Trading API

Table of Contents

Get started . . . 1
- Overview . . . 1
- Requirements . . . 1
- License . . . 1
- Communication . . . 1
- Security . . . 2
- Configuration . . . 3
- Session TAN . . . 5

Usage of the Trading API . . . 5
- Source code . . . 5
- Initialization . . . 6
- Services and functions definition . . . 8
- Create services . . . 8
- Functional programming . . . 9
- Pull and push functions . . . 10
  - Pull requests . . . 10
  - Push requests . . . 11
  - Push subscriptions . . . 12

Services and functions . . . 13
- Access service . . . 13
  - Client validation . . . 13
  - Client invalidation . . . 14
- Account service . . . 14
  - Get list of trading accounts . . . 15
  - Stream account information changes . . . 16
  - Stream account transactions changes . . . 18
- Stock exchange service . . . 20
  - Get information about all stock exchanges . . . 21
  - Get information about specific stock exchange . . . 22
- Depot service . . . 22
  - Stream depot changes . . . 22
  - Update depot . . . 24
- Security service . . . 25
  - Securities access . . . 25
  - Get security information . . . 26
  - Stream market data information . . . 27
  - Stream orderbook data . . . 30
  - Stream currency rate . . . 32

Get security historic data . . . 34
Order service . . . 35
Order types and parameters . . . 35
Stream orders . . . 38
Update orders . . . 41
Get securities quotes . . . 41
Accept quote . . . 44
Add order . . . 46
Change order . . . 48
Cancel order . . . 49
Activate order . . . 49
Deactivate order . . . 50
Order costs . . . 51
Errors . . . 57
Objects and types description . . . 58
AcceptQuoteRequest . . . 58
AccessTokenRequest . . . 59
ActivateOrderRequest . . . 59
AggregatedCosts . . . 60
AddOrderRequest . . . 61
CancelOrderRequest . . . 62
CashQuotation . . . 62
CategoryCost . . . 63
ChangeOrderRequest . . . 63
CurrencyRateReply . . . 64
CurrencyRateRequest . . . 64
Date . . . 64
DeactivateOrderRequest . . . 65
DepotEntries . . . 65
DepotEntry . . . 65
DepotPosition . . . 66
DetailCost . . . 66
Empty . . . 67
Error . . . 67
LimitToken . . . 67
LoginReply . . . 67
LoginRequest . . . 67
LogoutReply . . . 68
LogoutRequest . . . 68
Order . . . 68
OrderBookEntry . . . 69

OrderCosts . . . 69
OrderModel . . . 69
OrderReply . . . 70
Orders . . . 70
OrderStatus . . . 70
OrderSupplement . . . 71
OrderType . . . 71
PriceEntry . . . 72
QuoteEntry . . . 72
QuoteReply . . . 72
QuoteRequest . . . 73
SecurityChangedField . . . 73
SecurityClass . . . 74
SecurityCode . . . 75
SecurityCodeType . . . 75
SecurityInfoReply . . . 75
SecurityInfoRequest . . . 76
SecurityMarketDataReply . . . 76
SecurityMarketDataRequest . . . 77
SecurityOrderBookReply . . . 77
SecurityOrderBookRequest . . . 78
SecurityPriceHistoryReply . . . 78
SecurityPriceHistoryRequest . . . 78
SecurityStockExchangeInfo . . . 79
SecurityWithStockExchange . . . 79
ShortMode . . . 82
StockExchange . . . 83
StockExchangeDescription . . . 83
StockExchangeDescriptions . . . 83
StockExchangeInfo . . . 83
StockExchangeRequest . . . 83
TimeResolution . . . 84
Timestamp . . . 84
TradingAccount . . . 86
TradingAccountInformation . . . 86
TradingAccountRequest . . . 86
TradingAccounts . . . 87
TradingAccountTransactions . . . 87
TradingPhase . . . 87
TradingPossibility . . . 87
TradingState . . . 88

TrailingNotation . . . 88
Transaction . . . 88
UnitNote . . . 89
Validation . . . 89
Questions and Answers . . . 89

Version: 0.2.1-SNAPSHOT

Get started

Overview

Trading API (TAPI) is a part of the standard Consorsbank trading application ActiveTrader / ActiveTrader Pro. It allowes to write user managed application for the specific trading activities. TAPI support pull / push requests and based on the Google Remote Procedure Call (GRPC) technology.

With help of the TAPI you have an access to the accounts, orders, depots and market data information.

Important

This documentation is the description of the early version of Trading API. It's not final and it's subject to change. With the lookback to the new BaFin regulation it's especially important to understand that the ordering part of the TAPI can be partially changed to fullfil new law directives.

Additional information and discussions can be found there. Please contact to the support team to get access to this part of the community.

Requirements

- Active Trader / Active Trader Pro with Java 8 64 bits, minimum 8Gb RAM
- Client Java 8 or higher 64 bits

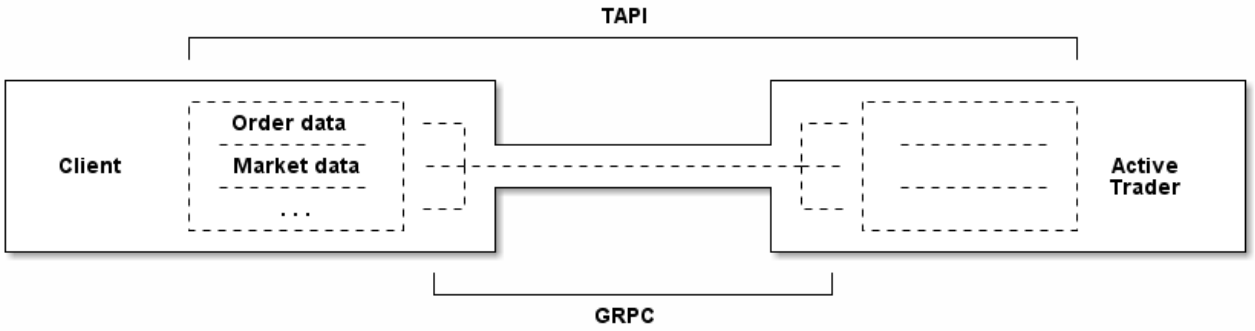
License

Trading-API is licensed under Apache 2.0 license.

The license and notice texts can be found in the delivered **LICENSE** and **NOTICE.TXT** files.

Communication

GRPC technology uses one connection to transfer all necessary data beetwen client and ActiveTrader. As developer you don't need to care about communication. All low level communication, security and converting functions takes TAPI engine. You can concentrate on the high level functions and bussines logic of the application.

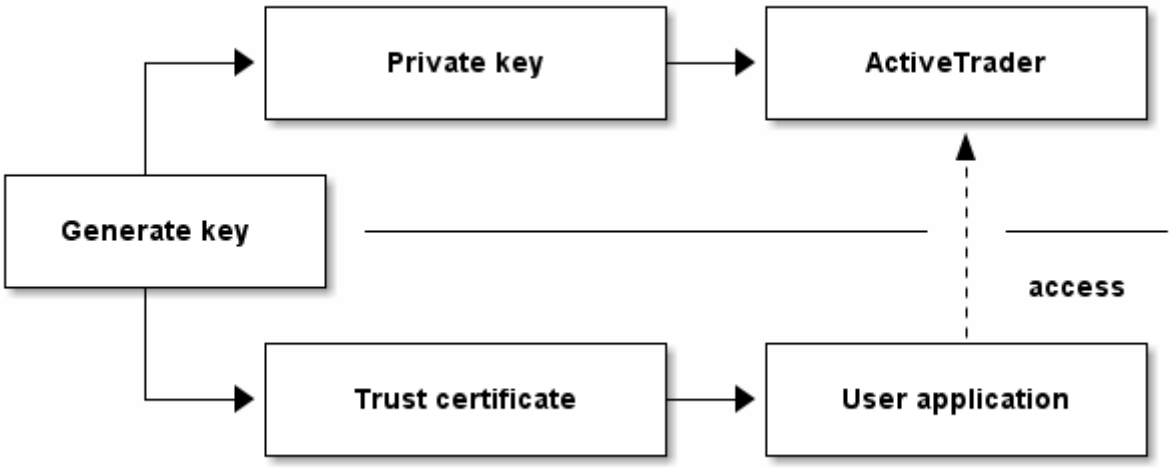


Security

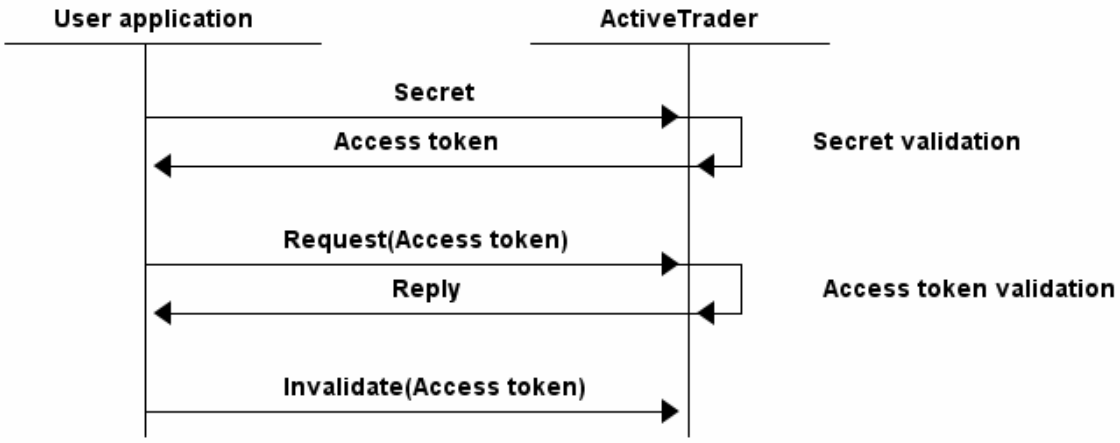
TAPI uses HTTP/2 protocol with RSA protection. RSA self signed certificate can be generated directly in the ActiveTrader application. Typical use case is to use user application from same PC, but it's also possible to connect from the remote machine.

Important

To accept self signed certificate from the user application is necessary to use trusted part of the RSA keys (Trust Certificate) by the connection initialization in the client application. This certificate can be exported from the ActiveTrading application.



To provide users access control can be used **secret** string. That's an equivalent of password. Secret is defined in the Active Trader and used only for validation. User application sends secret to the ActiveTrader. If the validation in the ActiveTrader is passed then ActiveTrader generates a session **access token** and returns it to the user application. With this token is possible to execute other requests. After end of processing the token should be invalidated.

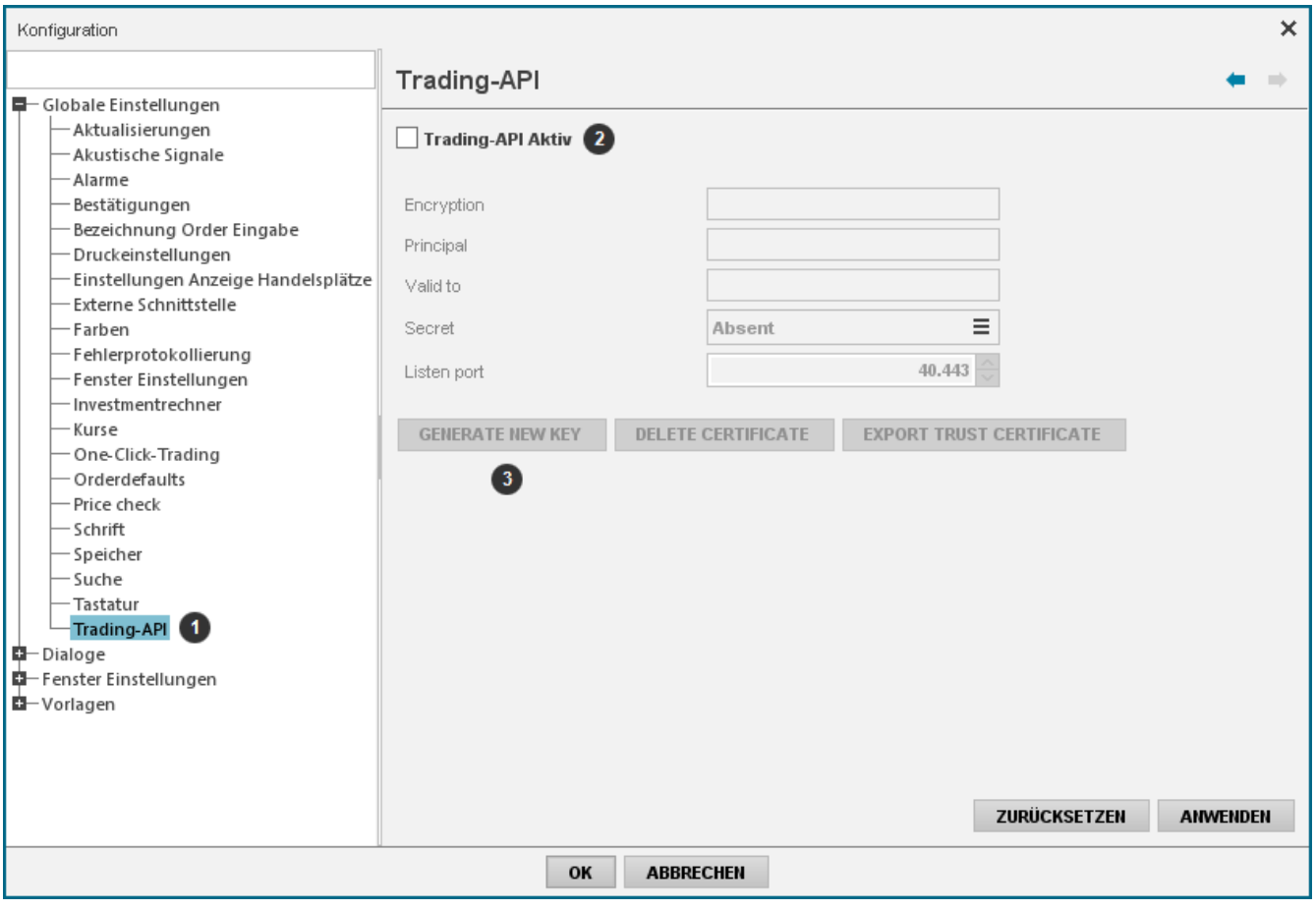


Important

We don't keep **secret** data in the configuration. Only double MD5 hash of the secret + solt is stored in the configuration. As result we can't restore the secret. For more information see Defense against rainbow tables

Configuration

To activate TAPI is need to complete next steps:



① Select Trading-API settings in the configuration

②

Activate TAPI checkbox

③ Optional: if key exists, press [GENERATE NEW KEY] button

In the opened dialog

① Enter or correct key settings

② Press [GENERATE NEW KEY] button and wait unil new key is generated

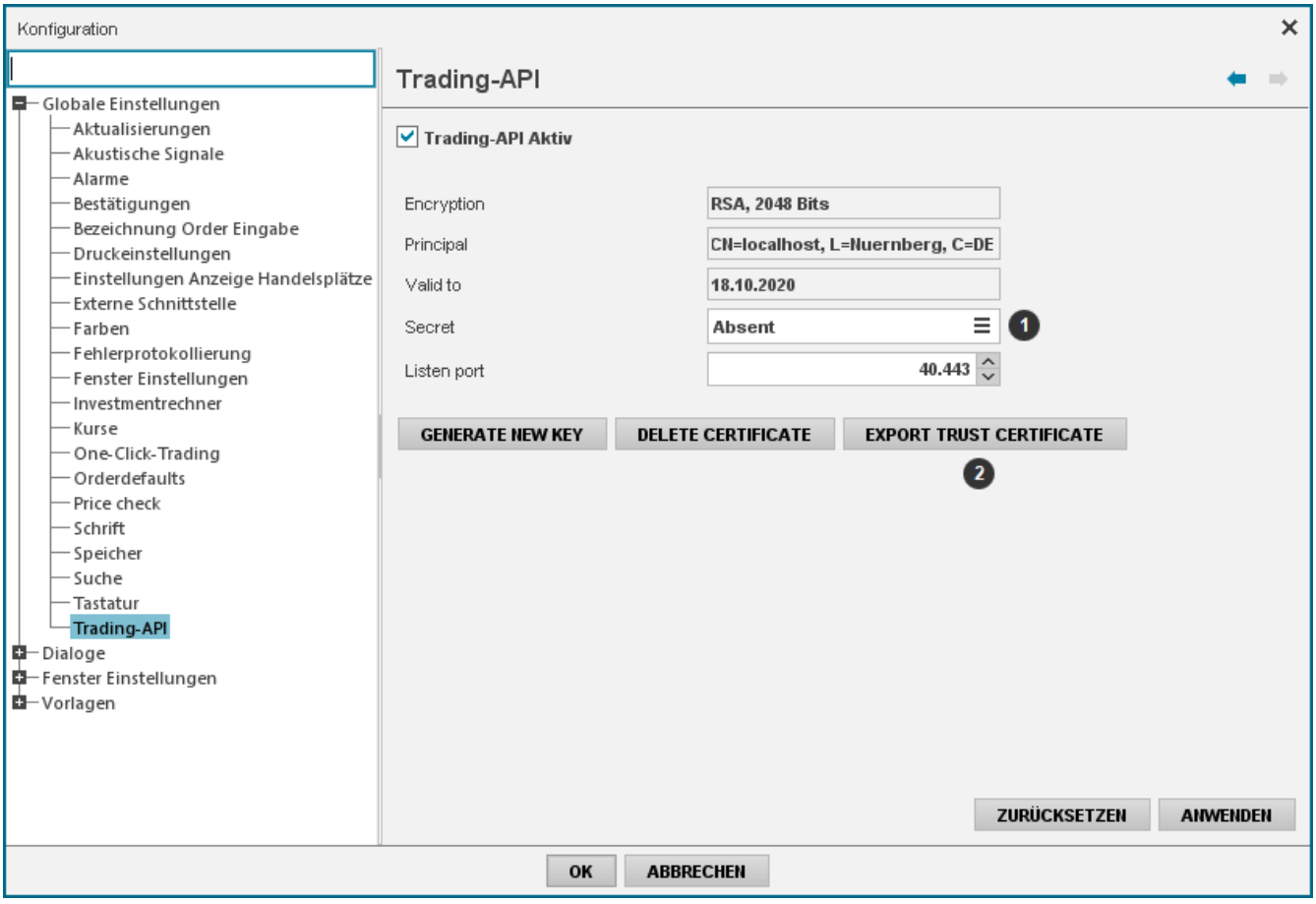
③ Press [OK] button to accept new settings

After confirmation of the new settings TAPI will be activated.

