MBS SDK .NET Integration Guide

February 2024

sportradar

SPORTS TECHNOLOGY. REIMAGINED.

1 Document history

Version	Author	Date	Comments
1.0	Dejan Pavšek	2024-02-16	Initial version

2 Table of contents

3 Getting started

MBS SDK is a client library that facilitates the integration of a client's system with Sportradar's Managed Betting Services (MBS). The library simplifies the integration by reducing the effort required for code development, testing and maintenance.

Managed Betting Services is a suite of services designed for betting operators and lotteries to fully manage their sportsbooks. The services include integrated trading tools, analytics, risk management and access to AI-driven insights into betting operations.

Before you start using the SDK, read the MTS documentation found at docs.betradar.com, chapter Managed Trading Services (MTS) / MTS – Ticket Integration Manual.

SDK examples and code documentation are available at https://sdk.sportradar.com.

4 Obtaining the SDK

The SDK can be found in the following Nuget repository:
https://www.nuget.org/packages/Sportradar.Mbs.Sdk

package=Sportradar.Mbs.Sdk

version=0.9.0 (always download the latest version, current stable version is 0.9)

5 SDK setup and use

The SDK can be set up and used by following these steps:

- Create the SDK configuration by creating an instance of the `MbsSdkConfig` class.
- Create an instance of the `MbsSdk` class (based on the required configuration).
- Connect the created SDK instance to the MBS server.
- Use the SDK by composing transactions, sending them to MBS and receiving responses from MBS. This is the actual interaction between the client's system and MBS which is required to use MBS services.
- Tear down the SDK instance when it is no longer needed.

5.1 Configuration

Required attributes:

Values of the required attributes must be provided by the Sportradar integration team. See also MTS - Connectivity (v3.0).

Create a config object that will be used when building the SDK:

```
MbsSdkConfig config = new MbsSdkConfig(<wsServer>,<authServer>,
<authClientId>,<authClientSecret>,<authAudience>,<operatorId>);
```

List of required attributes:

`WsServer`

- URI of the WebSocket server to connect to.

`AuthServer`

- URI of the authorization server to send a token request to. The token request is an https POST request and should be made before establishing a WebSocket connection.

`AuthClientId`

- Client ID to include in the token request.

`AuthClientSecret`

- Client secret to include in the token request.

`AuthAudience`

- Audience attribute to use in the token request.

`OperatorId`

- Operator ID that identifies the client and should be included in each ticket. The value must be provided by the Sportradar integration team.

Optional attributes:

The values of the optional attributes should not be changed unless there is a specific need to do so.

`ProtocolMaxSendBufferSize;`

- Specifies the maximum number of requests that have been sent but whose responses have not yet been received. The default value is 1000. When this threshold is reached, any new request attempt is rejected by the SDK (the ProtocolSendBufferFullException is thrown). Any new request attempts will succeed only after some buffer space is released.

`ProtocolReceiveResponseTimeout;`

- Specifies the maximum time the SDK will wait for response to be received. After the timeout has been reached, the ProtocolTimeoutException is thrown. If MbsSdk is configured to retry operations (protocolRetryCount), the exception is suppressed and request operation is repeated.

`ProtocolRetryCount;`

- Specifies the number of retries to send a request after the response timeout has expired. The default value is 0 (zero).

`UnhandledExceptionHandler;`

- This method manages exceptions that may occur within the SDK that are not otherwise handled. The exceptions are usually logged for later analysis. You must implement your own logging mechanism.

In the rare case that you need to set an optional attribute, you can do so by calling the appropriate setter method. For example, the following call sets the protocolRertyCount attribute:

```
config.ProtocolRetryCount = value;
```

5.2 Creating and connecting the SDK

Create an SDK instance which will make use of the previously created configuration:

```
MbsSdk mbsSdk = new MbsSdk(config);
```

The below call will create a WebSocket session to the MBS's WebSocket server:

```
mbsSdk.Connect();
```

When connect() is invoked, the provided configuration is used for the first time. If SDK is not able to successfully establish WebSocket connection with a server, an exception is thrown. See SDK troubleshooting to identify the issue.

5.3 Composing transactions

Communication between the client's system and MBS takes place through the exchange of transactions. Transactions can be of various types. For example, when dealing with risk management, transactions will contain the punter's betting data (commonly known as tickets) and responses will contain, among other data, the ticket acceptance recommendation. Other types of transactions may include data about online gaming events and their outcomes, or data about actions that affect the punter's monetary balance (transfers, withdrawals, etc).

