, all 4 parameters are `false`.

> **NOTICE:** For pre-match fixtures, `IsInterrupted` will never be true.

9.5 Market resulting lifecycle

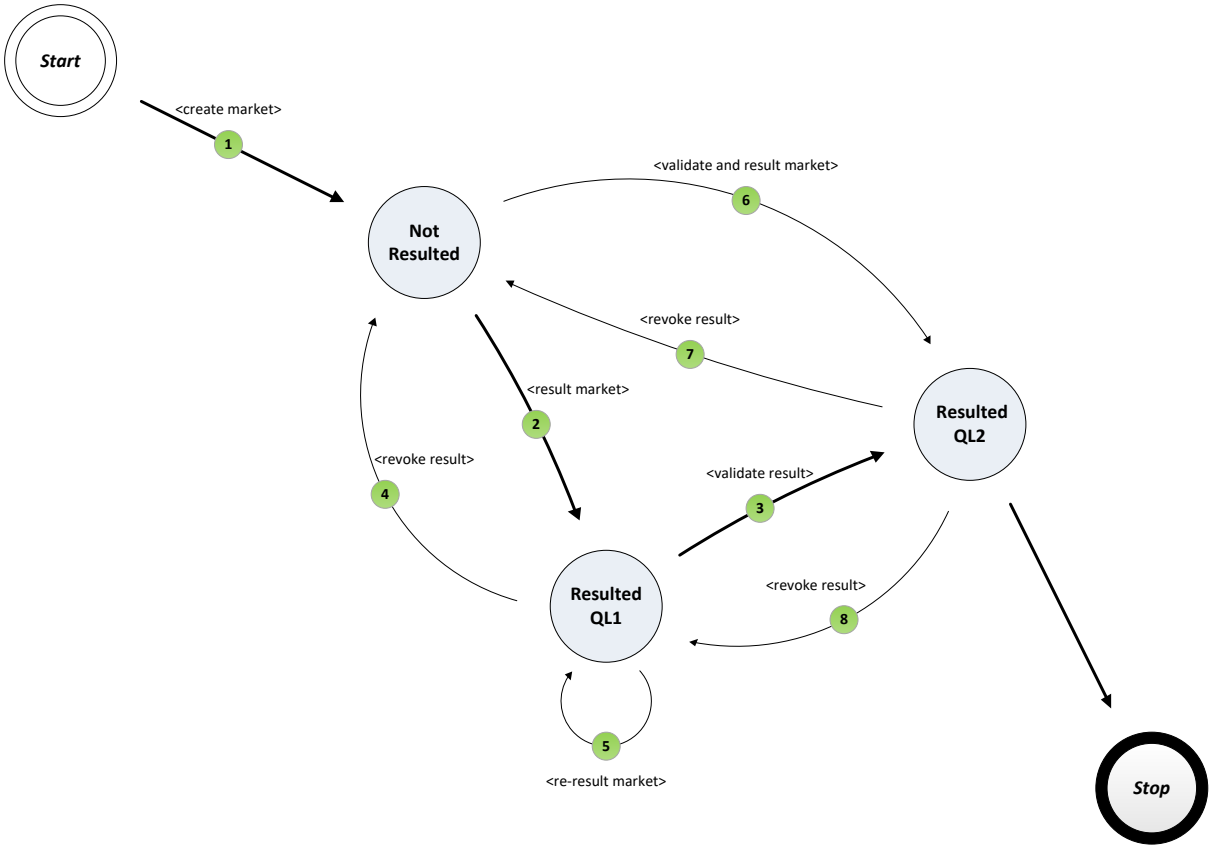


Fig. 5: Market resulting lifecycle: states and transitions

States and transitions

The following state transitions happen during the lifecycle:

#	Reason
1	The market is created and set to *Not resulted*.
2	The market is resulted and set to *Resulted QL1*.
3	The result is paid out. The market is set to *Resulted QL2*.
4, 7, 8	The pay-out is revoked. The market state is moved backwards.
5	The market is re-resulted and stays at *Resulted QL1*.
6	The market is resulted and paid out in one step and set to *Resulted QL2*.

Market info

```
<Markets>
  <Market Id="12675835" Name="3way - result" OpenDate="2021-07-05T08:00:00Z"
CutOffDate="2021-07-05T14:00:00Z" IsEnabled="true" ResultingType="NotResulted">
    <Options>
      […]
    </Options>
    <MarketType Name="3way period Y">
      <MarketTypeSport>
        <MarketTypeSoccer Period="RegularTime" />
      </MarketTypeSport>
    </MarketType>
  </Market>
</Markets>
```

Status parameters in the market info

The `Market` info contains 2 status parameters:

- `ResultingType`: The resulting state the market is in: *NotResulted*, *QL1*, *QL2*.
- `IsEnabled`: Can be *True* or *False* when not resulted and is always *False* in *QL1* and *QL2*.