Warning

In some cases if any initialization errors are appear, the TAPI activation indication in the status bar of the application will be red. To check activation problems move mouse cursor over this indication and check tooltip for the error.

Before to start using client application, it's need to export trusted certificate from ActiveTrader.



① Set secret for an access control. Longer string is better.

② Press [EXPORT TRUST CERTIFICATE] and store file (for example with name: **roots.pem**) to the location where client application can use it.

Session TAN

Some operations of the TAPI can be used without TAN. All activities with the necessity of the autorisation are need to use of the session TAN.

Important

Do not forget to activate session TAN. It's not possible to use individual TAN's with TAPI

Usage of the Trading API

Source code

An example aplication is delivered with precompiled GRPC libraries. If you want to generate source code by youself you can use protobuf files from **protobuf** directory. Please refer to the Java Generated Code Reference.

Initialization

To initialize client application and connect to the ActiveTrader with help of TAPI you have to process next steps:

Important

All examples are avialable as a sorce code and can be found in the specific folders.

Input point is the file **TestClient.java**

To start examples you have to put into start parameters full path to the trust certificate and secret. For example: "C:\PathToTrustCert\roots.pem 1234567"

Initialize the root GRPC engine with trusted certificate, exported from configuration dialog of the ActiveTrader.

Example Java: Initaialize GRPC

```
private String accessToken = "";

/**
 * Test client.
 */
public static void main(String[] args) throws Exception {
  if (args.length<2) {
    System.err.println("Please generate key in AT and "
        + "put location of the trust certificate and secret as arguments");
    System.exit(-1);
  }
  File caFile = new File(args[0]); ①

  if (!caFile.isFile()) { ②
    System.err.println("File: "+caFile+" does not exist");
    System.exit(-1);
  }

  String server = "localhost"; ③
  int port = 40443;

  System.err.println("Connecting... "+server+":"+port);
  try (TestClient client = new TestClient(server, port, caFile, args[1])) { ④

    // ...

  } catch (Exception e) {
    e.printStackTrace();
  }
}

public TestClient(String host, int port, File caFile, String secret) throws
SSLException {
  this(NettyChannelBuilder.forAddress(host, port)
      .negotiationType(NegotiationType.TLS)
      .sslContext(GrpcSslContexts.forClient().trustManager(caFile).build())
      .build());
  accessToken = login(secret);
  if (accessToken.isEmpty()) {
    // Error
  }
}
```

① Get trusted certificate location as a parameter of the command line

② Check if a file of the cetrificates exists

③ Define connection settings

④ Initialize GRPC connection

Services and functions definition

After an initialization is GRPC ready and it can be used for the building service stubs to operate with TAPI services. Stubs are sets of service functions with help of then is possible to access data and execute user actions. They are very similar to the normal programming functions with small differences.

- Each remote function can have one (and only one) input parameter.
- Each remote function can have one (and only one) result parameter.
- The function can send data asynchronically

Functions can be simple (one request → one reply), server sends events (one request from client → many replies from server), client sends events (many requests from client → one reply from server) or server and clients sends events (many requests from client → many replies from server). TAPI uses only first and second types of functions. First type of the function used for the simple user activity (pull) and second type for subcriptions or streaming (push).

Services and functions are defined in the special files as protobuf protocol v.3

Example Protobuf: RPC definition

```
service ServiceName { ①
  rpc SimpleFunction(ClientParameterType1) returns (ServerParameterType1); ②
  rpc PushFunction(ClientParameterType2) returns (stream ServerParameterType2); ③
 }
```

① **ServiceName** is service stub name. This name will used for the access to the service functions

② **SimpleFunction** is simple request / reply function.

③ **PushFunction** is push function with one request and streams data from the server. Please check **stream** keyword before ServerParameterType2

Important

You don't need to use protobuf files directly, but it's good idea to know a syntax of the the protocol. These files are need if you want to use TAPI with other progaming languages. grpc.io conatains all necessary infomation about code generation and usage for all supported programming languages.

Create services

There are two types of the service functions: blocking and unblocking service functions. Blocking service functions are used for the synchronical answer from the function and similar to the typical program functions. By call of the blocking functions, program waits until a result of the function is arrived. It can produce timeouts by the execution. Unblocking function are used for the background processing. In this case a program don't wait until result is arrived and listen for the result asynchronically. This type of the stubs is used for the push functions. For more information please refer to the grpc.io web site.

For blocking calls used blocking service stubs, for asynchronical calls unblocking service stubs.

Example Java: Initaialize Stubs

```
private final ManagedChannel channel;
private final AccessServiceGrpc.AccessServiceBlockingStub accessServiceBlockingStub;
private final SecurityServiceGrpc.SecurityServiceBlockingStub
securityServiceBlockingStub;
private final SecurityServiceGrpc.SecurityServiceStub securityServiceStub;
private final StockExchangeServiceGrpc.StockExchangeServiceBlockingStub
stockExchangeServiceBlockingStub;
private final AccountServiceGrpc.AccountServiceBlockingStub
accountServiceBlockingStub;
private final AccountServiceGrpc.AccountServiceStub accountServiceStub;
private final OrderServiceGrpc.OrderServiceStub orderServiceStub;
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;
private final DepotServiceGrpc.DepotServiceStub depotServiceStub;
/**
 * Construct client for accessing ActiveTrader using the existing channel.
 */
TestClient(ManagedChannel channel) { ①
  this.channel = channel;
  securityServiceBlockingStub = SecurityServiceGrpc.newBlockingStub(channel); ②
  securityServiceStub = SecurityServiceGrpc.newStub(channel); ③
  stockExchangeServiceBlockingStub = StockExchangeServiceGrpc.newBlockingStub(channel
);
  accountServiceBlockingStub = AccountServiceGrpc.newBlockingStub(channel);
  accountServiceStub = AccountServiceGrpc.newStub(channel);
  orderServiceStub = OrderServiceGrpc.newStub(channel);
  orderServiceBlockingStub = OrderServiceGrpc.newBlockingStub(channel);
  depotServiceStub = DepotServiceGrpc.newStub(channel);
  accessServiceBlockingStub = AccessServiceGrpc.newBlockingStub(channel);
}
```

① Use managed channel from previous step

② Initialize blocking stub

③ Initialize non blocking stub

Warning

Be care that you use non blocking stub to subscribe for push events. It's technically possible, but blocks program execution until server will stop send events.

Functional programming

Objects in the GRPC world are **immutable** (or not changable). To create these objects are used object builders. They fill objects and build final immutable object for the execution. It looks like a chain of the functions calls.

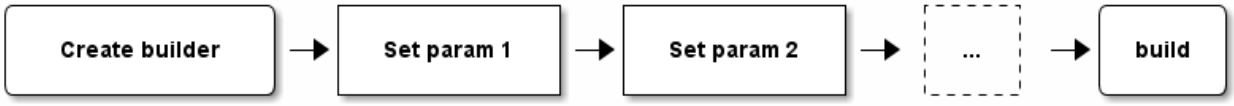
Example Java: Create object

```
AddOrderRequest request = AddOrderRequest.newBuilder() ①
    .setAccessToken(accessToken) ②
    .setAccountNumber("123467890")
    .setSecurityWithStockexchange(securityWithStockExchange)
    .setValidation(Validation.VALIDATE_ONLY)
    .setAmount(1)
    .setOrderModel(OrderModel.LIMIT)
    .setLimit(1)
    .setOrderType(OrderType.BUY)
    .setCashQuotation(CashQuotation.KASSA)
    .setValidityDate(Date.newBuilder().setDay(10).setMonth(10).setYear(2018).build())
    .build(); ③
```

① Create builder

② Set objects parameters

③ Build final object



Important

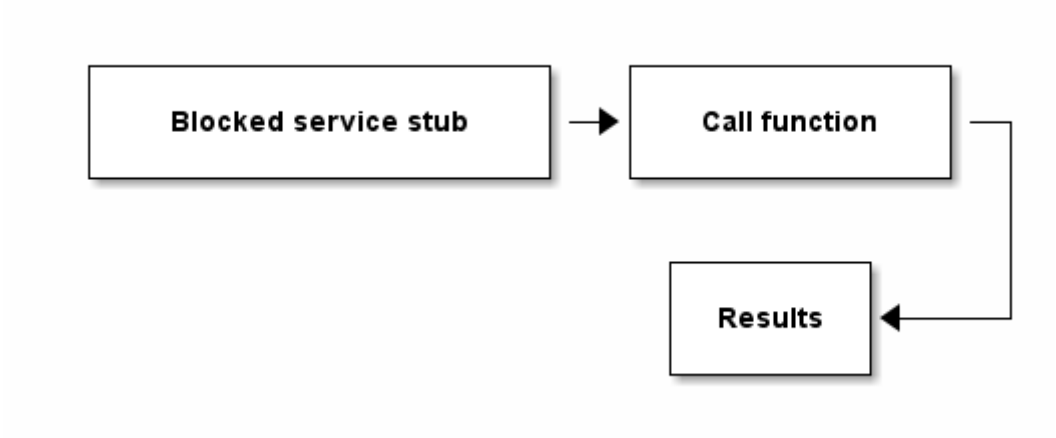
It's necessary to fill only parameters that need for the function call. All other parameters should stay untouched or should have default values.

Pull and push functions

There is two possibilites to call remote function.

Pull requests

The first one is using blocking service stub. In this case an operation blocks program execution until results are arrived.

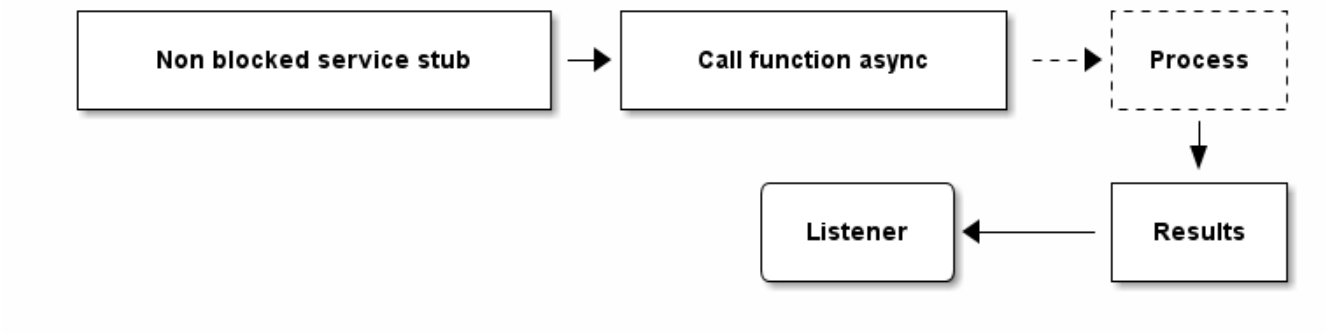


Example Java: Blocking execution

```
private TradingAccounts getTradingAccounts() {
  AccessTokenRequest request = AccessTokenRequest.newBuilder()
      .setAccessToken(accessToken)
      .build();
  return accountServiceBlockingStub.getTradingAccounts(request);
}
```

Push requests

Other possibility is not to wait for the results, but listen for then asynchronically.



Example Java: Noblocking execution

```
private void getTradingAccountsAsync() {
  AccessTokenRequest request = AccessTokenRequest.newBuilder()
      .setAccessToken(accessToken)
      .build();
  accountServiceStub.getTradingAccounts(request, new StreamObserver<TradingAccounts>()
{ ①
    @Override
    public void onNext(TradingAccounts tradingAccounts) {
      System.out.println(tradingAccounts); ②
    }

    @Override
    public void onError(Throwable t) {
      t.printStackTrace(); ③
    }

    @Override
    public void onCompleted() {
      System.out.println("No results anymore"); ④
    }
  });
}
```

① Subscribe for the results

② Process results

③ Called by technical error

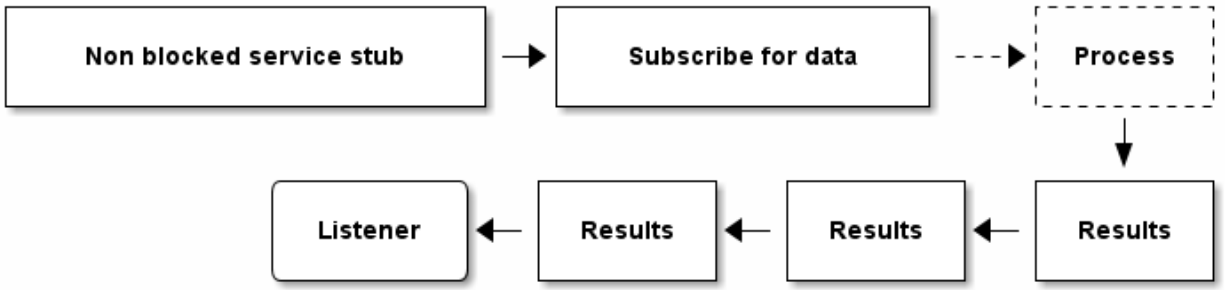
④ Called after processing end. No new result are comming anymore.

The results are comming only once. The operation will be automatically completed.

Push subscriptions

In some cases server can sent more then one answer or **streams** the data. This case is simular t requests, but has some differences.

- It can come more then one results answer back
- Server can send "break" notification to the client if no data is exist anymore
- Client can send "break" notification to the server if no data is need anymore. See **ServerSubs**



Services and functions

Access service

Access service provides functionality to validation / invalidation of the client.

Table 1. AccessService

Function	Description
LoginReply = Login(LoginRequest)	Validates client by the TAPI and gets access data
LogoutReply = Logout(LogoutRequest)	Invalidates client by the TAPI and gets logout result

Client validation

Client validation should be a first request after connection initalization. To validate client you have to set a **secret** that was set in the ActiveTrader configuration and process a login function. As result of this function returned session access token. This token is necessary to set to each request that will be send by TAPI.

