MTS Java SDK developer guide

August 2016

betradar
driven by facts

Table of Contents

1 Introduction

To make **M**anaged **T**rading **S**ervice integration as quick and easy as possible **S**oftware **D**evelopment **K**it was developed.

SDK exposes MTS feed interface in a more user-friendly way and isolates the client from having to do proper connection handling, message sending, JSON feed parsing and dispatching.

This document contains info about Java implementation and usage of the SDK.

2 SDK global overview

MTS supports AMQP and UDP protocol. SDK uses AMQP. While SDK tries to hide as many details as possible, clients should still be familiar with AMQP protocol. Basics can be read at https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

We also recommend reading “MTS integration with AMQP” and “MTS Ticket Integration” documents.

3 Using the SDK

In this document pseudo code is used to give you basic understanding, for actual usage examples please check the SDK example project.

First you need to create a new instance of the SDK:

```
MtsSdkApi mtsSdk = new MtsSdk();
```

This creates the SDK but does not start anything yet.

You need to call open method first. There are 3 variants.

First option:

```
MtsSdkApi.open()
```

This will initialize the SDK using the “mts-sdk.properties” file located in your resources folder.

Second option:

```
MtsSdkApi.open(String filePath)
```

This will initialize the SDK using the file found in the path you specified. It can be absolute or relative path. Settings format in the file must be the same as in “mts-sdk.properties”.

Third option:

```
MtsSdkApi.open(Properties properties)
```

This will initialize the SDK using passed Properties instance where key values are same as they would be in “mts-sdk.properties” file.

It is possible to create multiple SDK instances using different settings (by using last two open methods) which can prove useful if you want to run multiple clients on the same machine.

After SDK is initialized you can create one or multiple messages senders. There are four possible senders.

3.1 Ticket sender

The main sender is TicketSender which is used to send tickets to the MTS. You obtain a new instance by calling

```
TicketSender ticketSender =
MtsSdkApi.getTicketSender(TicketResponseListener responseListener)
```

where ***responseListener*** is your implementation of ticket response listener. In TicketResponseListener events will be triggered when your ticket publish succeeded/failed and when you receive a ticket acceptance response from the MTS.

Ticket sender enables two ways of sending tickets. First one is by using

```
TicketSender.send(Ticket ticket)
```

This will send ticket asynchronously and trigger response in your response listener. This is the recommended way of sending tickets.

We also developed second option for clients who are more used to synchronous sending.

```
TicketResponse response = TicketSender.sendBlocking(Ticket ticket) throws
ResponseTimeoutException
```

This method will block calling thread until ticket acceptance reply is received or throw ResponseTimeoutException in case reply is not received in time. This method returns TicketResponse object directly and no responseReceived will be triggered in your implementation of TicketResponseListener.

No matter which method you use you need to generate a Ticket instance first which will be send to the MTS.

This is done over TicketBuilder class. MTS ticket version 1.X uses object hierarchy which we concealed and made the builder flat with only selection as a sub object on the ticket. Below is an example of ticket generation

```
Ticket ticket =  TicketSender.newBuilder()
        .setBookmakerId(1)
        .setTicketId(“ticket id”)
        .setLimitId(2)
        .setChannelId(SourceChannel.INTERNET)
        .setDeviceId("device1")
        .setEndCustomerId("User123456")
        .setEndCustomerIp("1.3.3.7")
        .setLanguageId("EN")
        .setCurrency("EUR")
        .setStake(5.0)
        .setSystem(0)
        .setBonusWin(10.2)
        .addSelection()
        .setLine(LineType.PREMATCH)
        .setMarket("lcoo:10/1/*/1")
        .setMatch(9569629)
        .setOdd(1.1)
        .buildSelection()
        .build()
```

To add selections you call

```
TicketBuilder.addSelection()
```

which returns selection builder where you can set selection properties. When you are done with selection you call

```
SelectionBuilder.buildSelection()
```

which builds the selection, adds the selection to parent and returns parent ticket builder.

3.2 Ticket cancellation sender

When you want to cancel an accepted ticket you use TicketCancelSender. Only asynchronous sending is supported.

You obtain a new instance by calling

```
TicketCancelSender ticketSender = MtsSdkApi.getTicketCancelSender(
TicketCancelResponseListener responseListener)
```

where ***responseListener*** is your implementation of ticket cancellation response listener. In TicketCancelResponseListener events will be triggered when your ticket cancellation publish succeeded/failed and when you receive a ticket cancellation response from the MTS.

Example of ticket cancellation generation

```
TicketCancel ticketCancel = TicketCancelSender.newBuilder()
        .setCancelMessageId("messageID")
        .setTicketId("ticket id")
        .setBookmakerId(1)
        .setCancellationReason(101)
        .setReasonMessage("customer cancelled ticket")
        .build()
```

After you have constructed a TicketCancel instance you send it using TicketSender.

```
TicketCancelSender.send(TicketCancel ticketCancel)
```

3.3 Ticket acknowledgement sender

Clients who want to send ticket acceptance acknowledgement can do that over the TicketAcknowledgmentSender class. As acknowledgment has no reply your listener needs to implement only publish failure/success handler methods.

Sender creation:

```
TicketAcknowledgmentSender ticketAckSender =
MtsSdkApi.getTicketAcknowledgmentSender (
TicketAcknowledgmentResponseListener responseListener)
```

Message creation:

```
TicketAcknowledgment ticketAcknowledgment =
TicketAcknowledgmentSender.newBuilder()
        .setTicketId(“ticket id”)
        .setAckStatus(TicketAckStatus.ACCEPTED)
        .setBookmakerId(1)
        .setSourceCode(100)
        .build();
```

Message sending:

```
TicketAcknowledgmentSender.send(TicketAcknowledgment  ticketAcknowledgment);
```

3.4 Ticket cancel acknowledgement sender

Ticket cancellation acknowledgment sender is very similar to ticket acceptance acknowledgment sender.

Sender creation:

```
TicketCancelAcknowledgmentSender ticketCancelAckSender =
MtsSdkApi.getTicketCancelAcknowledgmentSender (
TicketCancelAcknowledgmentResponseListener responseListener)
```

Message creation:

```
TicketCancelAcknowledgment ticketCancelAcknowledgment =
TicketCancelAcknowledgmentSender.newBuilder()
        .setTicketId(“ticket id”)
        .setAckStatus(TicketCancelAckStatus.CANCELLED)
        .setBookmakerId(1)
        .setSourceCode(101)
        .build();
```

Message sending:

```
TicketCancelAcknowledgmentSender.send(TicketCancelAcknowledgment
ticketCancelAcknowledgment);
```

4 SDK settings

Setting	Mandatory	Default	Description
mts.sdk.vhost = /vhost	true	/	AMQP virtual host
mts.sdk.username = user	true	/	AMQP username
mts.sdk.password = pass	true	/	AMQP password
mts.sdk.test = false	false	true	Which environment to connect to
mts.sdk.ssl = true	false	true	Use SSL for communication
mts.sdk.hostname= 127.0.0.1	false	integration-mts.betradar.com if test=true, else tradinggate.betradar.com	Hostname
mts.sdk.port = 1337	false	5671 if ssl=true, else 5672	Port
mts.sdk.node = 1	false	1	Node id to be used when creating routing key
mts.sdk. ticket_response_timeout = 15000	false	15000	How long (in ms) to wait while waiting for ticket response. Used only with sendBlocking method
mts.sdk. messages_per_second = 40	false	40	Max message rate allowed to be send to the MTS for each sender