9.6 Option lifecycle

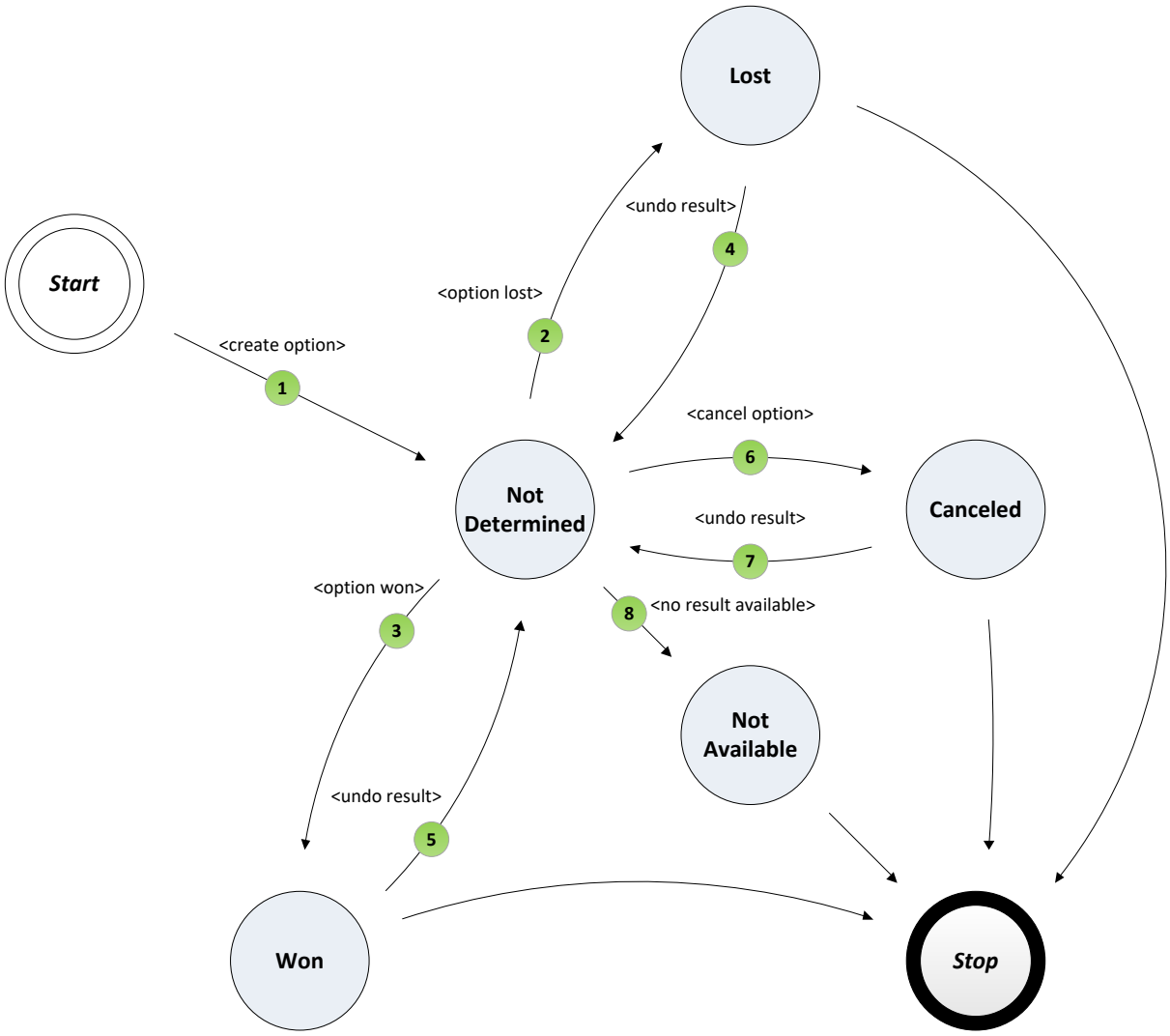


Fig. 6: Option lifecycle: states and transitions

States and transitions

The following state transitions happen during the lifecycle:

#	Reason
1	The option is created and set to *Not determined*.
2	The option is set to *Lost*.
3	The option is set to *Won*.
4, 5, 7	The result is undone. The option is set to *Not determined*.
6	The option is set to *Cancelled*.
8	No results are available. The option is set to *Not available*.