Example Java: Login

```
private String login(String secret) {
  LoginRequest loginRequest = LoginRequest.newBuilder()
      .setSecret(secret)
      .build();
  LoginReply loginReply = accessServiceBlockingStub.login(loginRequest);
  if (loginReply.getError()!=Error.getDefaultInstance()) {
    System.err.println(loginReply.getError());
    return "";
  } else {
    return loginReply.getAccessToken();
  }
}
```

Client invalidation

After finishing of usage of the TAPI is necessary to invalidate session. That can be done with help of the logout function. .Example Java: Logout

```
private void logout() {
  if (accessToken!=null && !accessToken.isEmpty()) {
    LogoutRequest request = LogoutRequest.newBuilder()
        .setAccessToken(accessToken)
        .build();
    accessServiceBlockingStub.logout(request);
  }
}
```

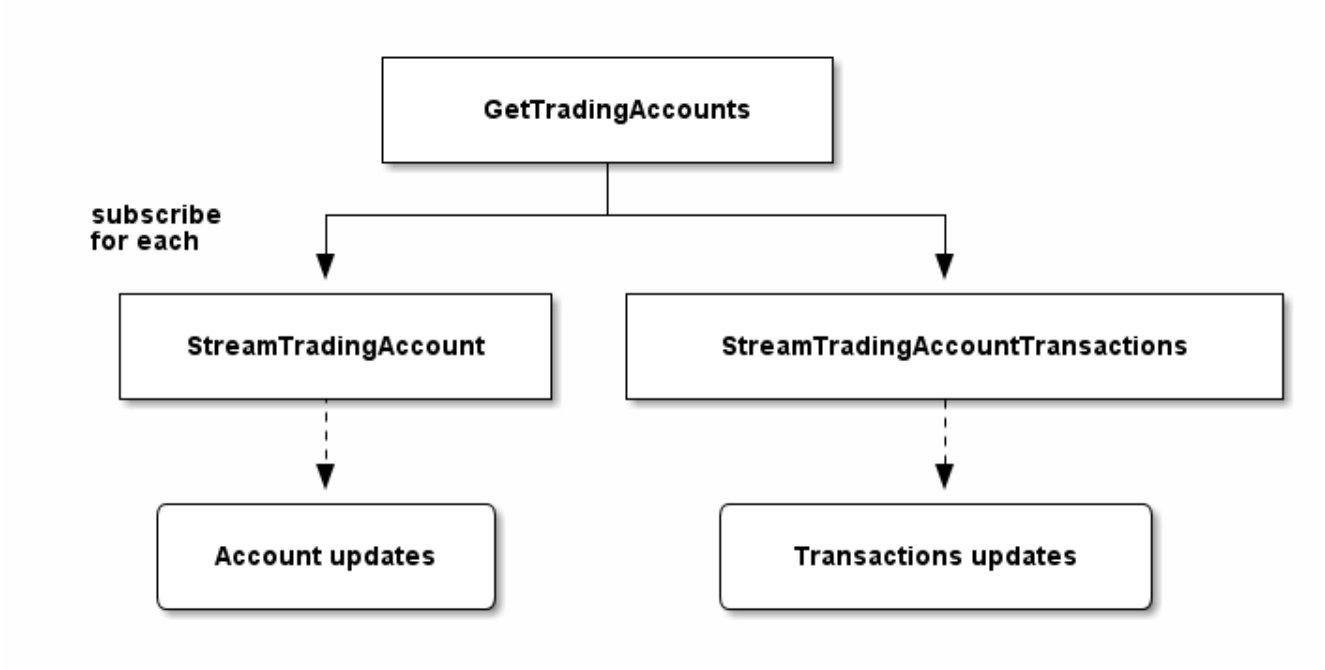
Account service

Account service provides access to the trading accounts and gives possibility to access to the accounts data and transactions.

Table 2. AccountService

Function	Description
TradingAccounts = GetTradingAccounts(AccessTokenRequest)	Gets trading accounts **@return** TradingAccounts List of trading accounts
TradingAccountInformation ⇐ StreamTradingAccount(TradingAccountRequest)	Subscribes one trading account for updates **@param** TradingAccount Trading account for push **@stream** TradingAccountInformation Specific information for subscribed account (balance, kredit line, etc.)
TradingAccountTransactions ⇐ StreamTradingAccount Transactions(TradingAccountRequest)	Subscribes one trading account for the transactions updates **@param** TradingAccount Trading account for push **@stream** TradingAccountInformation Transactions list for subscribed account

To subscribe for the account and transactions changes you have to get trading accounts and subscribe each account with help of steam functions for data changes independently.



Get list of trading accounts

Many activites need information about an account data. Most important information is a trading account number. To get list of all avialable trading accounts you have to use **Get Trading Accounts** function.

Example Java: getTradingAccounts()

```
private static final Empty EMPTY = Empty.getDefaultInstance();

private final AccountServiceGrpc.AccountServiceBlockingStub
accountServiceBlockingStub;

private TradingAccounts getTradingAccounts() {
  AccessTokenRequest request = AccessTokenRequest.newBuilder()
      .setAccessToken(accessToken)
      .build();
  return accountServiceBlockingStub.getTradingAccounts(request);
}
```

Important

Not all accounts in the pro version of the ActiveTrader can be used for the trading functionality. Please refer to the TradingAccount and check for the access avialibility of the selected account (flag: **tradable**).

Useful

Typical use case is to get this information once and keep it during one connection session.

Stream account information changes

To get pushed trading account information is need to subscribe an account for the trading account stream.

Example Java: streamTradingAccountInformation()

```
private void streamTradingAccountInformation() {
  TradingAccounts tradingAccounts = getTradingAccounts(); ①
  for (int accountIndex = 0; accountIndex<tradingAccounts.getAccountsCount();
accountIndex++) {
    TradingAccount account = tradingAccounts.getAccounts(accountIndex); ②
    TradingAccountRequest request = TradingAccountRequest.newBuilder()
        .setAccessToken(accessToken)
        .setTradingAccount(account)
        .build();

    TradingAccountInformationObserver observer = ③
        new TradingAccountInformationObserver(request, accountServiceStub);
    accountObservers.put(account, observer);
  }
}
```

① Get trading accounts

② Iterate over each account

③ Subscribe account

TradingAccountInformationObserver is the help class to encapsulate typical processing with the stream data.

Example Java: TradingAccountInformationObserver

```
import com.consorsbank.module.tapi.grpc.AccountServiceGrpc.AccountServiceStub;
import com.consorsbank.module.tapi.grpc.account.TradingAccount;
import com.consorsbank.module.tapi.grpc.account.TradingAccountInformation;
import com.consorsbank.module.tapi.grpc.account.TradingAccountRequest;
import com.consorsbank.module.tapi.grpc.common.Error;


public class TradingAccountInformationObserver extends ServerSubscription
<TradingAccount, TradingAccountInformation>{
  private final AccountServiceStub orderServiceStub;

  public TradingAccountInformationObserver(
      TradingAccountRequest request, AccountServiceStub orderServiceStub) {

    this.orderServiceStub = orderServiceStub;
    this.orderServiceStub.streamTradingAccount(request, this); ①
  }

  @Override
  public void onCompleted() { ②
    System.out.println("Call completed!");
  }

  @Override
  public void onNext(TradingAccountInformation reply) { ③
    Error error = reply.getError(); ④
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async trading information: %s%n", reply);
    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for account information

② Called on the completting subscription

③ Called on the account information change

④ Check for an error

ServerSubscription contains functions to control streaming of the data. If, for example, you don't need to recieve data from any subscription, you can unsubscribe it with help of **close** function.

Example Java: ServerSubscription

```
import io.grpc.Status;
import io.grpc.StatusRuntimeException;
import io.grpc.stub.ClientCallStreamObserver;
import io.grpc.stub.ClientResponseObserver;

public abstract class ServerSubscription<R, T> implements ClientResponseObserver<R, T
>, Closeable {
  private ClientCallStreamObserver<R> observer;

  /**
   * This function is called during subscription initialization.
   * observer will created by GRPC engine and can be used for
   * data flow control.
   */
  @Override
  public void beforeStart(ClientCallStreamObserver<R> observer) {
    this.observer = observer; ①
  }

  /**
   * Call this function to unsubscribe from server
   */
  @Override
  public void close() {
    observer.cancel("Fin", null); ②
  }
  // ...
}
```

① Store client observer by initialization

② Stop listening events (unsubscribe from stream)

Stream account transactions changes

To get pushed trading account transactions list is need to subscribe account for trading account transactions stream.

Example Java: streamTradingAccountTransactions()

```
private void streamTradingAccountTransactions() {
  TradingAccounts tradingAccounts = getTradingAccounts(); ①
  for (int accountIndex = 0; accountIndex<tradingAccounts.getAccountsCount();
accountIndex++) {
    TradingAccount account = tradingAccounts.getAccounts(accountIndex); ②
    TradingAccountRequest request = TradingAccountRequest.newBuilder()
        .setAccessToken(accessToken)
        .setTradingAccount(account)
        .build();
    TradingAccountTransactionsObserver observer = ③
        new TradingAccountTransactionsObserver(request, accountServiceStub);
    accountTransactionsObservers.put(account, observer);
  }
}
```

① Get trading accounts

② Iterate over each account

③ Subscribe account

Example Java: TradingAccountTransactionsObserver

```
import com.consorsbank.module.tapi.grpc.AccountServiceGrpc.AccountServiceStub;
import com.consorsbank.module.tapi.grpc.account.TradingAccount;
import com.consorsbank.module.tapi.grpc.account.TradingAccountRequest;
import com.consorsbank.module.tapi.grpc.account.TradingAccountTransactions;
import
com.consorsbank.module.tapi.grpc.account.TradingAccountTransactions.Transaction;
import com.consorsbank.module.tapi.grpc.common.Error;

public class TradingAccountTransactionsObserver extends ServerSubscription
<TradingAccount, TradingAccountTransactions>{
  private final AccountServiceStub orderServiceStub;

  public TradingAccountTransactionsObserver(
      TradingAccountRequest request, AccountServiceStub orderServiceStub) {

    this.orderServiceStub = orderServiceStub;
    this.orderServiceStub.streamTradingAccountTransactions(request, this); ①
  }

  @Override
  public void onCompleted() { ②
    System.out.println("Call completed!");
  }

  @Override
  public void onNext(TradingAccountTransactions transactions) { ③
    Error error = transactions.getError(); ④
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async trading transactions: %s%n", transactions);
      TradingAccount account = transactions.getAccount();
      List<Transaction> transactionsList = transactions.getTransactionsList();
      // ...
    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for the account information

② Called on the completting subscription

③ Called on the account transactions change

④ Check for an error

Stock exchange service

StockExchange service provides access to the stock exchanges.

Table 3. StockExchangeService

Function	Description
StockExchangeDescriptions = GetStockExchanges(AccessTokenRequest)	Gets predefined stockexchages **@return** StockExchangeDescriptions list of stock exchange informations
StockExchangeDescription = GetStockExchange(StockExchangeRequest)	Gets specific stock exchange **@param** StockExchange Requested stock exchange **@return** StockExchangeDescription Stock exchange information

Get information about all stock exchanges

TAPI has list of predefined stock exchanges that can be fetched directly. Each stock exchange has id and optional issuer. Some stock exchanges have identical id's, but different issuers. For examples:

id	issuer	shortcut id	stock exchange name
ETR		ETR	Xetra
OTC	BAAD	BAA	Baada bank on OTC
OTC	7649	LUS	Lang und Schwarz
OTC	7004		Commerzbank
OTC		OTC	Any issuer on OTC

Useful

It's possible to use shortcut id, if it exists, to access to the same stock exchange. For example stock exchange ("OTC", "BAAD") equivalent to the stock exchange ("BAA","")

These stock exchanges can be requested with help of **Get Stock Exchanges** function. As a result will delivered stock exchanges with names.

Example Java: getStockExchanges()

```
private static final Empty EMPTY = Empty.getDefaultInstance();

private final StockExchangeServiceGrpc.StockExchangeServiceBlockingStub
stockExchangeServiceBlockingStub;

private StockExchangeDescriptions getStockExchanges() {
  AccessTokenRequest request = AccessTokenRequest.newBuilder()
      .setAccessToken(accessToken)
      .build();
  return stockExchangeServiceBlockingStub.getStockExchanges(request);
}
```

Useful

This information can be stored and reused in the user application.

Get information about specific stock exchange

It's also possible to get information about one specific stock exchange. Next example shows how to request stock exchange information about Baada issuer on the OTC:

Example Java: getBaadaStockExchangeInformation()

```
private StockExchangeDescription getBaadaStockExchangeInformation() {
  StockExchange stockExchange = StockExchange.newBuilder()
      .setId("OTC")
      .setIssuer("BAAD")
      .build(); ①

  StockExchangeRequest request = StockExchangeRequest.newBuilder()
      .setAccessToken(accessToken)
      .setStockExchange(stockExchange)
      .build(); ②
  return stockExchangeServiceBlockingStub.getStockExchange(request); ③
}
```

① Prepare Baada on OTC stock exchange

② Build request

③ Request information about stock exchange

Depot service

With help of the depot service is possible to stream information about depot entries. Depot service has next functions:

Stream depot changes

Table 4. DepotService

Function	Description
DepotEntries ⇐ StreamDepot(TradingAccountRequest)	Subscribes one trading account for the depot data updates **@param** TradingAccount Trading account for push **@stream** DepotEntries depot entries linked to the account
Empty = UpdateDepot(TradingAccountRequest)	Initiates depot update action. All changes come by the StreamDepot subscription. This function doesn't wait for the action result. **@param** TradingAccount Trading account for update

Example Java: streamDepotData()

```
private final DepotServiceGrpc.DepotServiceStub depotServiceStub;

private void streamDepotData() {
  TradingAccounts tradingAccounts = getTradingAccounts(); ①
  for (int accountIndex = 0; accountIndex<tradingAccounts.getAccountsCount();
accountIndex++) {
    TradingAccount account = tradingAccounts.getAccounts(accountIndex);

    TradingAccountRequest request = TradingAccountRequest.newBuilder()
        .setAccessToken(accessToken)
        .setTradingAccount(account)
        .build();

    DepotObserver depotObserver = new DepotObserver(request, depotServiceStub); ②
    depotObservers.put(account, depotObserver);
  }
}
```

① Get trading accounts

② Greate depot observer for each account

Update events are delivered on each depot entry change and contain information about whole depot.

Example Java: DepotObserver

```
public class DepotObserver extends ServerSubscription<TradingAccountRequest,
DepotEntries>{
  private final DepotServiceStub serviceStub;

  public DepotObserver(TradingAccountRequest request, DepotServiceStub
depotServiceStub) {
    this.serviceStub = depotServiceStub;
    this.serviceStub.streamDepot(request, this); ①
  }


  @Override
  public void onCompleted() { ②
    System.out.println("Call completed!");
  }

  @Override
  public void onNext(DepotEntries depotEntries) { ③
    Error error = depotEntries.getError(); ④
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async depot: %s%n", depotEntries);
      TradingAccount account = depotEntries.getAccount();
      List<DepotEntry> entriesList = depotEntries.getEntriesList();
      // ...

    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for the depot data

② Called on the completing subscription

③ Called on the depot entries change

④ Check for an error

Update depot

Sometime is necessary to update depot manually. To initiate this update you have to call **UpdateDepot** function. This function takes trading account as input parameter and doesn't deliver any results. It's also don't wait for the results. All changes are delivered directly to the depot listeners in the background.

Warning

This function used for the compatibility proposals and can be removed in the future versions of the TAPI.

Security service

Securities access

All securities can be accessed with help of security codes. Typical trading securities codes are WKN and ISIN. TAPI is able to operate with additional security code types:

Security code type	Tradable	Description
WKN	Yes	WKN
ISIN	Yes	ISIN
ID_NOTATION	No	FactSet id notation
ID_OSI	No	FactSet id osi
ID_INSTRUMENT	No	FactSet id instrument
MNEMONIC	No	German short id
MEMONIC_US	No	US short id (only for us instruments)

Security code can be created with help of next code.

Example Java: getSecurityCode()

```
public static SecurityCode getSecurityCode(String code, SecurityCodeType codeType) {
  return SecurityCode.newBuilder() ①
      .setCode(code) ②
      .setCodeType(codeType==null ? SecurityCodeType.NO_CODE_TYPE : codeType) ③
      .build(); ④
}
```

① Create builder for SecurityCode object

② Set security code

③ Set security code type (WKN, ISIN, . . .)

④ Build immutable SecurityCode object

Important

In some cases TAPI can guess security code type by input security code value. In this case the security code type should be set to **NO_CODE_TYPE** value. Please care that's not guaranty right detection of the security code. The good practice is to use security code type by the building of the security code object.

Security service contains next functions:

Table 5. SecurityService

Function	Description
SecurityInfoReply = GetSecurityInfo(SecurityInfoRequest)	Gets security information about security **@param** SecurityInfoRequest Request object with interested security **@return** SecurityInfoReply Complete information about security
SecurityMarketDataReply ⇐ StreamMarketData(SecurityMarketDataRequest)	Subscribes security with stock exchange for market data updates **@param** SecurityMarketDataRequest Market data request with interested security and stock exchange **@stream** SecurityMarketDataReply Reply with all market data values
SecurityOrderBookReply ⇐ StreamOrderBook(SecurityOrderBookRequest)	Subscribes security with stock exchange for orderbook updates **@param** SecurityOrderBookRequest Orderbook data request with interested security and stock exchange **@stream** SecurityOrderBookReply Reply with all orderbook values
CurrencyRateReply ⇐ StreamCurrencyRate(CurrencyRateRequest)	Subscribes for currency rate from one currency to another currency. **@param** SecurityOrderBookRequest currency rate request with interested currencies from/to **@stream** CurrencyRateReply reply with currency rate
SecurityPriceHistoryReply = GetSecurityPriceHistory(SecurityPriceHistoryRequest)	Requests history data for one security on one stockexchange in intraday or historical format **@param** SecurityPriceHistoryRequest Data with security, stockexchange, how many days and resolution **@return** SecurityPriceHistoryReply List of the historical quotes or an error

Get security information

Security information can be requested with help of **GetSecurityInfo** function. As a result will delivered security name, class, codes and stock exchanges with trading possibilites. This information can be used by the ordering. It also contains data about tradabity on the different markets and by different issuers. The flags LimitToken with values **QUOTE_ONLY** and **LIMIT_AND_QUOTE** show if for the security is possible to use quote buy and sell on the selected market by the selected issuer. For more information please see GetQuote and AcceptQuote functions.

Example Java: requestSecurityInfo()

