MTS SDK

October 2018

Integration guide

betradar
driven by facts

Document version

Version	Author	Date	Comments
1.4	David Hrovat	2018-10-11	Added exclusiveConsumer configuration property
1.3	David Hrovat	2017-11-10	Added port number configuration property
1.2	David Hrovat	2017-09-12	Added provideAdditionalMarketSpecifiers property to config section
1.1	Uros Bregar	2017-09-09	Added description about Rest and Cache log Added accessToken configuration property Updated method SetIdUof
1.0	Uros Bregar	2015-03-13	Initial version

Sportradar AG
www.sportradar.com

Feldlistrasse 2
CH-9000 St. Gallen
Switzerland

P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com

Company Identification
No:CHE-113.075.404

Table of Contents

Sportradar AG
www.sportradar.com

Feldlistrasse 2
CH-9000 St. Gallen
Switzerland

P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com

Company Identification
No:CHE-113.075.404

Getting started

Before starting to use the SDK please read the appropriate MTS documentation found on www.betradar.com under Help/Developer Zone/Downloads. SDK examples and code documentation is available on the https://sdk.sportradar.com.

Logging

To more easily find the log entry associated with a specific action that occurred within the SDK, the logs are split into several files.

- Feed log: contains log entries for all messages send to or received from the MTS
- Rest log: contains log entries for all messages send or received from API calls[1]
- Cache log: contains log entries for all messages related to internal cache(s)
- Execution log: contain log entries for all important actions and all error / warning conditions which occur within the SDK
- Client interaction log: logs the interaction between the user code and the SDK
- Statistics log: contains periodically written statistic information

To enable the SDK logging the logging framework used by the SDK has to be properly configured. The configuration is done by call of the following method:

```
SdkLoggerFactory.Configure(new FileInfo("config_file_path"));
```

The default configuration file can be obtained from the SDK example project available on the SDK site.
Logs are used by the support team, so it is recommended to send them along with any issue related e-mails.

[1] REST API calls are made only when building ticket with UnifiedFeed selections

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

Configuration

The configuration needed by the SDK must be provided via the app.config file, which must contain the following section:

```
<mtsSdkSection
      username="username",
      password="password",
      host="integration-mts.betradar.com",
      vhost="/vhost",
      useSsl="false",
      node="3",
      bookmakerId="1",
      limitId="1",
      currency="EUR",
      channel="Internet",
      accessToken="your_uf_access_token",
      provideAdditionalMarketSpecifiers="true",
      port="5671",
      exclusiveConsumer="true" />
```

Required attributes:

- username: Username used to connect to the AMQP broker. Betradar provides this value.
- password: Password used to connect to the AMQP broker. Betradar provides this value.
- host: The hostname of the AMQP broker. Please use the following hostnames unless the integration team provides different ones.
    - Integration environment: integration-mts.betradar.com
    - Production environment: tradinggate.betradar.com

Optional attributes:

- vhost: The name of the virtual host configured on the AMQP broker. If the value is not specified, the value of '/username' attribute is used as virtual host.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

- `useSsl`: The value specifying whether SSL will be used when connecting to the broker. Default value is `true`.
- `node`: This value is used to filter MTS responses which were produced as responses to requests send by different SDK instances. In most configurations each SDK should use different node value. Default value is `1`.
- `bookmakerId`: When provided, it is used as the default value for the `BookmakerId` on the ticket. The value can be overridden when building the ticket. Betradar provides this value.
- `limitId`: When provided, it is used as the default value for the `LimitId` property on the ticket. The value can be overridden when building the ticket. Betradar provides the set of available values.
- `currency`: When provided, it is used as the default value for the Currency property on the ticket. The value must comply with the ISO 4217 standard.
- `channel`: When provided, it is used as the default value for the `SenderChannel` property on the ticket. Value must be one of the `SenderChannel` enumeration members.
- `accessToken`: When selections are build using UnifiedOdds ids, the accessToken is used to access sports API. Also ensure that server running the sdk is whitelisted on api.betradar.com. Betradar provides this value.
- `provideAdditionalMarketSpecifiers`: This value is used to indicate if the sdk should add market specifiers for specific markets. Only used when building selection using UnifiedOdds ids. If this is set to true and the user uses UOF markets, when there are special cases (market 215, or $score in SOV/SBV template), sdk automatically tries to add appropriate specifier; if set to false, user will need to add this manually.
- `port`: Port should be chosen through the useSsl property. Manually setting port number should be used only when non-default port is required.
- exclusiveConsumer: The value specifying whether the rabbit consumer channel should be exclusive. Default value is `true`.

For more information about the ticket properties please refer to the MTS_Ticket_Integration document.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

Obtaining the SDK

The SDK is provided as a code library (Sportradar.MTS.SDK.dll), which is available on the SDK site and via the NuGet package manager. The usage of NuGet package manager is recommended, since it supports update notifications and makes it easier to obtain new releases of the SDK.

SDK setup and teardown

The SDK is setup by the following steps:

- Creating an instance of the `MtsSdk` class.
- Attaching to the following events exposed by the `MtsSdk` type.
  - SendTicketFailed – raised if the ticket could not be send to the AMQP broker within the predefined timeout (15 seconds). This usually indicates an Internet connection or firewall issues.
  - TicketResponseReceived – occurs when a response to ticket placement or ticket cancellation request from the MTS is received.
  - UnparsableTicketResponseReceived – occurs when the response from the MTS cannot be deserialized. This usually indicates that a deprecated version of the SDK is being used.
  - TicketResponseTimedOut - event to notify user if the ticket response did not arrive in timely fashion (when sending in non-blocking mode)
- Opening the created `MtsSdk` instance.