Option info

```
<Options>
  <Option Id="46164481" Name="1" Type="1" IsEnabled="true" Status="NotDetermined"
Odds="8.51039" />
  <Option Id="46164482" Name="X" Type="X" IsEnabled="true" Status="NotDetermined"
Odds="3.38434" />
  <Option Id="46164483" Name="2" Type="2" IsEnabled="true" Status="NotDetermined"
Odds="1.70353" />
</Options>
```

Status parameters in the option info

The `Market` info contains 2 status parameters:

- `Status`: The state the option is in: *NotDetermined*, *Lost*, *Won*, *Canceled*, *NotAvailable*.
- `IsEnabled`: Can be *True* or *False* when not determined and is always *False* in *Lost*, *Won*, and *Canceled*.

10 Implementation strategies

There are different implementation strategies for the integration of the bwin Feed. Which strategy works best for you depends on which result shall be achieved: One strategy might lead to a low latency under the circumstance of a high number of concurrent connections while another strategy will provide the opposite result.

This section highlights the available configuration options and the associated positive and negative effects for the integration of the bwin Feed API via HTTP Long Polling Interface. This shall make the recommended usage of the service visible to you. The implementation can also be optimized for in-play or pre-match content. The following topics are covered:

- Snapshot versioning (see section 10.1)
- In-play content strategy (see section 10.2)
- Pre-match content strategy (see section 10.3)

> **NOTICE:** Make sure to have read and understood the long polling (see section 2.6) and “getting started” (see section 3) information before studying the implementation strategies.

10.1 Snapshot versioning

The Snapshot Versioning is an important concept for using the Long Polling Interface. If you decide to utilize the snapshot versioning, the bwin Feed knows when an update is available and when not.

Each data snapshot has a version number by default. Snapshot Versioning means that you send a version number with your request. The server compares your incoming version number with the current one and reacts accordingly.

10.1.1 Snapshot versioning used with in-play content

Scenario – Version outdated

If your provided version number is outdated, the server immediately returns the latest snapshot, because an update already exists.

Scenario – Version not outdated

If your provided version is *greater than* the latest server snapshot version, the operation holds the request for 15 seconds. If a newer snapshot becomes available within this time frame, it is immediately returned. If there is no new snapshot available within 15 seconds, the latest snapshot is returned, regardless of the version number.

10.1.2 Snapshot versioning used with pre-match content

The snapshot versioning for requesting pre-match content works slightly different than for in-play content. For pre-match content you use constant calling to get the latest version number.

Scenario – Version outdated

If your requested minimum version is *equal to or lower* than the current server version (i.e. there is a newer version on the server), the server returns the newest snapshot.

Scenario – Version not outdated

If your requested minimum version is *greater than* the current server version, the server returns an error code **409**.

10.1.3 Snapshot versioning not used

If you provide no version number with your call, the server immediately returns the latest snapshot. There are some circumstances where the call does not return the latest snapshot immediately: when no update of the event has arrived or the event has expired, for instance.

10.2 In-play content implementation strategy

For retrieving in-play content, two strategies are available:

- **Constant event long polling** (see section 10.2.1)
- **Fixed index polling** (see section 10.2.2)

Which strategy works better for you depends on your system set-up:

- In case of ample bandwidth and processing power, we strongly recommend "Constant Event Long Polling".
- If the bandwidth is low or other resources are less available, "Fix Index Polling" is an alternative, but with the drawback of a higher latency and slower event update detection.

10.2.1 Constant event long polling

The preferred in-play content implementation strategy is "Constant Event Long Polling". The strategy works as follows:

1. The "Liveindex" snapshot is used as table of contents to detect if fixtures have been added or removed.
2. Constantly long-poll the `GET Live Index` operation by using the version from the latest index you received, starting with version "0". Store the returned list of Fixture IDs and their versions.
3. A distinct long polling session is constantly established for each listed fixture in the "index" snapshot.
4. If a fixture is no longer in the index snapshot (i.e. it has been removed from the offer), the long polling session for this fixture is closed.
5. If the fixture reappears later in the index, for instance because of late resulting after the cut-off date, reopen the long polling session for this fixture to fetch the latest result.