```
private final SecurityServiceGrpc.SecurityServiceBlockingStub
securityServiceBlockingStub;

private SecurityInfoReply requestSecurityInfo(String code, SecurityCodeType
securityCodeType) {
  SecurityCode securityCode = TestUtils.getSecurityCode(code, securityCodeType); ①
  SecurityInfoRequest request = SecurityInfoRequest.newBuilder() ②
      .setAccessToken(accessToken)
      .setSecurityCode(securityCode) ③
      .build(); ④
  SecurityInfoReply response = securityServiceBlockingStub.getSecurityInfo(request);
⑤
  return response;
}
```

① Create security code object

② Create builder for SecurityinfoRequest object

③ Set security code

④ Build immutable SecurityinfoRequest object

⑤ Request security information

Stream market data information

To get pushed market data (push courses) is need to subscribe security and stock exchange for this information with help of **StreamMarketData** function. **SecurityMarketDataReply** contains all values, which were changed.

Example Java: streamMarketData()

```
private final SecurityServiceGrpc.SecurityServiceStub securityServiceStub;

private MarketDataDataObserver streamMarketData(String code,
    SecurityCodeType securityCodeType, String stockExchangeId, String currency) {

  SecurityWithStockExchange securityWithStockExchange = ①
      TestUtils.getSecurityWithStockExchange(code, securityCodeType, stockExchangeId,
null);

  SecurityMarketDataRequest request =  SecurityMarketDataRequest.newBuilder()
      .setAccessToken(accessToken)
      .setSecurityWithStockexchange(securityWithStockExchange) ②
      .setCurrency(currency!=null ? currency : "")
      .build();

  MarketDataDataObserver marketDataDataObserver = ③
      new MarketDataDataObserver(request, securityServiceStub);

  String key = code+"|"+securityCodeType+"|"+stockExchangeId+"|"+currency;
  marketDataObservers.put(key, marketDataDataObserver);
  return marketDataDataObserver;
}
```

① Prepare security with stock exchange

② Prepare request

③ Greate **MarketDataDataObserver** to subscribe market data information

Update events deliver changed market data values.

To subscribe for the market data information it is necessary to create **SecurityWithStockExchange** object.

Example Java: getSecurityWithStockExchange()

```
public static SecurityWithStockExchange getSecurityWithStockExchange(
    SecurityCode securityCode, StockExchange stockExchange) {

  return SecurityWithStockExchange.newBuilder() ①
      .setSecurityCode(securityCode) ②
      .setStockExchange(stockExchange) ③
      .build(); ④
}
```

① Create builder for SecurityWithStockExchange object

② Set security code

③ Set stock exchange

④ Build immutable SecurityWithStockExchange object

You have to prepare **StockExchange** object.

Example Java: getStockExchange()

```
public static StockExchange getStockExchange(String stockExchangeId, String issuer) {
  Builder stockBuilder = StockExchange.newBuilder(); ①
  if (issuer!=null) {
    stockBuilder.setIssuer(issuer); ②
  }
  return stockBuilder.setId(stockExchangeId) ③
      .build(); ④
}
```

① Create builder for StockExchange object

② Set issuer

③ Set stock exchange id

④ Build immutable StockExchange object

Example Java: MarketDataDataObserver

```
public class MarketDataDataObserver extends ServerSubscription
<SecurityMarketDataRequest, SecurityMarketDataReply> {
  private final SecurityServiceStub securityServiceStub;

  public MarketDataDataObserver(SecurityMarketDataRequest request,
      SecurityServiceStub securityServiceStub) {

    this.securityServiceStub = securityServiceStub;


    securityServiceStub.streamMarketData(request, this); ①
  }

  @Override
  public void onCompleted() {
    System.out.println("Call completed!"); ②
  }

  @Override
  public void onNext(SecurityMarketDataReply response) {
    Error error = response.getError();
    if (error==Error.getDefaultInstance()) { ③
      System.out.printf("Async client onNext: %s%n", response);
      double lastPrice = response.getLastPrice();
      Timestamp lastDateTime = response.getLastDateTime();
      // ...

    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for the market data

② Called of complete subscription

③ Process madret data events

Stream orderbook data

To get pushed orderbook data (second level push courses) is need to subscribe security and stockexchange for this information with help of **StreamOrderBook** function. SecurityOrderBookReply contains actual orderbook values. To create **SecurityOrderBookRequest** is need to create SecurityWithStockExchange object first. Currently order book data supported only on Xetra stock exchange.

Example Java: streamOrderBook()

```
private final SecurityServiceGrpc.SecurityServiceStub securityServiceStub;

private OrderBookDataObserver streamOrderBook(String code,
    SecurityCodeType securityCodeType, String stockExchangeId, String currency) {

  SecurityWithStockExchange securityWithStockExchange =  ①
      TestUtils.getSecurityWithStockExchange(code, securityCodeType, stockExchangeId,
null);

  SecurityOrderBookRequest request = SecurityOrderBookRequest.newBuilder()
      .setAccessToken(accessToken)
      .setSecurityWithStockexchange(securityWithStockExchange)
      .setCurrency(currency==null ? "" : currency)
      .build(); ②

  OrderBookDataObserver orderBookDataObserver =  ③
      new OrderBookDataObserver(request, securityServiceStub);

  String key = code+"|"+securityCodeType+"|"+stockExchangeId+"|"+currency;
  orderBookObservers.put(key, orderBookDataObserver);
  return orderBookDataObserver;
}
```

① Prepare security with stock exchange

② Prepare request

③ Greate **OrderBookData** observer to subscribe orderbook information

Example Java: OrderBookDataObserver

```
public class OrderBookDataObserver extends ServerSubscription<
SecurityOrderBookRequest, SecurityOrderBookReply> {
  private final SecurityServiceStub securityServiceStub;

  public OrderBookDataObserver(SecurityOrderBookRequest request,
      SecurityServiceStub securityServiceStub) {
    this.securityServiceStub = securityServiceStub;

    securityServiceStub.streamOrderBook(request, this); ①
  }

  @Override
  public void onCompleted() {
    System.out.println("Call completed!"); ②
  }

  @Override
  public void onNext(SecurityOrderBookReply response) {
    Error error = response.getError();
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async client onNext: %s%n", response); ③
      List<OrderBookEntry> orderBookEntriesList = response.getOrderBookEntriesList();
      for (OrderBookEntry orderBookEntry : orderBookEntriesList) {
        double askPrice = orderBookEntry.getAskPrice();
        double askVolume = orderBookEntry.getAskVolume();
      }
      // ...

    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for orderbook data

② Called on complete subscription

③ Process order book events

Stream currency rate

It is possible to get pushed currency rate from one currency to other. This information can be used for the realtime convertation of the money values.

Important

For the market data quotes / order books it is possible to define target currency. It this case the convertation will be processed automatically.

Example Java: streamCurrencyRate(String currencyFrom, String currencyTo)

```
private final SecurityServiceGrpc.SecurityServiceStub securityServiceStub;

private CurrencyRateDataObserver streamCurrencyRate(String currencyFrom, String
currencyTo) {
  CurrencyRateRequest request = CurrencyRateRequest.newBuilder()
      .setAccessToken(accessToken)
      .setCurrencyFrom(currencyFrom)
      .setCurrencyTo(currencyTo)
      .build(); ①

  CurrencyRateDataObserver currencyRateDataObserver = ②
      new CurrencyRateDataObserver(request, securityServiceStub);

  String key = currencyFrom+"|"+currencyTo;
  currencyObservers.put(key, currencyRateDataObserver);
  return currencyRateDataObserver;
}
```

① Greate request

② Greate CurrencyRateData observer to subscribe currency rate

Example Java: CurrencyRateDataObserver

```
public class CurrencyRateDataObserver extends ServerSubscription<CurrencyRateRequest,
CurrencyRateReply> {
  private final SecurityServiceStub securityServiceStub;

  public CurrencyRateDataObserver(CurrencyRateRequest request,
      SecurityServiceStub securityServiceStub) {

    this.securityServiceStub = securityServiceStub;


    securityServiceStub.streamCurrencyRate(request, this); ①
  }

  @Override
  public void onCompleted() {
    System.out.println("Call completed!"); ②
  }

  @Override
  public void onNext(CurrencyRateReply response) {
    Error error = response.getError();
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async client onNext: %s%n", response);
      double currencyRate = response.getCurrencyRate(); ③
      // ...

    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for currency rate

② Called on complite subscription

③ Process currency rate events

Get security historic data

Security historic data can be requested with help of getSecurityPriceHistory() function. As a result will delivered security with stock exchange, currency, historic data list. The parameter **days** defines how many trading day will be requested. For intraday types of the time resolution this value should be **15 or less**. Be careful the amount of the data can be very big, especialy for intraday data with tick resolution.

Example Java: getSecurityPriceHistory()

```
private final SecurityServiceGrpc.SecurityServiceBlockingStub
securityServiceBlockingStub;

private SecurityPriceHistoryReply getSecurityPriceHistory(
    String code, SecurityCodeType securityCodeType, String stockExchangeId,
    int days, TimeResolution resolution) {

  SecurityWithStockExchange securityWithStockExchange =  ①
      TestUtils.getSecurityWithStockExchange(code, securityCodeType, stockExchangeId,
null);
  SecurityPriceHistoryRequest request = SecurityPriceHistoryRequest.newBuilder() ②
      .setAccessToken(accessToken)
      .setSecurityWithStockexchange(securityWithStockExchange) ③
      .setDays(days) ④
      .setTimeResolution(resolution) ⑤
      .build(); ⑥
  SecurityPriceHistoryReply response =
      securityServiceBlockingStub.getSecurityPriceHistory(request); ⑦
  return response;
}
```

① Create security with stock exchange object

② Create builder for SecurityPriceHistoryRequest object

③ Set security with stock exchange

④ Set days number

⑤ Set time resolution (tick, minute, etc)

⑥ Build immutable SecurityPriceHistoryRequest object

⑦ Request historic data

Important

Last example will block program execution until the result is returned. Sometimes to avoid timeouts by the listening it's better to use non blocking calls.

Order service

With help of the order service is possible to execute, change, activate, deactivate or cancel orders. It's also allow to listen for the orders and to control them asyncronically.

Order types and parameters

TAPI allowes to execute different order types on the stock exchanges with different parameters. Next tables shows what parameters are necessary (**x**) and optional (**o**) depends from selected stock exchange and order model. The full list of order possibilites is delivered by the GetSecurityInformation function.

Parameter \ Order model	Market	Limit	StopMarket	StopLimit	OneCancelsOtherMarket	OneCancelsOtherLimit	TrailingStopMarket	TrailingStopLimit
account_number	x	x	x	x	x	x	x	x
security_with_stockexchange	x	x	x	x	x	x	x	x
order_type	x	x	x	x	x	x	x	x
amount	x	x	x	x	x	x	x	x
order_supplement	o, 1)	o, 1)						
validity_date	x	x	x	x	x	x	x	x
limit		x			x	x		
stop			x	x	x	x	x	x
stop_limit				x		x		
trailing_distance							x	x
trailing_notation							x	x
trailing_limit_tolerance								x, 2)
dripping_quantity	o, 3)	o, 3)						
position_id	o, 4)	o, 4)	o, 4)	o, 4)	o, 4)	o, 4)	o, 4)	o, 4)
validation	o ,5)	o ,5)	o ,5)	o ,5)	o ,5)	o ,5)	o ,5)	o ,5)
cash_quotation	o, 6)	o, 6)	o, 6)	o, 6)	o, 6)	o, 6)	o, 6)	o, 6)
risk_class_override	o	o	o	o	o	o	o	o
target_market_override	o	o	o	o	o	o	o	o
tax_nontransparent_override	o	o	o	o	o	o	o	o

Parameter \ Order model	Market	Limit	StopMarket	StopLimit	OneCancelsOtherMarket	OneCancelsOtherLimit	TrailingStopMarket	TrailingStopLimit
accept_additional_fees	o	o	o	o	o	o	o	o
closed_realestate_fond_override	o	o	o	o	o	o	o	o

① Stock exchange: ETR only, values: IMMIDIATE_OR_CANCEL, FILL_OR_KILL, ICEBERG

② Stock exchange: TRG only

③ Stock exchange: ETR only, when order_supplement equals ICEBERG, pro only

④ Position id is optional field by the order action with existing depot position

⑤ Default value: **WITHOUT_VALIDATION**. Order is routed directly to market

⑥ Cash quotation:

Stock exchange	NOTHING	KASSA	AUCTION	OPENING	INTRADAY	CLOSING
ETR	x	x	x	x	x	x
German	x	x				
Other	x					

The full list of the order service functions can be checked in the next list. Some functionality avialable only in pro version of the ActiveTrader.

Important

To execute new order is need at least to fill all madatory fields. See **x** values in the each column for selected order model.

Example Java: OneCancelOtherMarket order

```
SecurityWithStockExchange securityWithStockExchange = TestUtils
.getSecurityWithStockExchange("710000", SecurityCodeType.WKN, "ETR", null);

//
AddOrderRequest addRequest = AddOrderRequest.newBuilder() ①
    .setOrderModel(OrderModel.ONE_CANCELS_OTHER_MARKET) ②
    .setAccountNumber(account.getAccountNumber())
    .setSecurityWithStockexchange(securityWithStockExchange)
    .setAmount(1)
    .setValidation(Validation.VALIDATE_ONLY)
    .setLimit(39)
    .setStop(40)
    .setOrderType(OrderType.BUY)
    .setValidityDate(TestUtils.getDate(LocalDate.now())) // today
    .build();
OrderReply addReply = getOrdersBlockingStub().addOrder(addRequest); ③
```

① Fill mandatory parameters

② OneCancelOtherMarket order and it's parameters

③ Process order

Stream orders

To get information about current orders ist need to subscribe for this information with help of StreamOrders function. All order changes will come asynchronically as a list of the all avialable orders.

Useful

An amount of the updated orders can be configured in the ActiveTrader general settings.

Example Java: streamOrders()

```
private final OrderServiceGrpc.OrderServiceStub orderServiceStub;

private void streamOrders() {
  TradingAccounts tradingAccounts = getTradingAccounts(); ①
  for (int accountIndex = 0; accountIndex<tradingAccounts.getAccountsCount();
accountIndex++) {
    TradingAccount account = tradingAccounts.getAccounts(accountIndex); ②
    TradingAccountRequest request = TradingAccountRequest.newBuilder()
        .setAccessToken(accessToken)
        .setTradingAccount(account)
        .build();
    OrdersObserver ordersObserver = new OrdersObserver(request, orderServiceStub); ③
    ordersObservers.put(account, ordersObserver);
  }
}
```

① Get trading accounts

② Iterate over each account

③ Subscribe account

Example Java: OrdersObserver

```
import com.consorsbank.module.tapi.grpc.OrderServiceGrpc.OrderServiceStub;
import com.consorsbank.module.tapi.grpc.account.TradingAccount;
import com.consorsbank.module.tapi.grpc.account.TradingAccountRequest;
import com.consorsbank.module.tapi.grpc.common.Error;
import com.consorsbank.module.tapi.grpc.order.Order;
import com.consorsbank.module.tapi.grpc.order.Orders;

public class OrdersObserver extends ServerSubscription<TradingAccount, Orders>{
  private final OrderServiceStub orderServiceStub;

  public OrdersObserver(TradingAccountRequest request, OrderServiceStub
orderServiceStub) {
    this.orderServiceStub = orderServiceStub;
    this.orderServiceStub.streamOrders(request, this); ①
  }

  @Override
  public void onCompleted() { ②
    System.out.println("Call completed!");
  }

  @Override
  public void onNext(Orders orders) { ③
    Error error = orders.getError(); ④
    if (error==Error.getDefaultInstance()) {
      System.out.printf("Async orders: %s%n", orders);
      TradingAccount account = orders.getAccount();
      List<Order> ordersList = orders.getOrdersList();
      // ...

    } else {
      System.out.printf("Error: %s%n", error);
    }
  }
}
```

① Subscribe for the order data

② Called on the completing subscription

③ Called on the depot entries change

④ Check for an error

Important

In the pro version of the ActiveTrader **order number** value can be changed (for example after activation / deactivation of the order). To match saved in the client application orders with the orders from update is need to use unique_id field from the Order object.

Update orders

Sometime is necessary to update orders manually. To initiate this update you have to call **UpdateOrders** function. This function takes trading account as input parameter and doesn't deliver any results. It's also don't wait for the results. All changes are delivered directly to the orders listeners in the background.

Warning

This function used for the compatibility proposals and can be removed in the future versions of the TAPI.

Get securities quotes

Before to execute order add action is often necessary to get actual quote information for the selected security. This information can be subscribed in the security service or requested in the order service. It's not a matter where this information is requested for the normal order add action, but it's important in case if you want to process accept quote on the short term market. In this case is necessary to use get quote function from order service.

Important

Please call get quote function before to call of the accept order action.

