INTERACTIVE BROKERS

WT Web API for Institutions

User Documentation

Version 3.3 – November 22, 2017

Contents

1. Introduction 4
2. Implementation 4
3. Supported Calls and their Responses 4
4. Third party calls to IB 5
   4.1 Initialization 5
   4.2 Authentication 6
   4.3 Ping 9
   4.4 Market Data Subscribe 9
   4.5 Market Data Unsubscribe 12
   4.6 Chart Data Subscribe 12
   4.7 Chart Data Unsubscribe 14
   4.8 BookTrader Data Subscribe 14
   4.9 BookTrader Data Unsubscribe 15
   4.10 Logout 16
   4.11 Competition 16
5. Unsolicited Messages from IB 17
6. Signed data and its verification from calling program 18
7. ERROR Codes 19
8. Security Checks 21
9. Style Sheet for Each Institution 21
10. Translations 22
11. Authentication Plugin for IBroker Android app 22
   11.1 Plugin Installation 22
   11.2 Plugin Lifecycle 23
   11.3 IBApiSessionDescription 24
   11.4 Signature Verification 24
   11.5 Error Codes 24
   11.6 Plugin Method Detail 24

11.7 Android Demo App 27
12. Authentication Plugin for iBroker iOS app 27
12.1 Plugin Installation 28
12.2 Plugin Lifecycle 28
12.3 Plugin APIs 28
12.4 Error codes 30
12.5 Signature Verification 31
12.6 App Store Submission 31
13. Required from Institutions 32
Appendix 32

WT Web API

1. Introduction

The purpose is to provide third parties with a web-based interface that supports:

- User authentication.
- Market data subscriptions.
- Chart subscription.

We also provide a mobile version of the API for Android and iOS that allows for user authentication. See sections 11 and 12.

To download the API for Mobile versions, please contact IB Sales Engineering (salesengineering@interactivebrokers.com**)**.

2. Implementation

This API uses window.postMessagethat allowsthird party to get data from IB Server.

- Third party website loads a page from IB site inside an iframe
- The API implements a bi-directional postMessage cross-domain communication
- The postMessage message source must be validated on both sides

Requirements:

- Browser must support window.postMessage(IE9+, FF31+,Chrome31+,Safari7+).
- Institution application must be implemented as single page application.
- Websocket support (IE10+, FF31+,Chrome31+,Safari7+).

3. Supported Calls and their Responses

The communication between the iframe and the parent is made using PostMessage messages.

Both incoming and outgoing messages are JSON objects.The data JSON Object will contain ACTION property that identifies which package should handle the message. We use the following ACTION Values: MD and CONTRACT for market data, CHART for

chart data and BOOKTRADER for book trader data. The output package could also contain error codes in an ERRORS property (array type).

Note: With IE9 you will have to use JSON.parse() and JSON.stringify().

Examples are provided with some of the calls in later sections.

4. Third party calls to IB

Prerequisites:

Each Institution needs to:

- Have a white branded ID (string) – that is validated and known to IB.
- Know how long the server should keep the session (minutes, integer between 1 and 30).
- Have an iframe in the DOM (to display the login form).
- Have a selected language for the login form (available: en, cn, de, es, fr, it, jp, nl, ru, tw).
- Have Client version: 1.3.

Requests

Third party makes a call to IB API, and that API invokes call to backend that generates response that is sent back to caller.

The sample implementation of the communication of third party with IB server is by using the module wtapi.js (look in appendix for the location of file). This module wtapi.js can be accessed with the methods of window.WTAPIobject. This module is standalone, does not require any additional library to work.

The communication is asynchronous and implemented with PostMessage messages.

4.1 Initialization

1. Load the IB page with GET parameters from the IB server to the iframe. The parameters are the following:

WB_ID <string> : white branded ID

lang<string> : (optional) display language of the login form

Ex:https://zh.wgw.interactivebrokers.com/wtapibeta/?WB_ID=Asterix&lang=en

Once the iframe loads, it sends a message to parent:

{ method: 'loaded'}

If IB server is down or getting restarted, no response is sent back to parent.

2. Next aninit call is made to iframe:

{ method: 'init', wb_id: <string>, max_timeout: <number>, client_version_id: <number> }

Response:

{ method: 'initialized'}

If the INIT call is made successfully, the frame URL and whitebranded ID are validated and the login page within the iframe is shown.

If INIT is not successful, one of the following messages is sent back to parent:
{ method: 'message', message: 'Initialization failed', ERRORS: <Array> }

Potential Error Codes
WTE099
WTE018
WTE019
WTE023

4.2 Authentication

Authentication is done within the iframe. If login succeeds, the following messages are sent back to the parent. 2 Signature messages are sent along with accounts message.