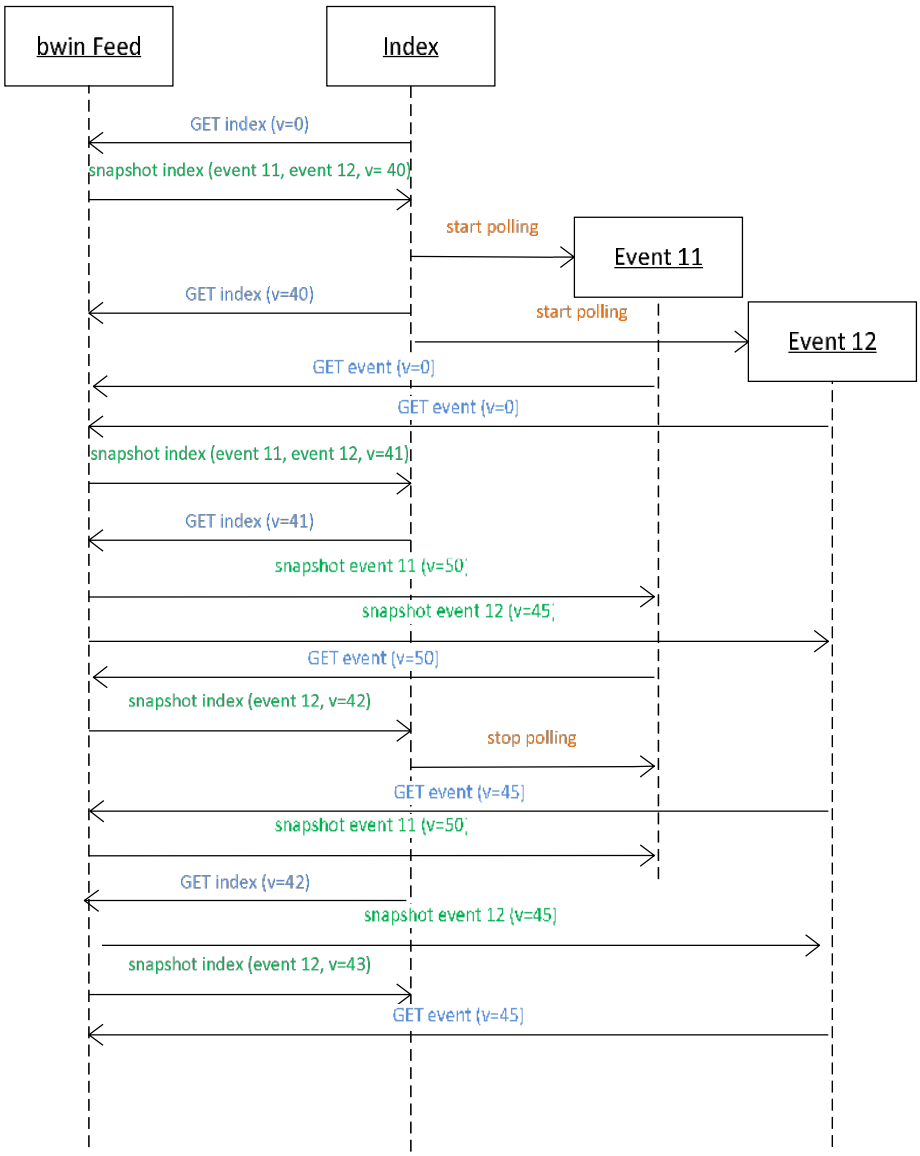


Fig. 7: Constant event long polling

Advantages:

- Early event update detection
- Low latency

Disadvantages:

- High number of open connections
- Higher bandwidth

10.2.2 Fixed index polling

The in-play content implementation strategy “Fixed Index Polling” is supported, but not recommended if low latency is required. The strategy works as follows:

1. The index snapshot is constantly polled in a fixed interval (e.g. 2 seconds), but without a version number.
2. The returned index snapshot includes the event version. Compare this version number with the one from the last update. If a fixture version number has changed, the fixture snapshot has been updated.
3. For each updated fixture, use **`GET Live Event`**. The call immediately returns a snapshot.

> IMPORTANT NOTICE: The returned snapshot might not be the newest one, if there was no update to the events!