Example Java: requestQuotesDirect()

```
private void requestQuotesDirect(String code, SecurityCodeType securityCodeType) {
  QuoteRequest quoteRequest = QuoteRequest.newBuilder()
      .setAccessToken(accessToken)
      .setOrderType(OrderType.BUY) ①
      .setSecurityCode(TestUtils.getSecurityCode(code, securityCodeType)) ②
      .setAmount(10) ③
      .addStockExchanges(TestUtils.getStockExchange("BAA", null)) ④
      .addStockExchanges(TestUtils.getStockExchange("OTC", null))
      .addStockExchanges(TestUtils.getStockExchange("TRG", null))
      .build();
  QuoteReply quoteReply = orderServiceBlockingStub.getQuote(quoteRequest); ⑤
  if (quoteReply.getError()!=Error.getDefaultInstance()) {
    List<QuoteEntry> priceEntriesList = quoteReply.getPriceEntriesList(); ⑥
    for (QuoteEntry quoteEntry : priceEntriesList) {
      if (quoteEntry.getError()!=Error.getDefaultInstance()) {
        // ...
      }
    }
  }
}
```

① Set order type

② Set security code

③ Set amount for the ordering

④ Set requestes stock exchanges

⑤ Request quotes with help of the **blocking** stub

⑥ Process results

Other difference between quotes from the security service and order service is that it's depended from user rights. An order service can have limited amount of the requests for the long terms stockexchanges. You can request data more then from one stock exchange for one security at once. In this case function returns all results for all requested stock exchanges after processing of all data for each stock exchange. That can follow to the timeouts by the functions calls. You can use asychronical requests if to don't have timeouts is important. It's also possible to process many get quote requests in parallel and pack to each request only one stockexchange. The results from the requests can be processed in background.

Example Java: requestQuotesAsynchronically()

```
private void requestQuotesAsynchronically(String code, SecurityCodeType
securityCodeType) {
  final StreamObserver<QuoteReply> quotesObserver = new StreamObserver<QuoteReply>() {
①
    @Override
    public void onNext(QuoteReply quoteReply) {
      System.err.println(quoteReply); ②
      if (quoteReply.getError()!=Error.getDefaultInstance()) {
        List<QuoteEntry> priceEntriesList = quoteReply.getPriceEntriesList();
        // priceEntriesList will contain only one entry
        for (QuoteEntry quoteEntry : priceEntriesList) {
          StockExchange stockExchange = quoteEntry.getStockExchange();
          if (quoteEntry.getError()!=Error.getDefaultInstance()) {
            // ...
          }
        }
      }
    }
    @Override
    public void onError(Throwable t) {
      t.printStackTrace();
    }
    @Override
    public void onCompleted() {}
  };
  final SecurityCode securityCode = TestUtils.getSecurityCode(code, securityCodeType);
③
  final StockExchange[] stockexchanges = { ④
      TestUtils.getStockExchange("BAA", null),
      TestUtils.getStockExchange("OTC", null),
      TestUtils.getStockExchange("TRG", null),
  };

  // Request data
  for (StockExchange stockExchange : stockexchanges) {
    QuoteRequest quoteRequest = QuoteRequest.newBuilder() ⑤
        .setAccessToken(accessToken)
        .setOrderType(OrderType.BUY)
        .setSecurityCode(securityCode)
        .setAmount(10)
        .addStockExchanges(stockExchange)
        .build();
    orderServiceStub.getQuote(quoteRequest, quotesObserver);
  }

  // ...
}
```

① Prepare listener

② Process results in the background

③ Prepare security code

④ Prepare stock exchanges

⑤ Request quotes with help of the **non blocking** stub

Accept quote

To place an order on the short term market is necessary to request quote first. If **QuoteEntry** will contain filled **quote_reference** field then it's possible to place order with help of accept quote function to the short term market. Otherwise accept quote is not possible. By the calling of the accept quote function you have to use same parameters as you used for the get quote. Otherwise the request parameters will be denied and the action will be finished with an error.

Example Java: acceptQuote()

```
private void acceptQuote() {
  QuoteRequest quoteRequest = QuoteRequest.newBuilder() ①
      .setAccessToken(accessToken)
      .setOrderType(OrderType.BUY)
      .setSecurityCode(TestUtils.getSecurityCode("710000", SecurityCodeType.WKN))
      .setAmount(10)
      .addStockExchanges(TestUtils.getStockExchange("OTC", null))
      .build();

  PerformanceCounter pcGetQuote = new PerformanceCounter("GetQuote");
  QuoteReply quoteReply = orderServiceBlockingStub.getQuote(quoteRequest); ②
  pcGetQuote.print();

  if (quoteReply.getError()==Error.getDefaultInstance()) {
    QuoteEntry quoteEntry = quoteReply.getPriceEntries(0); ③
    if (quoteEntry.getError()==Error.getDefaultInstance() &&
        !quoteEntry.getQuoteReference().isEmpty()) { ④
      SecurityWithStockExchange securityWithStockExchange = ⑤
          TestUtils.getSecurityWithStockExchange(
              quoteReply.getSecurityCode(), quoteEntry.getStockExchange());

      AcceptQuoteRequest acceptQuoteRequest = AcceptQuoteRequest.newBuilder() ⑥
          .setAccessToken(accessToken)
          .setOrderType(OrderType.BUY)
          // .setAccountNumber(accountNumber)
          .setSecurityWithStockexchange(securityWithStockExchange)
          .setAmount(10)
          .setValidation(Validation.VALIDATE_ONLY) ⑦
          .setLimit(quoteEntry.getBuyPrice())
          .setQuoteReference(quoteEntry.getQuoteReference()) ⑧
          .build();
      OrderReply acceptQuote = orderServiceBlockingStub.acceptQuote(
acceptQuoteRequest); ⑨
      System.err.println(acceptQuote);
    }
  }
}
```

① Prepare get quote request

② Call get quote function

③ Get recieved quotes (it should be only one quote, because we requested data for one stock exchange)

④ Check if we have quote reference

⑤ Prepare security with stock exchange object

⑥ Preapare accept quote request

⑦ Set validation flag

⑧ Set to the request quote reference

⑨ Call accept quote function and wait for the result

Warning

By default **validation** parameter equals **WITHOUT_VALIDATION** and accept quote request will directly routed to the market. If you want only check a validity of the request, then you need to set **validation** parameter to **VALIDATE_ONLY**. An order will validated by the backend system, but will not be routed to the market. It's very useful for the test goals. Order number in the reply in this case is undefined.

Add order

To put new order to the long term marken is necessary to use **AddOrder** function. This function takes **AddOrderRequest** as an input parameter and returns OrderReply as a result. Typically this function is called with blocked stub, but it can also used with non blocking stub if more then one order need to executed in parallel.

The possible parameter combinations can be fetched from security information reply. There are defined order possibilities for defined markets.

Pro version of the ActiveTrader allowes to use additional flag **inactive**. If this flag set to the value **true** then order is placed to the system, but isn't routed to the market. With the help of ActivateOrder function an order can be activated.

Important

It's important to fill only parameters that need for the order execution

Example Java: addOrder()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private OrderReply addOrder() {
  SecurityCode securityCode = TestUtils.getSecurityCode("710000", SecurityCodeType.
WKN); ①
  StockExchange stockExchange = TestUtils.getStockExchange("OTC", null); ②
  SecurityWithStockExchange securityWithStockExchange =
      TestUtils.getSecurityWithStockExchange(securityCode, stockExchange); ③
  AddOrderRequest request = AddOrderRequest.newBuilder() ④
      .setAccessToken(accessToken)
      .setAccountNumber("123456789") ⑤
      .setSecurityWithStockexchange(securityWithStockExchange)
      .setValidation(Validation.VALIDATE_ONLY) ⑥
      .setAmount(1)
      .setOrderModel(OrderModel.LIMIT) ⑦
      .setLimit(1)
      .setOrderType(OrderType.BUY) ⑧
      .setCashQuotation(CashQuotation.KASSA)
      .setValidityDate(TestUtils.getDate(GregorianCalendar.getInstance()))
      .build();
  OrderReply addOrderReply = orderServiceBlockingStub.addOrder(request); ⑨
  System.err.println(addOrderReply);
  return addOrderReply;
}
```

① Prepare security code

② Prepare stock exchange

③ Prepare security with stock exchange

④ Prepare request

⑤ Set trading account

⑥ Set validation flag

⑦ Set order limit

⑧ Set order type

⑨ Process order add

Warning

By default **validation** parameter equals **WITHOUT_VALIDATION** and accept quote request will directly routed to the market. If you want only check a validity of the request, then you need to set **validation** parameter to **VALIDATE_ONLY**. An order will validated by the backend system, but will not be routed to the market. It's very useful for the test goals. Order number in the reply in this case is undefined.

Change order

To change order is necessary to have order number of existing order and an account with this order linked to. The order can be changed if the status of the orders allowes that and new state is allowed by the markt.

Useful

Sometimes can happens that order is changed / executed on the market, but status information is not delivered into the application.

Example Java: changeOrder()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private void changeOrder(Order order, TradingAccount account) {
  if (order!=null) {
    String orderNumber = order.getOrderNumber();
    if (orderNumber!=null && !orderNumber.isEmpty()) {
      ChangeOrderRequest changeOrderRequest = ChangeOrderRequest.newBuilder()
          .setAccessToken(accessToken)
          .setAccountNumber(account.getAccountNumber()) ①
          .setOrderNumber(order.getOrderNumber()) ②
          .setValidation(Validation.VALIDATE_ONLY) ③
          .setOrderModel(OrderModel.LIMIT) ④
          .setLimit(2)
          .build();

      OrderReply changeOrderReply = ⑤
          orderServiceBlockingStub.changeOrder(changeOrderRequest);
      System.out.println(changeOrderReply);
    }
  }
}
```

① Set account number

② Set order number

③ Set validation flag

④ Set new limit

⑤ Process change order

Warning

By default **validation** parameter equals **WITHOUT_VALIDATION** and accept quote request will directly routed to the market. If you want only check a validity of the request, then you need to set **validation** parameter to **VALIDATE_ONLY**. An order will validated by the backend system, but will not be routed to the market. It's very useful for the test goals.

Cancel order

To cancel order is necessary to have order number of existing order and an account with this order linked in. The order can be canceled if the status of the orders allowes that.

Example Java: cancelOrder()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private void cancelOrder(Order order, TradingAccount account) {
  if (order!=null) {
    String orderNumber = order.getOrderNumber();
    if (orderNumber!=null && !orderNumber.isEmpty()) {
      CancelOrderRequest cancelOrderRequest = CancelOrderRequest.newBuilder()
          .setAccessToken(accessToken)
          .setAccountNumber(account.getAccountNumber()) ①
          .setOrderNumber(order.getOrderNumber()) ②
          .setValidation(Validation.VALIDATE_ONLY) ③
          .build();

      OrderReply cancelOrderReply = ④
          orderServiceBlockingStub.cancelOrder(cancelOrderRequest);
      System.out.println(cancelOrderReply);
    }
  }
}
```

① Set account number

② Set order number

③ Set validation flag

④ Process change order

Warning

By default **validation** parameter equals **WITHOUT_VALIDATION** and accept quote request will directly routed to the market. If you want only check a validity of the request, then you need to set **validation** parameter to **VALIDATE_ONLY**. An order will validated by the backend system, but will not be routed to the market. It's very useful for the test goals.

Activate order

Activate order function used to activate inactive order and route an order to the market. This function available only in pro version of the ActiveTrader.

Example Java: activateOrder()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private void activateOrder(Order order, TradingAccount account) {
  if (order!=null) {
    String orderNumber = order.getOrderNumber();
    if (orderNumber!=null && !orderNumber.isEmpty()) {
      ActivateOrderRequest activateOrderRequest = ActivateOrderRequest.newBuilder()
          .setAccessToken(accessToken)
          .setAccountNumber(account.getAccountNumber()) ①
          .setOrderNumber(order.getOrderNumber()) ②
          .setValidation(Validation.VALIDATE_ONLY) ③
          .build();

      OrderReply changeOrderReply =  ④
          orderServiceBlockingStub.activateOrder(activateOrderRequest);
      System.out.println(changeOrderReply);
    }
  }
}
```

① Set account number

② Set order number

③ Set validation flag

④ Process activate order

Deactivate order

Deactivate order function used to deactivate active order and remove an order from the market. This function available only in pro version of the ActiveTrader.

Example Java: deactivateOrder()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private void deactivateOrder(Order order, TradingAccount account) {
  if (order!=null) {
    String orderNumber = order.getOrderNumber();
    if (orderNumber!=null && !orderNumber.isEmpty()) {
      DeactivateOrderRequest deactivateOrderRequest = DeactivateOrderRequest
.newBuilder()
          .setAccessToken(accessToken)
          .setAccountNumber(account.getAccountNumber()) ①
          .setOrderNumber(order.getOrderNumber()) ②
          .setValidation(Validation.VALIDATE_ONLY) ③
          .build();

      OrderReply changeOrderReply = ④
          orderServiceBlockingStub.deactivateOrder(deactivateOrderRequest);
      System.out.println(changeOrderReply);
    }
  }
}
```

① Set account number

② Set order number

③ Set validation flag

④ Process deactivate order

Order costs

BaFin Rules

The law allowes to execute orders only after requesting of the order costs with a possibility to decline order in the case if order costs are not accepted. That follows to request order cost before order execution. **The order execution without requesting of the order costs is not allowed**.

Processing of the order costs

You can request an information about estimated order costs in AddOrderRequest and AcceptQuoteRequest.

Warning

OrderCost contains **only estimated values**. The real values are depended from real execution quotes, time of the execution and other parameters.

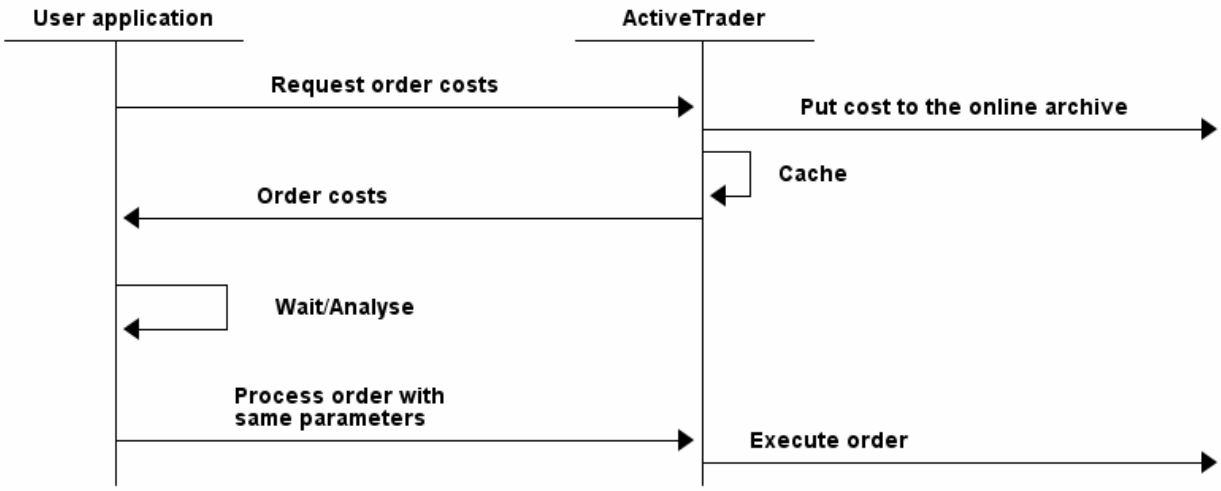
To request cost information is need to execute orders requests with Validation values **VALIDATE_WITH_TOTAL_COSTS**, **TOTAL_COST_ONLY** or **VALIDATE_WITH_DETAIL_COSTS**. In first and second cases only total or aggregated costs are calculated. In second case detail

information about costs is delivered. In all cases only validation of the request is happened, without real execution. To execute order you have to repeat same request with Validation **WITHOUT_VALIDATION** type.

Important

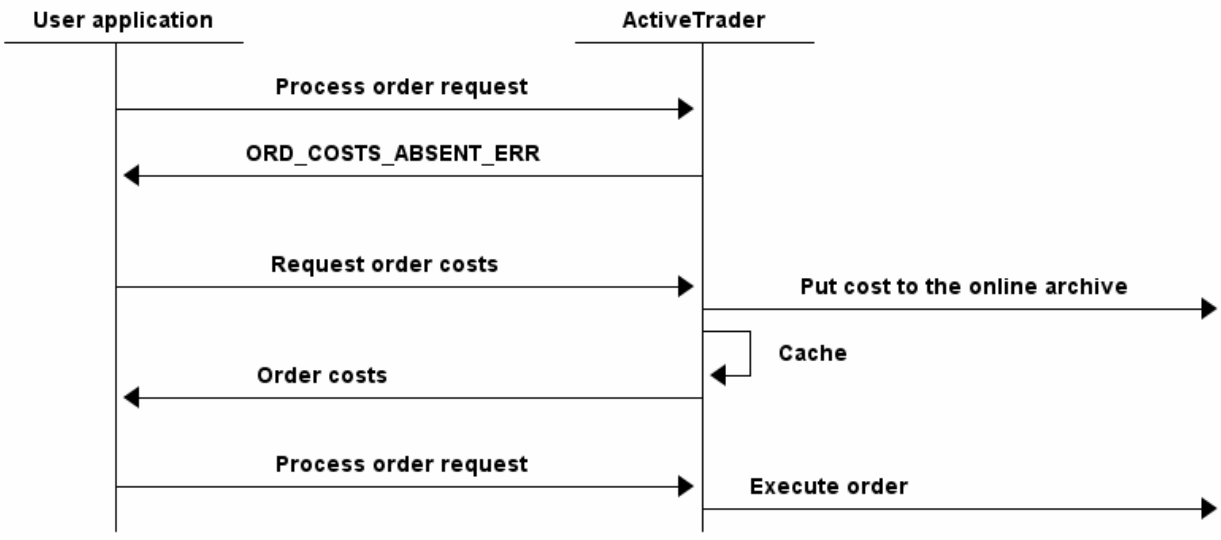
During prepartion phase **TOTAL_COST_ONLY** validation type can be replaced with **VALIDATE_WITH_TOTAL_COSTS** type.

All order costs can be checked in the online archive.



Error handling

If no costs are avialable or costs information is outdated then by the order execution will be return **ORD_COSTS_ABSENT_ERR** error. In such cases is necessary to repeat order costs request and order execution request.



Useful

To speed up order execution and save time by the order placement it's possible to prepare and validate order with **VALIDATE_WITH_TOTAL_COSTS** or **TOTAL_COST_ONLY** flags and some time later execute the same order with **WITHOUT_VALIDATION** flag. The host system will recognize parameters of the order and will forward order directly to the market. Otherwise to fullfil all BaFin requirements host system will request order costs before the order execution. In the each case the information about order costs will be placed into online archive before the order execution.

Warning

This request doesn't send order to the market, but validates only. As the result no real order is placed into the system.

Example Java: getOrderCosts()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private OrderReply getOrderCosts() {
  SecurityCode securityCode = TestUtils.getSecurityCode("710000", SecurityCodeType.
WKN); ①
  StockExchange stockExchange = TestUtils.getStockExchange("OTC", null); ②
  SecurityWithStockExchange securityWithStockExchange =
      TestUtils.getSecurityWithStockExchange(securityCode, stockExchange); ③
  AddOrderRequest request = AddOrderRequest.newBuilder() ④
      .setAccessToken(accessToken)
      .setAccountNumber("123456789") ⑤
      .setSecurityWithStockexchange(securityWithStockExchange)
      .setValidation(Validation.VALIDATE_WITH_TOTAL_COSTS) ⑥
      // .setValidation(Validation.VALIDATE_WITH_DETAIL_COSTS) ⑦
      .setAmount(1)
      .setOrderModel(OrderModel.LIMIT)
      .setLimit(1)
      .setOrderType(OrderType.BUY)
      .setCashQuotation(CashQuotation.KASSA)
      .setValidityDate(TestUtils.getDate(GregorianCalendar.getInstance()))
      .build();
  OrderReply addOrderReply = orderServiceBlockingStub.addOrder(request); ⑧

  OrderCosts orderCosts = addOrderReply.getOrderCosts(); ⑨
  double estimatedTotalCosts = orderCosts.getEstimatedTotalCosts();
  for (CategoryCost categoryCost : orderCosts.getCategorieCostsList()) {
    double totalSumAbsolute = categoryCost.getTotalSumAbsolute();
    //
  }
  System.err.println(addOrderReply);
  return addOrderReply;
}
```