5.4 Sending transactions

MbsSdk supports various operations. The operations are grouped in interfaces called protocols. An example of such a protocol is Ticket Protocol.

All operations are accessible through the `mbsSdk` instance. An example of how to use the Ticket Protocol is shown below:

```
ITicketProtocol protocol = mbsSdk.TicketProtocol;
```

5.4.1. Ticket protocol

TicketProtocol is used to send requests to and receive responses from MBS.

An example of how to access the TicketProtocol methods is shown below.

```
TicketResponse response = await
mbsSdk.TicketProtocol.SendTicketAsync(request);
```

The following TicketProtocol methods are available:

`SendTicketAsync (TicketRequest request);`

- This method sends a ticket to MTS and returns a ticket response from MTS. For information on the content of the "request" argument, see the Ticket Placement Request (v3.0) documentation. For information on what to expect in the response, see the Ticket Placement Response (v3.0) documentation.

`SendTicketAckAsync (TicketAckRequest request);`

- This method sends a ticket acknowledgement message to MTS and returns an acknowledgement response from MTS. For information on the content of the "request" argument, see the Acknowledgement Messages (v3.0) documentation. For the response, see Acknowledgement Reply Messages (v3.0)

`SendCancelAsync (CancelRequest request);`

- This method sends a ticket cancellation request to MTS and returns a cancellation response from MTS. For information on the content of the "request" argument, see the Cancellation Request (v3.0) documentation. For the response, see Cancellation Response (v3.0)

`SendCancelAckAsync (CancelAckRequest request);`

- This method sends a cancellation acknowledgement message to MTS and returns an acknowledgement response from MTS. For information on the content of the "request" argument, see the Acknowledgement Messages (v3.0) documentation. For the response, see Acknowledgement Reply Messages (v3.0)

`SendCashoutAsync (CashoutRequest request);`

- This method sends a ticket cash-out request to MTS and returns a cash-out response from MTS. For information on the content of the "request" argument, see the Cash-out Request (v3.0) documentation. For the response, see Cash-out Response (v3.0)

`SendCashoutAckAsync (CashoutAckRequest request);`

- This method sends a cash-out acknowledgement message to MTS and returns an acknowledgement response from MTS. For information on the content of the "request" argument, see the Acknowledgement Messages (v3.0) documentation. For the response, see Acknowledgement Reply Messages (v3.0)

`SendExtSettlementAsync (ExtSettlementRequest request);`

- This method sends an external settlement request to MTS and returns an external settlement response from MTS. For information on the content of the "request" argument, see the External Settlement Request (v3.0) documentation. For the response, see External Settlement Response (v3.0)

`SendExtSettlementAckAsync (ExtSettlementAckRequest request);`

- This method sends an external settlement acknowledgement message to MTS and returns an acknowledgement response from MTS. For information on the content of the "request" argument, see the Acknowledgement Messages (v3.0) documentation. For the response, see Acknowledgement Reply Messages (v3.0)

5.5 SDK troubleshooting

In case of an error, MbsSdk delegates exception to the hosting application. If an exception occurred as a result of operation execution, then exception is thrown as a result of the invoked method/operation. If the exception cannot be linked to a particular operation, then the exception is passed to the application through unhandledExceptionHandler (see config).

MbsSdk exceptions list:

`AuthTokenFailureException:`

- Thrown if the SDK could not obtain the authentication token (misconfiguration or network issues).

`ProtocolMessageTooBigException:`

- Thrown if JSON serialized message size exceeds 128 kB, which is the largest allowed message size.

`ProtocolSendBufferFullException:`

- Thrown when buffer capacity is reached. See protocolMaxSendBufferSize config (Configuration section).

`ProtocolSendFailedException:`

- Thrown when the SDK could not send the message to the server (misconfiguration or network issues).

`ProtocolTimeoutException:`

- Thrown if no response is received in a configured time (protocolReceiveResponseTimeout config in the Configuration section).

`SdkNotConnectedException:`

- Thrown if SDK is not connected or is closed.

`ServerErrorResponseException:`

- Thrown if the server returns an error response.

`WebSocketConnectionException:`

- Thrown if a WebSocket connection could not be established (misconfiguration or network issues).

5.6 Closing the SDK

Once the SDK instance is no longer needed, it must be torn down to release the resources it holds.

```
mbsSdk.Dispose();
```

END OF DOCUMENT