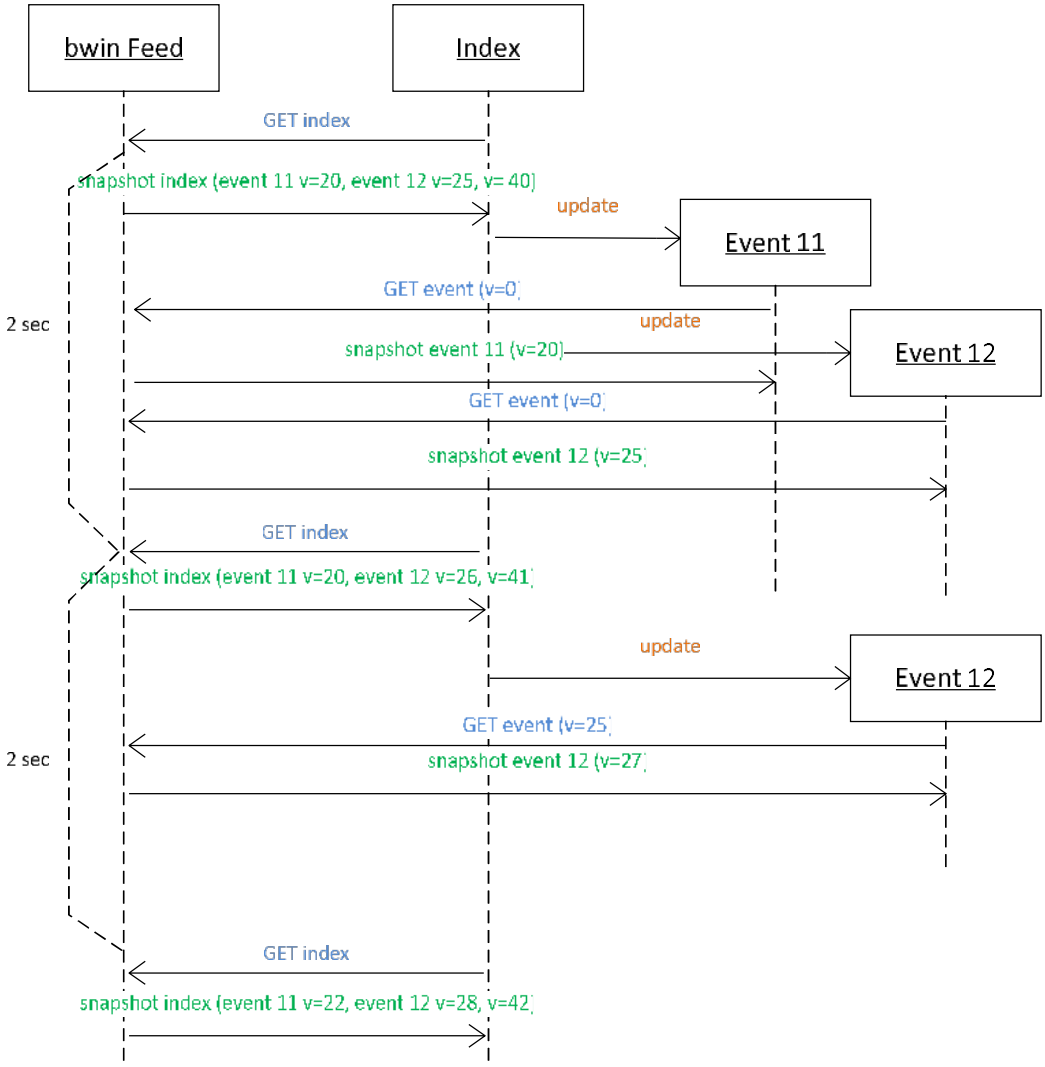


Fig. 8: Fixed index polling

Advantages:

- Fewer connections and easier connection handling

Disadvantages:

- Slow event update detection
- Higher latency

10.2.3 In-play implementation pitfalls

Try to avoid the following common implementation mistakes:

Differential update

A common mistake would be using the following strategy, which can cause updates to be missed:

1. The index snapshot is constantly long polled with **`GET Live Index`**.
2. The returned index snapshot includes the event version. You compare this version number with the one from the last update. If a fixture version number has changed, the fixture snapshot has been updated.
3. For each updated fixture, you call the **`GET Live Event`** operation without version and a snapshot is immediately returned.

The returned snapshot is not the snapshot with the expected update!

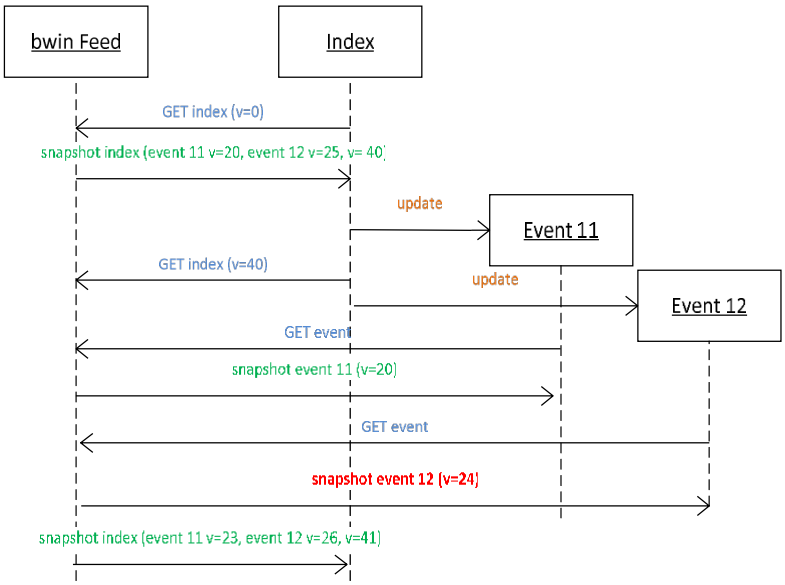


Fig. 9: Differential update mistake

Impact:

- High latency
- Slow event update detection
- Event updates can be missed

Long polling with wrong version

A common mistake is to use the event version from the index snapshot for the **`GET Live Event`** operation. When a long polling session for an event is established with the index event version, it can take up to 15 seconds to get the requested snapshot. Do not use the index event version with **`GET Live Event`** long polling operations.