① Prepare security code

② Prepare stock exchange

③ Prepare security with stock exchange

④ Prepare request

⑤ Set order parameters

⑥ Set validation with total costs

⑦ Alternative: Set validation with detail costs

⑧ Process order add

⑨ Extract order costs

Early request of the order costs

Useful

It's possible to prepare an order for the execution and keep an infomation about order costs for the future execution without additional checks. This information will be cached in the application and reused by the real order execution. No additional steps will be performed. All data about order costs will be (additionally) avialible in the online archive.

It's important to understand that sometime costs information can be absolette. In this case an order execution will be failed with the error **ORD_COSTS_NOT_REQUESTED**. In such cases is need to repeat order costs request and order execution request.

Example Java: addOrderWithCostsRequest()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private OrderReply addOrderWithCostsRequest() {
  SecurityCode securityCode = TestUtils.getSecurityCode("710000", SecurityCodeType.
WKN); ①
  StockExchange stockExchange = TestUtils.getStockExchange("OTC", null); ②
  SecurityWithStockExchange securityWithStockExchange =
      TestUtils.getSecurityWithStockExchange(securityCode, stockExchange); ③
  Builder requestBuilder = AddOrderRequest.newBuilder() ④
      .setAccessToken(accessToken)
      .setAccountNumber("123456789") ⑤
      .setSecurityWithStockexchange(securityWithStockExchange)
      .setAmount(1)
      .setOrderModel(OrderModel.LIMIT)
      .setLimit(1)
      .setOrderType(OrderType.BUY)
      .setCashQuotation(CashQuotation.KASSA)
      .setValidityDate(TestUtils.getDate(GregorianCalendar.getInstance()));
  AddOrderRequest costsRequest = requestBuilder
      .setValidation(Validation.TOTAL_COSTS_ONLY) ⑥
      .build();
  OrderReply costsReply = orderServiceBlockingStub.addOrder(costsRequest); ⑦

  if (costsReply.getError()!=Error.getDefaultInstance()) {
    System.err.print("Error: "+costsReply.getError());
    return costsReply;
  }

  // Can be later
  AddOrderRequest executionRequest = requestBuilder
      .setValidation(Validation.WITHOUT_VALIDATION) ⑧
      .build();

  OrderReply executionReply = orderServiceBlockingStub.addOrder(executionRequest); ⑨
  System.err.println(executionReply);
  return executionReply;
}
```

① Prepare security code

② Prepare stock exchange

③ Prepare security with stock exchange

④ Prepare request

⑤ Set order parameters

⑥ Set validation only with total costs

⑦ Process costs calulation request

⑧ Prepare real order add request (**Warning real execution**)

⑨ Process order execution

Example Java: acceptQuoteWithCostsRequest()

```
private final OrderServiceGrpc.OrderServiceBlockingStub orderServiceBlockingStub;

private void acceptQuoteWithCostsRequest() {
  SecurityCode securityCode = TestUtils.getSecurityCode("710000", SecurityCodeType.
WKN);
  StockExchange stockExchange = TestUtils.getStockExchange("OTC", null);
  SecurityWithStockExchange securityWithStockExchange = TestUtils
.getSecurityWithStockExchange(securityCode, stockExchange);

  AcceptQuoteRequest costsRequest = AcceptQuoteRequest.newBuilder() ①
      .setAccessToken(accessToken)
      .setOrderType(OrderType.BUY)
      .setSecurityWithStockexchange(securityWithStockExchange)
      .setAmount(10)
      .setValidation(Validation.TOTAL_COSTS_ONLY) ②
      //.setLimit(10) ③
      //.setQuoteReference() ④
      .build();
  OrderReply costsReply = orderServiceBlockingStub.acceptQuote(costsRequest); ⑤

  if (costsReply.getError()!=Error.getDefaultInstance()) {
    System.err.print("Error: "+costsReply.getError());
    return;
  }

  // Later
  QuoteRequest quoteRequest = QuoteRequest.newBuilder() ⑥
      .setAccessToken(accessToken)
      .setOrderType(OrderType.BUY)
      .setSecurityCode(securityCode)
      .setAmount(10)
      .addStockExchanges(stockExchange)
      .build();

  PerformanceCounter pcGetQuote = new PerformanceCounter("GetQuote");
  QuoteReply quoteReply = orderServiceBlockingStub.getQuote(quoteRequest); ⑦
  pcGetQuote.print();

  if (quoteReply.getError()==Error.getDefaultInstance()) {
    QuoteEntry quoteEntry = quoteReply.getPriceEntries(0); ⑧
    if (quoteEntry.getError()==Error.getDefaultInstance() &&
        !quoteEntry.getQuoteReference().isEmpty()) { ⑨

      AcceptQuoteRequest acceptQuoteRequest = AcceptQuoteRequest.newBuilder() ⑩
          .setAccessToken(accessToken)
          .setOrderType(OrderType.BUY)
```