These steps can be performed by the following code:

```
var config = MtsSdk.GetConfiguration();
var mtsSdk = new MtsSdk(config);
mtsSdk.SendTicketFailed += OnSendTicketFailed;
mtsSdk.TicketResponseReceived += OnTicketResponseReceived;
mtsSdk.UnparsableTicketResponseReceived += OnUnparsableTicketResponseReceived;
mtsSdk. TicketResponseTimedOut += OnTicketResponseTimedOut;
mtsSdk.Open();
```


Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

For more information on how to handle the events please refer to the SDK examples and/or the SDK code documentation.

Once the initialized `MtsSdk` instance is no longer needed, it must be teardown in order to release resources held by it. It is also recommended to detach from events before disposing the instance. This can be accomplished by the following code:

```
mtsSdk.SendTicketFailed -= OnSendTicketFailed;
mtsSdk.TicketResponseReceived -= OnTicketResponseReceived;
mtsSdk.UnparsableTicketResponseReceived -= OnUnparsableTicketResponseReceived;
mtsSdk. TicketResponseTimedOut -= OnTicketResponseTimedOut;
mtsSdk.Close();
```


Sportradar AG
www.sportradar.com

Feldlistrasse 2
CH-9000 St. Gallen
Switzerland

P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com

Company Identification
No:CHE-113.075.404

Building ticket instances

The SDK uses a "builder pattern" to simplify the process of creating new ticket instances. Below is the list of most noticeable builders.

- `TicketBuilder`: A root builder used as a starting point when building tickets.
- `SenderBuilder`: Used to specify the information about a ticket sender (bookmaker).
- `EndCustomerBuilder`: Used to build `EndCustomer` instances, representing the punter associated with the ticket. This information is part of the send element.
- `BetBuilder`: Used to build bet instances, which is part of the ticket. Each ticket must contain at least one bet.
- `SelectionBuilder`: Used to build selection instances, which are parts of bet. Each bet must contain at least one selection.

Below is a code snippet, which builds a ticket containing the mandatory information. Please note that some information from the configuration gets automatically applied to the ticket, so changing the configuration can make the snippet below produce an incomplete ticket. For more information refer to configuration section of this document and to MTS_Ticket_Integration document.
Builders can be obtained on mtsSdk instance through `BuilderFactory`.

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

```
var _builderFactory = _mtsSdk.BuilderFactory;
var ticket = _builderFactory.CreateTicketBuilder()
    .SetTicketId("ticketId")
    .SetSender(_builderFactory.CreateSenderBuilder()
        .SetCurrency("EUR")
        .SetEndCustomer(_builderFactory.CreateEndCustomerBuilder()
            .SetId("customerClientId")
            .SetConfidence(1)
            .SetIp(IPAddress.Loopback)
            .SetLanguageId("en")
            .Build())
        .Build())
    .AddBet(_builderFactory.CreateBetBuilder()
        .SetBetId("betId")
        .SetBetBonus(1)
        .SetStake(1, StakeType.Total)
        .AddSelectedSystem(1)
        .AddSelection(_builderFactory.CreateSelectionBuilder()
            .SetEventId(1)
            .SetId("selectionId")
            .SetOdds(11000)
            .Build())
        .Build())
    .BuildTicket();
```


Sportradar AG
www.sportradar.com

Feldlistrasse 2
CH-9000 St. Gallen
Switzerland

P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com

Company Identification
No:CHE-113.075.404

Sending tickets to MTS

SDK supports two ways of sending tickets to the MTS. The recommended way is to use the non-blocking mode. Non-blocking indicates the execution of the current thread is not blocked after the ticket is send and the response from MTS (TicketResponseReceived event) is process in another thread. To send a ticket in a non-blocking mode, the following line can be used: mtsSdk.SendTicket(ticket);
For this mode event TicketResponseTimedOut is also available to notify user if the ticket response did not arrive in timely fashion.

When sending the ticket in the blocking mode, the current thread is blocked until a response from MTS is received or a timeout occurs (15 seconds). When using the blocking mode the TicketResponseReceived event for that ticket is never raised. Ticket can be send in a blocking mode using the following line:

```
var ticketResponse = mtsSdk.SendTicketBlocking(ticket);
```


Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404

Tips and tricks

Building selection instances

The SDK supports markets used by three Betradar feeds – LO (Live Odds), LCoO (Live Cycle of Odds) and UF (Unified Feed) implemented by different methods on the `SelectionBuilder` type.

- `SetId(string id);`
  This method should be used when building string representations of the market identifiers directly (without the help from the SDK).
- `SetIdLo(int type, int subType, string sov, string selectionId);`
  This method should be used when building market identifiers from information provided by the LO feed.
- `SetIdLcoo(int type, int sportId, string sov, string selectionId);`
  This method should be used when building market identifiers from information provided by the LCoO feed.
- `SetIdUof(Product product, URN sportId, int marketId, string selectionId, IDictionary<string, string> specifiers, IReadOnlyDictionary<string, object> sportEventStatus);`
  This method should be used when building market identifiers from information provided by the UF feed. Note: this method will throw if accessToken is not provided.
  Method parameter sportEventStatus needs the following keys:
  - HomeScore (home_score in sport event status)
  - AwayScore (away_score in sport event status)
  - Server (current_server in sport event status)

  If you are using UnifiedFeed sdk the map with the correct keys may be obtained:
  var sportEventStatusProperties = sportEvent.Status.Properties;

Sportradar AG
www.sportradar.com
Feldlistrasse 2
CH-9000 St. Gallen
Switzerland
P: +41 71 517 72 00
F: +41 71 517 72 99
E: sales@sportradar.com
Company Identification
No:CHE-113.075.404