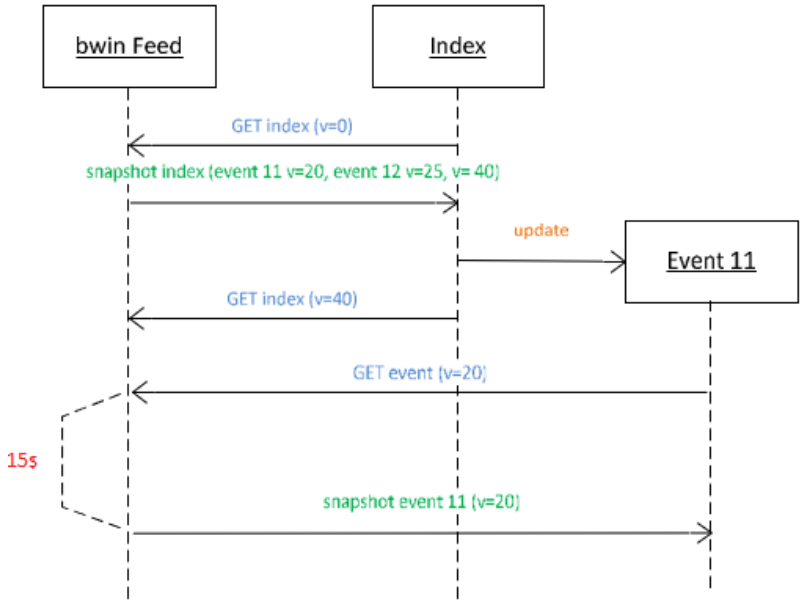


Fig. 10: Long polling with wrong version

Impact:

- High latency for the **`GET Live Event`** operation

10.3 Pre-match content implementation strategy

The Long Polling Interface for pre-match content must be used differently than the Long Polling Interface for in-play content because of following reasons:

- The sheer number of fixtures and the update rate are not suitable for a constant fixture long polling approach.
- The resulting information for specific markets is not guaranteed to be available when the cut-off date of the fixture is reached. There might be late resulting, for instance for 3-way markets.

10.3.1 Recommended pre-match strategy

The recommended strategy for the pre-match Long Polling Interface works as follows:

1. The “PreMatchIndex” snapshot is used as table of contents to detect if fixtures have been added or removed.
2. Constantly long poll the **`GET Prematch Index`** operation by using the version from the latest index you received, starting with version “0”. Store the returned list of Fixture IDs and their versions.
3. If a fixture has been added or the version has been updated, call the **`GET Prematch Event`** operation for the new or updated items and provide the new version number as minimum available version.
4. Feed the returned fixture snapshot directly into your trading system. **`GET Prematch Event`** guarantees that at least the minimum version of the event snapshot is returned.
5. Repeat steps 3 and 4 for each successful **`GET Prematch Index`** call.

10.4 Data compression

The bwin Feed is able to provide XML-based data, which can consume much bandwidth, depending on the bet content. Therefore, most bwin Feed users will need to optimize the bandwidth usage for one or the other reason.

When requesting data from the bwin Feed, you can choose to receive the response in a compressed format.

HTTP 1.1 compression

The bwin Feed uses the HTTP 1.1 compression[2]. The compression method is GZIP, which is supported by most mainstream programming language frameworks.

You can use the compression as follows:

1. You send a HTTP request and include the header attribute **`Accept-Encoding: gzip`**.
2. When processing the request, the server detects the **`Accept-Encoding`** header attribute and compresses the response data.
3. The response contains **`Content-Encoding: gzip`** to indicate the compressed content.

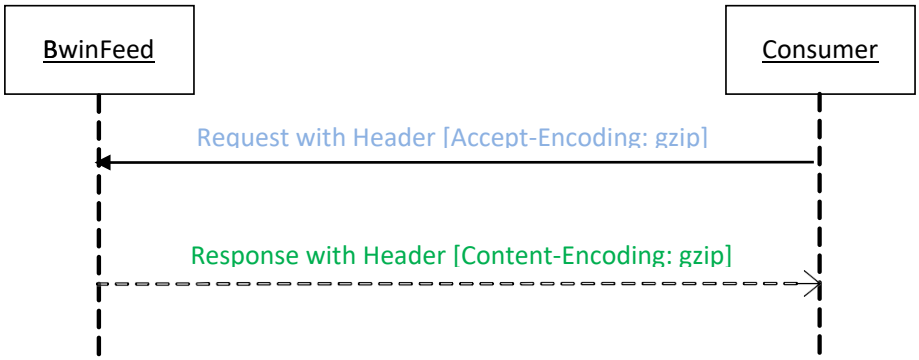


Fig. 11: HTTP 1.1 compression

[2] See https://tools.ietf.org/html/rfc2616

Entain

10.5 Examples of client implementations

10.5.1 C# / .NET

Implementation strategy “HttpClient”