```
            .setSecurityWithStockexchange(securityWithStockExchange)
            .setAmount(10)
            .setValidation(Validation.WITHOUT_VALIDATION) ⑪
            .setLimit(quoteEntry.getBuyPrice())
            .setQuoteReference(quoteEntry.getQuoteReference()) ⑫
            .build();
        OrderReply acceptQuote = orderServiceBlockingStub.acceptQuote(
acceptQuoteRequest); ⑬
        System.err.println(acceptQuote);
      }
    }
}
```

① Prepare costs request

② Set validation only with total costs

③ Limit is optional

④ Quote reference can be empty

⑤ Process costs calulation request

⑥ Prepare quote request

⑦ Process quote request

⑧ Get entry for the quote

⑨ Check quote

⑩ Prepare accept quote request

⑪ Set validation (**Warning real execution**)

⑫ Set quote reference

⑬ Process order execution

Errors

Error code	Target	Description
ACC_NOT_TRADING_ERR	Ordering	Process order with not trading account
ACC_NOT_VALID_ERR	Ordering	Process order with invalid account
ORD_ACTIVE_ERR	Ordering	Inactivate order in non pro version
ORD_AMOUNT_ERR	Ordering	Orders amount is not changable for selected order
ORD_DRIPPING_QUANTITY_ERR	Ordering	Dripping quantity error
ORD_COSTS_ERR	Ordering	Order costs information unavialable

Error code	Target	Description
ORD_COSTS_ABSENT_ERR	Ordering	Order costs information is not cached.
ORD_LIMIT_ERR	Ordering	Limit field is invalid
ORD_MODEL_ERR	Ordering	Order model is wrong
ORD_NOT_CANCELABLE_ERR	Ordering	Order is not cancelable
ORD_NOT_KNOWN_ERR	Ordering	Order is not known by the system
ORD_STOP_ERR	Ordering	Order stop is not changable
ORD_STOP_LIMIT_ERR	Ordering	Stop limit is not changable
ORD_TRALING_DISTANCE_ERR	Ordering	Traling distance is not changable
ORD_TRALING_LIMIT_TOLERANCE_ERR	Ordering	Traling limit tolerance is not changable
ORD_VALIDITY_DATE_ERR	Ordering	Validity date is not changable
ORD_WRONG_PRICE_TICKET_ERR	Ordering	Price ticket is not correct for this accept quote
ORD_QUOTE_ERR	Ordering	Quote is not valid
ORD_VALIDATION_ERR	Ordering	Selected validation can not be use for this request
ORD_COSTS_NOT_REQUESTED	Ordering	Costs are not requested before order or costs information is old
STK_NOT_FOUND_ERR	Stocks	Stock exchange is unknown
INTERNAL_ERR	System	Internal engine error
SYSTEM_ERR	System	GRPC error
UNSUPPORTED_OPERATION_ERR	System	Operation is not supported

Objects and types description

AcceptQuoteRequest

Accept quote request represents information about one order that should be placed on the short term market

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number
security_with_stockexchange	SecurityWithStockExchange	Security code with stock exchange

Field	Type	Description
order_type	OrderType	Order type. It should be relevant to the requested GetQuoteRequest
amount	double	Amount, It should be relevant to the requested GetQuoteRequest
limit	double	Limit
quote_reference	string	Quote reference from GetQuoteRequest
validation	Validation	Validation flag. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. If value is VALIDATE_ONLY then backend system doesn't send order actions to the market, but validate order parameters. If value is VALIDATE_WITH_TOTAL_COSTS then order will validated by backend system and request total costs for the order. If value is VALIDATE_WITH_DETAIL_COSTS then order will validated by backend system and request detail costs for the order.
risk_class_override	bool	Risk class override flag. If true then allowes override user risk class
target_market_override	bool	Target market override flag. If true then allowes override target market
tax_nontransparent_override	bool	Tax non trasparent override flag. If true then allowes override tax intransparesity
accept_additional_fees	bool	Accept additinal fees flag. If true then allowes accept non transparent fees

AccessTokenRequest

Access token request contains an access token data

Field	Type	Description
access_token	string	Access token

ActivateOrderRequest

Activate order request represents information for activation of one inactive order pro only

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number
order_number	string	Order number for that this changes should be accepted

Field	Type	Description
validation	Validation	Validation flag. This request allowes only **WITHOUT_VALIDATION** and **VALIDATE_ONLY** values. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. If value is VALIDATE_ONLY then backend system doesn't send order actions to the market, but validate order parameters.

AggregatedCosts

Aggregated costs contain an information about estimated costs for the selected order

Field	Type	Description
in_costs_absolute	double	In costs for the order
in_costs_relative	double	Percentage part of the in costs
in_costs_currency	string	Currency for the in costs
out_costs_absolute	double	Out costs for the order
out_costs_relative	double	Percentage part of the out costs
out_costs_currency	string	Currency for the out costs
instrument_costs_absolute	double	Instrument costs for the order
instrument_costs_relative	double	Percentage part of the instrument costs
instrument_costs_currency	string	Currency for the instrument costs
service_costs_absolute	double	Service costs for the order
service_costs_relative	double	Percentage part of the service costs
service_costs_currency	string	Currency for the servcie costs
subsidy_costs_absolute	double	Subsidy costs for the order
subsidy_costs_relative	double	Percentage part of the subsidy costs
subsidy_costs_currency	string	Currency for the subsidy costs
foreign_currency_costs_absolute	double	Foreign currency costs for the order
foreign_currency_costs_relative	double	Percentage part of the foreign currency costs

Field	Type	Description
foreign_currency_costs_currency	string	Currency for the foreign currency costs
performance_impact_absolute	double	Performance impact for the order
performance_impact_relative	double	Percentage part of the performance impact
performance_impact_currency	string	Currency for the performance impact
expected_amount	double	Expected amount estimated for the order
expected_amount_currency	string	Currency for the expected amount

AddOrderRequest

Add order request represents order data for the long term markets

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number which used for the execution
security_with_stockexchange	SecurityWithStockExchange	Security code with stock exchange
order_type	OrderType	Order type
amount	double	Amount of the securities
order_model	OrderModel	Order model
order_supplement	OrderSupplement	Order supplement
cash_quotation	CashQuotation	Cach quotation
validity_date	Date	Order validity date
limit	double	Limit value
stop	double	Stop value. This value can be used only together with StopMarket, StopLimit and OneCancelOter order models.
stop_limit	double	Stop limit used in the StopLimit and OneCancelOther order models
trailing_distance	double	Traling distance in traling notation units or empty value
trailing_notation	TrailingNotation	Trailing notation for the trailing orders
trailing_limit_tolerance	double	Trailing limit tolerance for the trailing orders
dripping_quantity	double	Dripping quantity pro only
position_id	string	Position id of the depot position. It used only for sale certainly securities from depot

Field	Type	Description
validation	Validation	Validation flag. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. If value is VALIDATE_ONLY then backend system doesn't send order actions to the market, but validate order parameters. If value is VALIDATE_WITH_TOTAL_COSTS then order will validated by backend system and request total costs for the order. If value is VALIDATE_WITH_DETAIL_COSTS then order will validated by backend system and request detail costs for the order.
risk_class_override	bool	Risk class override flag. If true then allowes override user risk class
target_market_override	bool	Target market override flag. If true then allowes override target market
tax_nontransparent_override	bool	Tax non trasparent override flag. If true then allowes override tax intransparesity
accept_additional_fees	bool	Accept additinal fees flag. If true then allowes accept non transparent fees
closed_realestate_fond_override	bool	Closed realestate fond override. If true then allowes sell fonds over fds
inactive	bool	Inactive order flag, pro only. If true then order market as inactive and don't routed to the marker. To activate order please use ActivateOrder function

CancelOrderRequest

Cancel order request represents canceling information for one order on the market or one inactive order

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number
order_number	string	Order number for that this changes should be accepted
validation	Validation	Validation flag. This request allowes only **WITHOUT_VALIDATION** value. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. Otherwise request will fail.

CashQuotation

Cash quotation is additional parameter of the order

Field	Description
NOTHING	Quotation is not defined
KASSA	Kassa quotation
AUCTION	Auction quotation
OPENING	Opening quotation
INTRADAY	Intraday quotation
CLOSING	Close quotation

CategoryCost

Represents one category cost.

Field	Type	Description
category_id	string	Category id
category_label	string	Human redable category label
total_sum_absolute	double	Total absolute sum of the children values
total_sum_relative	double	Total relative sum of the children values
currency	string	Currency for absolute sum
detail_costs	List of DetailCost	List of child values or detailed information.

ChangeOrderRequest

Change order request contains parameters for the order change. Be careful: not all combinations are possible.

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number
order_number	string	Order number for that this changes should be accepted
limit	double	New limit, shouldn't filled for market order
stop	double	New stop value
stop_limit	double	New stop limit value
amount	double	New amount
validity_date	Date	New validity date
order_model	OrderModel	New order model
order_supplement	OrderSupplement	New order supplement
dripping_quantity	double	Dripping quantity pro only

Field	Type	Description
validation	Validation	Validation flag. This request allowes only **WITHOUT_VALIDATION** and **VALIDATE_ONLY** values. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. If value is VALIDATE_ONLY then backend system doesn't send order actions to the market, but validate order parameters.
trailing_distance	double	New trailing distance
trailing_limit_tolerance	double	New traling limit tolerance

CurrencyRateReply

Returns currency rate

Field	Type	Description
currency_from	string	Source currency
currency_to	string	Target currency
currency_rate	double	Currency rate
error	Error	Error information if occuirs

CurrencyRateRequest

CurrencyRateRequest used for the determination of the currency rate from one currency rate to second currency. Results depends from user market data abonimentand can be realtime or delayed.

Field	Type	Description
access_token	string	Access token
currency_from	string	Source currency
currency_to	string	Target currency

Date

Represents a whole calendar date, e.g. date of birth. The time of day and time zone are either specified elsewhere or are not significant. The date is relative to the Proleptic Gregorian Calendar. The day may be 0 to represent a year and month where the day is not significant, e.g. credit card expiration date. The year may be 0 to represent a month and day independent of year, e.g. anniversary date. Related types are [google.type.TimeOfDay][google.type.TimeOfDay] and `google.protobuf.Timestamp`.

Field	Type	Description
year	int32	Year of date. Must be from 1 to 9999, or 0 if specifying a date without a year.
month	int32	Month of year. Must be from 1 to 12.
day	int32	Day of month. Must be from 1 to 31 and valid for the year and month, or 0 if specifying a year/month where the day is not significant.

DeactivateOrderRequest

Deactivate order request represents information for deactivation of one active order pro only

Field	Type	Description
access_token	string	Access token
account_number	string	Trading account number
order_number	string	Order number for that this changes should be accepted
validation	Validation	Validation flag. This request allowes only **WITHOUT_VALIDATION** and **VALIDATE_ONLY** values. If value is WITHOUT_VALIDATION then backend system sends order actions directly to the market. If value is VALIDATE_ONLY then backend system doesn't send order actions to the market, but validate order parameters

DepotEntries

Field	Type	Description
account	TradingAccount	Trading account
entries	List of DepotEntry	Depot entries list
error	Error	Error information if occuirs

DepotEntry

Depot entry contains information about one security in the depot. This entry combines data from one or more depot positions

Field	Type	Description
security_code	SecurityCode	Security code
positions	List of DepotPosition	List of linked depot positions. This list contains at least one element
effective_amount	double	Effective amount
scheduled_amount	double	Scheduled amount
total_amount	double	Total amount of the securities

Field	Type	Description
sell_possible	bool	True if sell possible for this entry or false otherwise. This value can be true only if all child positions have sell_possible = true
unit_note	UnitNote	Unit note
blocked	bool	True if entry is blocked or false otherwise
purchase_quotation	double	Purchase quotation or NaN if not defined
purchase_currency	string	Purchase currency or empty value if not defined
purchase_currency_rate	double	Purchase currency rate or NaN if not defined
open_sales	double	Open sales

DepotPosition

Depot posiotion contains information about one position in the depot

Field	Type	Description
amount	double	Ammount of the securities
position_id	string	Position identification
sell_possible	bool	True if sell of the position is possible or false otherwise
unit_note	UnitNote	Unit note
blocked	bool	True if entry is blocked or false otherwise
purchase_quotation	double	Purchase quotation or NaN if not defined. Currently this field ALWAYS undefined, reserved for future use
purchase_currency	string	Purchase currency or empty value if not defined
purchase_currency_rate	double	Purchase currency rate or NaN if not defined

DetailCost

Detail cost contains one entry for the selected category

Field	Type	Description
detail_id	string	Detail id
detail_label	string	Human redable detail label
value_absolute	double	Absolute value for this entry
value_relative	double	Relative value for this entry
currency	string	Currency for this entry

Field	Type	Description
detail_type	string	Specific entry type

Empty

Empty represents absent parameter or result

Field	Type	Description

Error

Error object

Field	Type	Description
code	string	Error code
message	string	Error message

LimitToken

Limit token represents a possibility to trade on the short term markets (AcceptQuote) and long term markets (AddOrder)

Field	Description
LIMIT_AND_QUOTE	AcceptQuote and AddOrder possible
QUOTE_ONLY	AcceptQuote only possible
LIMIT_ONLY	AddOrder only possible

LoginReply

Login reply provides information that need for the access to the TAPI

Field	Type	Description
access_token	string	Access token is used in each request by the access to the TAPI.
error	Error	Error information if occuirs

LoginRequest

Login request provides data for initial access to the TAPI

Field	Type	Description
secret	string	Secret is user defined access string. It's not possible to restore this secret directly. See double MD5 hash logic + salt

LogoutReply

Field	Type	Description
error	Error	Error information if occuirs

LogoutRequest

Field	Type	Description
access_token	string	Access token to invalidate.

Order

Order represent one order object

Field	Type	Description
security_with_stockexchange	SecurityWithStockExchange	Security with stock echange
order_type	OrderType	Order type
order_number	string	Order number
amount	double	Amount of the securities
order_model	OrderModel	Order model
order_supplement	OrderSupplement	Order supplement
cash_quotation	CashQuotation	Cache quotation
executed_amount	double	Executed amount
order_status	OrderStatus	Order status
status_timestamp	Timestamp	Date and time of the order status
validity_date	Date	Validity date of the order
limit	double	Limit value. Used as the order limit for all Limit order model with exception of the StopLimit order model. For this order model please use stop limit field.
stop	double	Stop value. This value can be used only together with StopMarket, StopLimit and OneCancelOter order models.
stop_limit	double	Stop limit value Can be used only tigether with the StopLimit order model. The meaning of the value is limit of the order after stop.
trailing_distance	double	Traling distance in traling notation units or empty value

Field	Type	Description
trailing_notation	TrailingNotation	Trailing notation for the trailing orders
trailing_limit_tolerance	double	Trailing limit tolerance for the trailing orders
dripping_quantity	double	Dripping quantity pro only
trading_partner_name	string	Trading partner name
execution_quote	double	Execution quote for the executed amount
unique_id	string	Unique id of the order. Used for the order matching. In the pro version of the ActiveTrader order_number can be changed after activation / deactivation. All order activities need actual or delivered form the system order_number. Typical scenario is to map and update unique id to the order number delivered from the API.

OrderBookEntry

Orderbook entry contains data about one level of the order book. This is bid price, ask price, bid volume, ask volume. Entries with lower index have lower ask and higer bid prices.

Field	Type	Description
bid_price	double	Bid price
ask_price	double	Ask price
bid_volume	double	Bid volume
ask_volume	double	Ask volume

OrderCosts

Order costs represents information about order action estimated costs. This information is only **estimated values** and is depended from real execution quotes, time, etc.

Field	Type	Description
estimated_total_costs	double	Estimated total cost for order action
cost_id	string	Reference backend cost id
categorie_costs	List of CategoryCost	List of the cost categories. Filled only by validation request with detailed information.
aggregated_costs	AggregatedCosts	Aggregated costs for the order. Filled only by validation request with validation information.

OrderModel

Order model represents possible orders

Field	Description
NO_ORDER_MODEL	Order model absent
MARKET	Market order
LIMIT	Limit order
STOP_MARKET	Stop market order.
STOP_LIMIT	Stop limit order
ONE_CANCELS_OTHER_MARKET	One cancels other market order
ONE_CANCELS_OTHER_LIMIT	One cancels other limit order
TRAILING_STOP_MARKET	Trailing stop market order
TRAILING_STOP_LIMIT	Trailing stop limit order

OrderReply

Order reply represents result of the add order or accept quote requests

Field	Type	Description
account	TradingAccount	Trading account
order	Order	Result order
order_costs	OrderCosts	Order costs. This field contains data if order costs are requested
error	Error	Error information if occuirs

Orders

Orders represent pushed information about orders from one trading accounts

Field	Type	Description
account	TradingAccount	Trading account
orders	List of Order	List of the orders
error	Error	Error information if occuirs

OrderStatus

Order status represents status of the order

Field	Description
NO_ORDER_STATUS	Status is not defined
NEW	New order

Field	Description
OPEN	Open order
EXECUTED	Fully executed order
PARTIALLY_EXECUTED	Partially executed order
CANCELED	Canceled order
CANCELED_FORCED	Canceled forced order
CANCELED_NOTED	Canceling noted order
CANCELED_TIMEOUT	Canceling timeout order
CHANGED	Changed order
CHANGED_NOTED	Changing noted order
INACTIVE	Inactive order pro only
INACTIVE_NOTED	Inactivation noted order pro only
STORNO	Storno order

OrderSupplement

Order supplement is additional parameter by order definition

Field	Description
NORMAL	Normal order supplement
IMMIDIATE_OR_CANCEL	Immidiate or cancel order supplement
FILL_OR_KILL	Fill or kill order supplement
ICEBERG	Icesberg order supplement. Allowes to delivery amount in portions. pro only
MARKET_PRICE	Market place order supplement

OrderType

Order type represents different buy / sell order possibilities

Field	Description
NO_ORDER_TYPE	Order type is not defined
BUY	Buy order type
SELL	Sell order type
SHORT_SELL	Short sell order type
SHORT_COVER	Short cover order type. This type is not allowed as input parameter
FORCED_COVER	Rorced cover order type. This type is not allowed as input parameter

PriceEntry

Field	Type	Description
open_price	double	Open price
close_price	double	Close price
high_price	double	High price
low_price	double	Low price
volume	double	Volume, can not be filled
open_time	Timestamp	Open time
close_time	Timestamp	Close time

QuoteEntry

Quote entry represents one answer with quote information from one stock exchange This reply contains both buy and sell price. The quote reference can be used only with requested order type. Second value is present only for information goals.

```
order type: BUY ->  quote reference for buy price and reference,
                    sell price and reference are informative only
order type: SELL -> quote reference for sell price and reference,
                    buy price and reference are informative only
```

Field	Type	Description
stock_exchange	StockExchange	Stock exchange where infomation was requested
buy_price	double	Buy price
buy_volume	double	Buy volume
sell_price	double	Sell price
sell_volume	double	Sell volume
last_price	double	Last price
last_volume	double	Last volume
last_time	Timestamp	Date and time of the last price
currency	string	Currency
quote_reference	string	Quote reference. Used for the accept quite request. Can be empty if accept quote is not possible.
order_type	OrderType	Used by call order type
error	Error	Error information if occuirs

QuoteReply

Quote reply represents data with quote answers from requested markets

Field	Type	Description
security_code	SecurityCode	Security code
order_type	OrderType	Order type
price_entries	List of QuoteEntry	List of the quites
error	Error	Error information if occuirs

QuoteRequest

Quote request represents data to get information about actual quotes on the selected makets

Field	Type	Description
access_token	string	Access token
security_code	SecurityCode	Security code
order_type	OrderType	Order type. Only BUY or SELL are allowed
amount	double	Amount of securities. Relevant to the short term markets
stock_exchanges	List of StockExchange	List of stock exchanges

SecurityChangedField

SecurityChangedField represents information about changed fields during marked data event. Not all field can be initialized. There are only changed fields. Some fields can be changed, but have undefined state for example NaN.

Field	Description
NONE	No data
LAST_PRICE	Price of the last trade
LAST_VOLUME	Volume last trade
LAST_DATE_TIME	Last quote date and time
TODAY_NUM_TRADES	Today number of tradings
TODAY_VOLUME	Today volume
ASK_PRICE	Last ask price
ASK_VOLUME	Volume last ask
ASK_TIME	Time of the last ask
BID_PRICE	Last bid price
BID_VOLUME	Volume last bid
BID_TIME	Time of the last bid
PREVIOUS_PRICE	Quote of the previous trading day
PREVIOUS_DATE	Date of the previous trading day
RELATIVE_DIFF	Relative difference to the previous day

Field	Description
ABS_DIFF	Absolute difference to the previous day
HIGH_PRICE	Highest price
LOW_PRICE	Lowest price
OPEN_PRICE	Price at opening
WEEK_HIGH_PRICE	Highest price of the previous week
DATE_WEEK_HIGH	Date of highest price of the previous week
WEEK_LOW_PRICE	Lowest price of the previous week
DATE_WEEK_LOW	Date of lowest price of the previous week
MONTH_HIGH_PRICE	Highest price of the previous month
DATE_MONTH_HIGH	Date of highest price of the previous month
MONTH_LOW_PRICE	Lowest price of the previous month
DATE_MONTH_LOW	Date of lowest price of the previous month
YEAR_HIGH_PRICE	Highest price of the current year
DATE_YEAR_HIGH	Date of the highest price of the current year
YEAR_LOW_PRICE	Lowest price of the current year
DATE_YEAR_LOW	Date of the lowest price of the current year
LAST_ADDENDUM	Addendum of the last price.
TRADING_PHASE	Trading phase
INDICATIVE_PRICE	Indicative price
PREVALENCE_VOLUME	Trading volume corresponding to the last price

SecurityClass

Represents information about security class

Field	Description
NO_SECURITY_CLASS	Security class is undefined on unknown
STOCK	Stock security class
BOND	Bond security class
CERTIFICATE	Certificate security class
PRECIOUS_METAL	Precious metal security class
PARTICIPATION_CERTIFICATE	Participation certificate security class
FUNDS	Funds security class
MUTUAL_FUNDS	Mutual funds security class
WARRANT	Warrants security class
FUTURE	Futures security class

Field	Description
INDEX	Indexies security class
OTHERS	Other securities security class
FUTURE_C1	Future c1 security class
FUTURE_C2	Future c2 security class
FUTURE_C3	Future c3 security class
TRACKERS	Trackers security class
CURRENCY	Currency security class
COMMODITY	Commodity security class

SecurityCode

Field	Type	Description
code	string	Security code
code_type	SecurityCodeType	Security code type (WKN, ISIN, etc)

SecurityCodeType

Represents information about security code type Some of the types are refer to the market data provider (FactSet)

Field	Description
NO_CODE_TYPE	Unknown code type
WKN	WKN code type
ISIN	ISIN code type
ID_NOTATION	Factset id notation
ID_OSI	Factset id osi
ID_INSTRUMENT	Factset id instrument
MNEMONIC	Mnemonic or symbol
MNEMONIC_US	US Mnemonic or symbol

SecurityInfoReply

Returns security information

Field	Type	Description
name	string	Security name
security_class	SecurityClass	Security class
security_codes	List of SecurityCode	Security codes with security type (WKN, ISIN, etc)

Field	Type	Description
stock_exchange_infos	List of SecurityStockExchangeInfo	Stockexchange info (stock exchange, name)
unit_note	UnitNote	Unit note
error	Error	Error information if occuirs

SecurityInfoRequest

Requests security information for security code

Field	Type	Description
access_token	string	Access token
security_code	SecurityCode	Security code with security type (WKN, ISIN)

SecurityMarketDataReply

Returns market data information

Field	Type	Description
security_with_stockexchange	SecurityWithStockExchange	Security with stockExchange object (security code, stock exchange)
changed_fields	List of SecurityChangedField	Security fields, which were changed
currency	string	Currency
last_price	double	Fields Price of the last trade
last_volume	double	Volume last trade
last_date_time	Timestamp	Last quote date and time
today_num_trades	int32	Today number of tradings
today_volume	double	Today volume
ask_price	double	Last ask price
ask_volume	double	Volume last ask
ask_time	Timestamp	Time of the last ask
bid_price	double	Last bid price
bid_volume	double	Volume last bid
bid_time	Timestamp	Time of the last bid
previous_price	double	Quote of the previous trading day
previous_date	Date	Date of the previous trading day
relative_diff	double	Relative difference to the previous day

Field	Type	Description
abs_diff	double	Absolute difference to the previous day
high_price	double	Highest price
low_price	double	Lowest price
open_price	double	Price at opening
week_high_price	double	Highest price of the previous week
date_week_high	Date	Date of highest price of the previous week
week_low_price	double	Lowest price of the previous week
date_week_low	Date	Date of lowest price of the previous week
month_high_price	double	Highest price of the previous month
date_month_high	Date	Date of highest price of the previous month
month_low_price	double	Lowest price of the previous month
date_month_low	Date	Date of lowest price of the previous month
year_high_price	double	Highest price of the current year
date_year_high	Date	Date of the highest price of the current year
year_low_price	double	Lowest price of the current year
date_year_low	Date	Date of the lowest price of the current year
last_addendum	string	Addendum of the last price.
trading_phase	TradingPhase	Trading phase
indicative_price	double	Indicative price
prevalence_volume	double	Trading volume corresponding to the last price
error	Error	Error information if occuirs

SecurityMarketDataRequest

Requests market data values for defined security with stockexchange

Field	Type	Description
access_token	string	Access token
security_with_stockexchange	SecurityWithStockExchange	Security with stockExchange object (security code, stock exchange)
currency	string	Currency

SecurityOrderBookReply

SecurityOrderBookReply represents information with orderbook market data. Currently this information is avialable for the Xetra stock exchange for the accepted for the second level market data instruments

Field	Type	Description
security_with_stockexchange	SecurityWithStockExchange	Security with stock exchange object (security code, stock exchange)
currency	string	Currency of the order book entries
order_book_entries	List of OrderBookEntry	List of the order book entries
error	Error	Error information if occuirs

SecurityOrderBookRequest

Requests orderbook data for security with stockexchange

Field	Type	Description
access_token	string	Access token
security_with_stockexchange	SecurityWithStockExchange	Security with stockExchange object (security code, stock exchange)
currency	string	Requested currency. If not filled used default currency. Otherwise values will be converted to requested currency.

SecurityPriceHistoryReply

Returns history data for defined security

Field	Type	Description
security_with_stockexchange	SecurityWithStockExchange	Security with stockExchange object (security code, stock exchange)
currency	string	Currency
price_entries	List of PriceEntry	List of the price entries
error	Error	Error information if occuirs

SecurityPriceHistoryRequest

Requests history data for one security on one stockexchange in intraday or historical format

Field	Type	Description
access_token	string	Access token
security_with_stockexchange	SecurityWithStockExchange	Security with stockExchange object (security code, stock exchange)
days	int32	Amount of the day in the past. This value should be positive. Maximal value for the intraday resolution is 15.
time_resolution	TimeResolution	Time resolution for the data

SecurityStockExchangeInfo

Security stock exchange info represents trading information about one security on the one market

Field	Type	Description
stock_exchange	StockExchange	Stockexchange (id und issuer)
buy_limit_token	LimitToken	Possible limit token for buy orders
sell_limit_token	LimitToken	Possible limit token for sell orders
buy_trading_types	List of TradingPossibility	Buy trading data (order models, order supplements, trailing notations)
sell_trading_types	List of TradingPossibility	Sell trading data (order models, order supplements, trailing notations)
maximal_order_date	Date	Maximal order validity date
short_mode	ShortMode	Short selling mode

SecurityWithStockExchange

Field	Type	Description
security_code	SecurityCode	Security code object
stock_exchange	StockExchange	Stock exchange object

Table 6. AccessService

Function	Description
LoginReply = Login(LoginRequest)	Validates client by the TAPI and gets access data
LogoutReply = Logout(LogoutRequest)	Invalidates client by the TAPI and gets logout result

Table 7. AccountService

Function	Description
TradingAccounts = GetTradingAccounts(AccessTokenRequest)	Gets trading accounts **@return** TradingAccounts List of trading accounts
TradingAccountInformation ⇐ StreamTradingAccount(TradingAccountRequest)	Subscribes one trading account for updates **@param** TradingAccount Trading account for push **@stream** TradingAccountInformation Specific information for subscribed account (balance, kredit line, etc.)

Function	Description
TradingAccountTransactions ⇐ StreamTradingAccountTransactions(TradingAccountRequest)	Subscribes one trading account for the transactions updates **@param** TradingAccount Trading account for push **@stream** TradingAccountInformation Transactions list for subscribed account

Table 8. DepotService

Function	Description
DepotEntries ⇐ StreamDepot(TradingAccountRequest)	Subscribes one trading account for the depot data updates **@param** TradingAccount Trading account for push **@stream** DepotEntries depot entries linked to the account
Empty = UpdateDepot(TradingAccountRequest)	Initiates depot update action. All changes come by the StreamDepot subscription. This function doesn't wait for the action result. **@param** TradingAccount Trading account for update

Table 9. OrderService

Function	Description
Orders ⇐ StreamOrders(TradingAccountRequest)	Subscribes one trading account for orders updates **@param** TradingAccount Trading account for push **@stream** Orders Orders list for seleted account
Empty = UpdateOrders(TradingAccountRequest)	Initiates orders update action. All changes come by the StreamOrders subscription. This function doesn't wait for the action result. **@param** TradingAccount Trading account for update
QuoteReply = GetQuote(QuoteRequest)	Request market quote for the selected security on the selected stock exchanges. **@param** QuoteRequest quote request with interested security and stock exchanges **@return** QuoteReply quote reply with quotes
OrderReply = AcceptQuote(AcceptQuoteRequest)	Sends accept quote order request to the short term market **@param** AcceptQuoteRequest accept quote request with order parameters **@return** OrderReply result order or error

Function	Description
OrderReply = AddOrder(AddOrderRequest)	Sends long term order to the market **@param** AddOrderRequest order request with order parameters **@return** OrderReply result order or error
OrderReply = ChangeOrder(ChangeOrderRequest)	Sends order change request to the market **@param** ChangeOrderRequest changed order request with order parameters **@return** OrderReply result order or error
OrderReply = CancelOrder(CancelOrderRequest)	Sends order cancel request to the market **@param** CancelOrderRequest cancel order request with order reference **@return** OrderReply result order or error
OrderReply = ActivateOrder(ActivateOrderRequest)	Sends order activate request to the market. pro only **@param** ActivateOrderRequest activate order request with order parameters **@return** OrderReply result order or error
OrderReply = DeactivateOrder(DeactivateOrderRequest)	Sends order deactivate request to the market. pro only **@param** DeactivateOrderRequest deactivate order request with order parameters **@return** OrderReply result order or error

Table 10. SecurityService

Function	Description
SecurityInfoReply = GetSecurityInfo(SecurityInfoRequest)	Gets security information about security **@param** SecurityInfoRequest Request object with interested security **@return** SecurityInfoReply Complete information about security
SecurityMarketDataReply ⇐ StreamMarketData(SecurityMarketDataRequest)	Subscribes security with stock exchange for market data updates **@param** SecurityMarketDataRequest Market data request with interested security and stock exchange **@stream** SecurityMarketDataReply Reply with all market data values

Function	Description
SecurityOrderBookReply ⇐ StreamOrderBook(SecurityOrderBookRequest)	Subscribes security with stock exchange for orderbook updates **@param** SecurityOrderBookRequest Orderbook data request with interested security and stock exchange **@stream** SecurityOrderBookReply Reply with all orderbook values
CurrencyRateReply ⇐ StreamCurrencyRate(CurrencyRateRequest)	Subscribes for currency rate from one currency to another currency. **@param** SecurityOrderBookRequest currency rate request with interested currencies from/to **@stream** CurrencyRateReply reply with currency rate
SecurityPriceHistoryReply = GetSecurityPriceHistory(SecurityPriceHistoryRequest)	Requests history data for one security on one stockexchange in intraday or historical format **@param** SecurityPriceHistoryRequest Data with security, stockexchange, how many days and resolution **@return** SecurityPriceHistoryReply List of the historical quotes or an error

Table 11. StockExchangeService

Function	Description
StockExchangeDescriptions = GetStockExchanges(AccessTokenRequest)	Gets predefined stockexchages **@return** StockExchangeDescriptions list of stock exchange informations
StockExchangeDescription = GetStockExchange(StockExchangeRequest)	Gets specific stock exchange **@param** StockExchange Requested stock exchange **@return** StockExchangeDescription Stock exchange information

ShortMode

Short mode gives information amout security shortability on the selected market.

Field	Description
NO_SHORT_MODE	Undefined short selling
YES	Short selling is possible
NO	No short selling
TEMPORARY_NO	Temporary no short selling
INTRADAY	Intraday short selling
OVERNIGHT	Overnight short selling

Field	Description
INTRADAY_AND_OVERNIGHT	Intraday and Overnight short selling

StockExchange

Stock exchange data

Field	Type	Description
id	string	Stock exchange id
issuer	string	Stock exchange issuer. Can be null

StockExchangeDescription

Field	Type	Description
stock_exchange_info	StockExchangeInfo	Stock exchange information
error	Error	Error information if occuirs

StockExchangeDescriptions

Field	Type	Description
stock_exchange_infos	List of StockExchangeInfo	List with stock exchange information
error	Error	Error

StockExchangeInfo

Field	Type	Description
stockExchange	StockExchange	Stock exchange object
name	string	Stock exchange name

StockExchangeRequest

Stock exchange request contains data that need to request stock exchange related data

Field	Type	Description
access_token	string	Access token
stock_exchange	StockExchange	Stock exchange

TimeResolution

Time resolution represents information about price aggregation

Field	Description
NO_RESOLUTION	Undefined resolution
TICK	Tick resolution. **intraday**. Be careful, in same cases there is a lot of data.
SECOND	Second resolution **intraday**
MINUTE	Minute resolution **intraday**
HOUR	Hour resolution **intraday**
DAY	Day resolution **historic**

Timestamp

A Timestamp represents a point in time independent of any time zone or calendar, represented as seconds and fractions of seconds at nanosecond resolution in UTC Epoch time. It is encoded using the Proleptic Gregorian Calendar which extends the Gregorian calendar backwards to year one. It is encoded assuming all minutes are 60 seconds long, i.e. leap seconds are "smeared" so that no leap second table is needed for interpretation. Range is from 0001-01-01T00:00:00Z to 9999-12-31T23:59:59.999999999Z. By restricting to that range, we ensure that we can convert to and from RFC 3339 date strings. See https://www.ietf.org/rfc/rfc3339.txt.

Examples

Example 1: Compute Timestamp from POSIX `time()`.

```
Timestamp timestamp;
timestamp.set_seconds(time(NULL));
timestamp.set_nanos(0);
```

Example 2: Compute Timestamp from POSIX `gettimeofday()`.

```
struct timeval tv;
gettimeofday(&tv, NULL);
```

```
Timestamp timestamp;
timestamp.set_seconds(tv.tv_sec);
timestamp.set_nanos(tv.tv_usec * 1000);
```

Example 3: Compute Timestamp from Win32 `GetSystemTimeAsFileTime()`.

```
FILETIME ft;
GetSystemTimeAsFileTime(&ft);
UINT64 ticks = (((UINT64)ft.dwHighDateTime) << 32) | ft.dwLowDateTime;
```

```
// A Windows tick is 100 nanoseconds. Windows epoch 1601-01-01T00:00:00Z
// is 11644473600 seconds before Unix epoch 1970-01-01T00:00:00Z.
Timestamp timestamp;
timestamp.set_seconds((INT64) ((ticks / 10000000) - 11644473600LL));
timestamp.set_nanos((INT32) ((ticks % 10000000) * 100));
```

Example 4: Compute Timestamp from Java `System.currentTimeMillis()`.

```
long millis = System.currentTimeMillis();
```

```
Timestamp timestamp = Timestamp.newBuilder().setSeconds(millis / 1000)
    .setNanos((int) ((millis % 1000) * 1000000)).build();
```

Example 5: Compute Timestamp from current time in Python.

```
timestamp = Timestamp()
timestamp.GetCurrentTime()
```

JSON Mapping

In JSON format, the Timestamp type is encoded as a string in the RFC 3339 format. That is, the format is "{year}-{month}-{day}T{hour}:{min}:{sec}[.{frac_sec}]Z" where {year} is always expressed using four digits while {month}, {day}, {hour}, {min}, and {sec} are zero-padded to two digits each. The fractional seconds, which can go up to 9 digits (i.e. up to 1 nanosecond resolution), are optional. The "Z" suffix indicates the timezone ("UTC"); the timezone is required. A proto3 JSON serializer should always use UTC (as indicated by "Z") when printing the Timestamp type and a proto3 JSON parser should be able to accept both UTC and other timezones (as indicated by an offset).

For example, "2017-01-15T01:30:15.01Z" encodes 15.01 seconds past 01:30 UTC on January 15, 2017.

In JavaScript, one can convert a Date object to this format using the standard toISOString() method. In Python, a standard `datetime.datetime` object can be converted to this format using `strftime` with the time format spec '%Y-%m-%dT%H:%M:%S.%fZ'. Likewise, in Java, one can use the Joda Time's `ISODateTimeFormat.dateTime()` to obtain a formatter capable of generating timestamps in this format.

Timestamp represents date + time format

Field	Type	Description
seconds	int64	Represents seconds of UTC time since Unix epoch 1970-01-01T00:00:00Z. Must be from 0001-01-01T00:00:00Z to 9999-12-31T23:59:59Z inclusive.
nanos	int32	Non-negative fractions of a second at nanosecond resolution. Negative second values with fractions must still have non-negative nanos values that count forward in time. Must be from 0 to 999,999,999 inclusive.

TradingAccount

TradingAccount is trading account connected to the depot

Field	Type	Description
account_number	string	Account number
depot_number	string	Depot number
name	string	Name of the account owners
tradable	bool	If account is tradable then true or false otherwise

TradingAccountInformation

TradingAccountInformation contains specific for this account information

Field	Type	Description
account	TradingAccount	Trading account
balance	double	Account balance
credit_limit	double	Credit limit information
buying_power	double	Buying power
credit_limit_intraday	double	Credit limit intraday information, pro only
buying_power_intraday	double	Buyng power intraday, pro only
error	Error	Error information if occuirs

TradingAccountRequest

Trading account request contains trading account related data

Field	Type	Description
access_token	string	Access token
trading_account	TradingAccount	Trading account

TradingAccounts

TradingAccounts contains account information from current session

Field	Type	Description
accounts	List of TradingAccount	List of trading accounts
error	Error	Error information if occuirs

TradingAccountTransactions

TradingAccountTransactions contains account transactions

Field	Type	Description
account	TradingAccount	Trading account
transactions	List of Transaction	List of transactions
error	Error	Error information if happened

TradingPhase

Represents stock exchange trading phase. Some values are refer to the Xetra stock exchange

Field	Description
NONE	Unknown trading phase
PRETRADE	Pretrade trading phase
POSTTRADE	Posttrade trading phase
START	Start trading phase
END	End trading phase
VOLA	Vola trading phase
OCALL	OCall trading phase
ICALL	ICall trading phase
CCALL	CCall trading phase
TRADE	Trade trading phase
TRADE_INDICATIVE	Trade indicative trading phase
TRADE_BEST_BID_ASK	Trade best bid / ask trading phase
TRADE_AUCTION_NO_INDICATIVE	Trade auction, but not indicative trading phase

TradingPossibility

Trading possibility represents allowed variants to trade on specific stock exchange. Each trading

possibility is a combination of the parameters. The list of the order possibilites is all possible combinations:

```
MARKET;NORMAL;ABSOLUTE;{KASSA;AUCTION}
LIMIT;NORMAL;ABSOLUTE;{KASSA;AUCTION}
MARKET;NORMAL;RELATIVE;{KASSA;AUCTION}
LIMIT;NORMAL;RELATIVE;{KASSA;AUCTION}
...
```

Field	Type	Description
order_model	OrderModel	Order model
order_supplement	OrderSupplement	Order supplement
trailing_notation	TrailingNotation	Trailing notation
cash_quotations	List of CashQuotation	List of allowed cash_quatations

TradingState

Trading state represents information about tradability. Security can be tradable or not (ex. index)

Field	Description
NO_TRADING_STATE	Trading state is not defined
TRADABLE	Tradable state
NOT_TRADABLE	Not tradable state

TrailingNotation

Trailing notation represent notation type by trailing orders

Field	Description
NO_TRAILING_NOTATION	Trailing notation is not defined
ABSOLUTE	Absolute order notation
RELATIVE	Relative order notation

Transaction

Transaction contains onformation about one transaction

Field	Type	Description
transaction_date	Date	Transaction date
amount	double	Amount value

Field	Type	Description
opponent	string	Transaction opponent
information	string	Information about transaction
value_date	Date	Value date

UnitNote

Unit node type

Field	Description
NO_UNIT_NOTE	Unit note is not defined
PIECE	Piece unit note
PERCENT	Percent unit node. Pieces = Percent/100
PERMIL	Permile unit node. Pieces = Percent/1000
POINTS	Points unit node
MISK	Misk unit node

Validation

Validation type for the order actions

Field	Description
WITHOUT_VALIDATION	Order action will routed directly to the market.
VALIDATE_ONLY	Order will checked by the backend system, but not will be routed to market
VALIDATE_WITH_TOTAL_COSTS	Order will checked by the backend system, but not will be routed to market. Additionally will be requested estimated order action total costs.
VALIDATE_WITH_DETAIL_COSTS	Order will checked by the backend system, but not will be routed to market. Additionally will be requested estimated order action detail costs.
TOTAL_COSTS_ONLY	For the order will be requested estimated order action total costs. No backend system validation is processed.

Questions and Answers

Why is used GRPC engine?

The GRPC is fastest engine for push and pull messaging with building security support. It allowes to reduce time for requests execution (ex. add order) and cares about low level communication.

Is will be supported another languages?

We delivery original protobuf protocol description. It can be used for the code generation for other programming languages like C++, Python, Go, Ruby, Node.js, Android Java, Objective-C,

PHP, Dart and Web. For more information please refer to the grpc.io

Is it planned to extend TAPI?

If we see an interest from the customers to use additional functionality, then TAPI can be extended.

Where I can ask questions / get additional information about TAPI?

Please use next link or refer to the support team.