The **`HttpClient`** class provides an easy way to use HTTP compression[3]:

```
using (var client =
new HttpClient(new HttpClientHandler() { AutomaticDecompression = DecompressionMethods.GZip
}))
{
  client.DefaultRequestHeaders
    .AcceptEncoding
    .Add(StringWithQualityHeaderValue.Parse("gzip"));

  using (var response = await client.GetAsync(here URL))
  {
    //here response handling code
  }
}
```

Implementation strategy “HttpWebRequest”

The **`HttpWebRequest`** class is an abstraction level lower as **`HttpClient`** and allows the direct configuration of HTTP compression for a specific HTTP request[4]:

```
var request = (HttpWebRequest) WebRequest.Create(here URL); request.Method = "GET";
request.Headers.Add("Accept-Encoding", "gzip");
request.AutomaticDecompression = DecompressionMethods.GZip;

using (var response = request.GetResponse())
{
  //here response handling code
}
```

[3] See https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-5.0
[4] See https://docs.microsoft.com/en-us/dotnet/api/system.net.httpwebrequest?view=net-5.0

10.5.2 Java 1.7 and HttpClient 4.3

Implementation strategy “HttpClientBuilder”

The **`HttpClientBuilder`** class provides the possibility to add interceptors for the HTTP compression handling[5]:

```
CloseableHttpClient client = HttpClientBuilder.create()
  .addInterceptorFirst(new HttpRequestInterceptor() { @Override
  public void process(HttpRequest httpRequest, HttpContext httpContext)
  throws    HttpException, IOException {

    if (!httpRequest.containsHeader("Accept-Encoding")) {
      httpRequest.addHeader("Accept-Encoding","gzip");
    }
  }})

  .addInterceptorFirst(new HttpResponseInterceptor() {
  @Override public void process(HttpResponse httpResponse, HttpContext httpContext)
  throws HttpException, IOException {

    HttpEntity entity = httpResponse.getEntity(); if (entity != null) {

      Header header = entity.getContentEncoding(); if (header != null) {
        HeaderElement[] codecs = header.getElements();
        for (HeaderElement codec : codecs) {
          if(codec.getName().equalsIgnoreCase("gzip")) {
          httpResponse.setEntity(new GzipDecompressingEntity(httpResponse.getEntity()));
          return;
          }
        }
      }
    }
  }).build();

try {
  HttpGet httpget = new HttpGet(here URL); CloseableHttpResponse
  response = client.execute(httpget);

  try {
    HttpEntity entity = response.getEntity();
    //here response handling code
  }
  finally {
    response.close();
  }
} catch (ClientProtocolException e) { e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
} finally {
  client.close();
}
```

[5] See https://hc.apache.org/httpcomponents-client-ga/

Appendix

Appendix A Rules and restrictions

> CAUTION: Potential blocking!
> If you violate the usage rules and restrictions, the access from your IP address will automatically be blocked.
> → Follow the usage rules listed below.

Scale-out

You are not allowed to scale out your solution based on the bwin Feed API. This means:

- Issue every web service request *only once*.
- Do not connect to one endpoint with multiple environments such as integration, test, and development.
- Do not call the same service multiple times from different IP addresses.
- Do not call the service simultaneously from different nodes (active - active or active - passive).

Request limits

You can send up to 1,500 requests within a time frame of 5 seconds. If you exceed this number:

- All requests above the limit will get an “HTTP 429” response (“too many requests”) instead of the requested data.
- Your access will be blocked for 15 seconds and every request within the blocked period will also get an “HTTP 429” response.

Firewall

The Entain firewall system is protected for instance from SYN flood attacks by incorrectly used or defective implementations. This means:

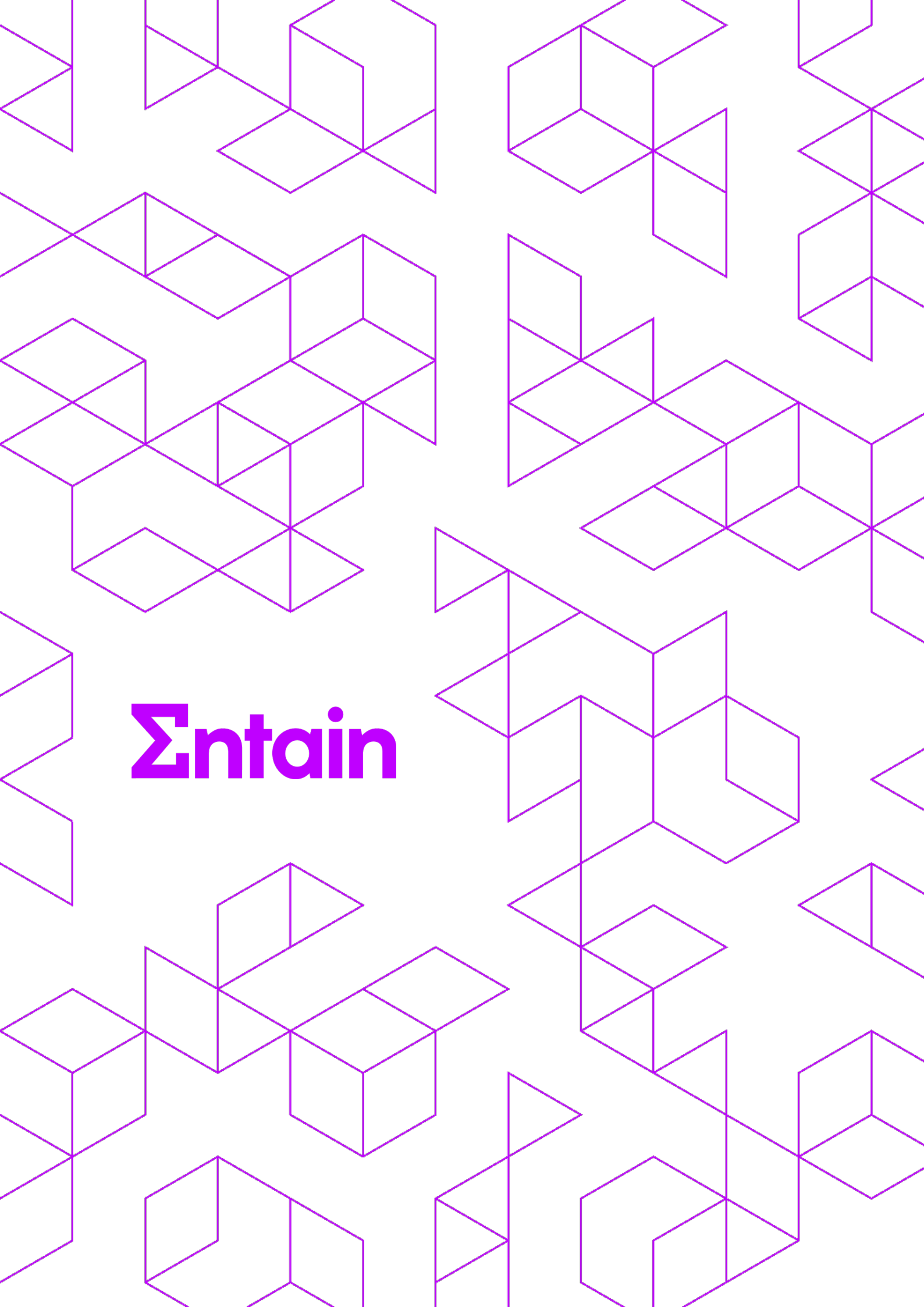
- No more than 2,000 simultaneous connections are allowed at any time.
- Always close all no longer needed HTTP connections.

Appendix B Glossary

The following important terms from the betting industry are used throughout this guide:

Term	Definition
Competition	One entity in our betting system that is the representation of a → *League*. Shall replace the term → *League* on the technical side on the long run.
Event	One entity in our betting system that is the representation of an → *Outside world event* such as a football match or Formula One race. Can be identified via a unique ID. To avoid ambiguity, the term → *Fixture* is being established for events in the betting system instead. For Events, → *Markets* are offered.
Event snapshot	The status of an → *Event*, including its → *Markets* and → *Options* at a certain point in time. Can be retrieved via the bwin Feed.
Fixture	One entity in our betting system that is the representation of an → *Outside world event*. Shall replace the term → *Event* on the long run.
Index	The list of all upcoming and traded → *Fixtures*.
Index snapshot	The status of the → *Index* at a certain point in time. Can be retrieved via the bwin Feed.
Long polling	A technical paradigm, where requests can be kept on record by the server for some time, if no immediate response is available.
League	A grouping of → *Events*, both in the outside world and the betting system such as the Premier League or the Formula One. To avoid ambiguity, the term → *Competition* is being established for leagues in the betting system instead.
Market	One bet we offer for an → *Event*. Typical Markets are “Who will win?” or “Which team will score the first goal?”. Each possible outcome of a market is an individual → *Option*.
Market type	A group of possible → *Markets* with the same systematics but different parameters. Example: “over/under *n* goals”.
Option	One of the potential outcomes of a → *Market*. Typical Options for the Market “Who will win?” would be: “Player A” and “Player B” (and in three-way bets also “Draw”).
Outside world event	One event in the real world such as a golf tournament or a football match.
Result	The outcome of an → *Option*: either won or lost.

Table of Figures


Σntain