Response sent to parent:

{ method: 'message', message: 'Logged in' }

{ method: 'setPing', timeout: 60000 }

{ method: 'data', message:

{"BACKEND_SESSION_ID":"581c0b12.00000291","ERRORS":[],"ACCOUNTS":"DU91474","MESSAGE_TYPE":"FORWARD","ID":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","CURRENT_TIME":1478294565714}

{"SIGNATURE_TYPE":"USER_SESSION","MESSAGE_TYPE":"FORWARD","ID":":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","RESULT":true,"SIGNATURE":"OBi7YFp3lBY1d+jMvGg7NwJtsJpNVYqyD67WbQR+wAfRzHvwLVGTEd3EeIaITw3tOtTMzzKuQR4/BJKp1qcLryGPHo5r3sJcgnNEXOqal9g8P/plNpjkgLVaVd0uDgp0aiis+EXW2Z38LKhOEP/3vdGvR4Lvin9jjtmEts724aUnMfVg9ZiNK/dbWd1ADgXgid/f1fkRekDDbDmGphYaQ3jEv0bSkCfg6d8rc/jW+gDAiBGMbPnu9Jxvx1vjbT4vkGPL9PkcpgEJHgFKDXbw6IlNDJ6sHvX41WXZuv/6o49DpgXyITQQhgPwIf7HbKs5uAA4IJSoNEfHVB+sHyv/hnONZIE1vNPjwewtRC84Sob1y7pblYY3eUusfzy756y63qTVBfOoLynUEj12pihs6zvghZd26kMGE1TexO+hs4EkepY1Fnq5jzR0IA5cp0dBXpehGiXbdXp1FwEFFDjWlNUlDpLSgIwcy4SKy6xstPviiasZktlUKbQP1ruTXw4ANcUXLcdggbhABhXWjqsFz8WN9M65hHTCx2CpM3gZn2uLRXz4gg2aRUL4ilXlgMRKi2ILDgWSxVn7R80Q1lhRalt1jmmUBjNY7pQJvGsSOQ3+66i4hiedctWcVugDHsF3IsxVgrp04PmIUjd4XNYKfuhrFsJeQVjCYHAfxduC+bo="}

{"SIGNATURE_TYPE":"USER_ACCOUNTS","MESSAGE_TYPE":"FORWARD","ID":"":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","RESULT":true,"SIGNATURE":"REJSCdTG23xmiPOgACV8uyoeq3vIiTnGo/3EygQ/C3R1FAiXn3e59tAEm6FER15LikWjNw9IXLcsEdiRsKbc+MHZ6iFsU7UrPY/ZORdWL0KkXEqu1swo2ZE4KaT3O0Dujnh+GFGSQcNdYP2I7lSWXS5442Bub06pShXuGOmC1Dd9FdmwKWCrR1kTwm9RsylhLB32OHaNtdaxMYgULPbDH8WVUjAYheRZ389F9yHoSqSRlzSQUdZIch66/747vABMuFzaAYqxxbM0LL0k1fpXpifuFAAqOTMjwtZB6HmSbpgwH45NRD5b7WRrwyFcm8HLpa2P2isc6lCk/AxY42Ay8IQUvluFzSmE3N+4vCbiWf9IiB2jcovZoZb0J/twm2E98J9ryP76IvgdULdMBDciy7J728dSoHr0SOPmBkjGWHrF8sLrKpVFm6YJ3t6UrOhMboOiPIIUe/m/e3WAUn48NOYaOpwiKjjkmBfi5H05Z+UtMMLZErt7w4vRdgbfEb+IgsLX75HBSUwcOhyHP0/Tm/DljFRKNYhTJ2yPE+oKqiC2M8iP8XNChurxFQKqIgT2CaIqCRSAd48C6hBEZqacX4CUGbeKDaptnRwkaIopzlqIQ+NWVLLlPulV3aIOS81W8+oq73dNa6/aXnR8fEoIKk9acvM7Sjy1NbSPumnYw+Q="}

If user authentication fails, user cannot proceed beyond login page.

4.3 Ping

A ping message needs to be sent to the iframe every 1 minute (defined as timeout below). The ping should start after a successful login and stops when thelogout() method is called.

If ping stops and the max_timeoutdefined in Section 4.1 is reached, the server will clean out the session and no market data or chart data will be sent from the server.

After successful login, the iframe sends a ping message as below:

{ method: 'setPing', timeout: 60000 }

After that the parent needs to send a ping message to iframe, at the timeout interval defined above.

The parent frame has to send the following message periodically:

{ method: 'ping' }

4.4 Market Data Subscribe

To subscribe to market data the application should call the 'mdSubscribe' method with a conid-exchange-pairs parameter. This parameter can contain one or more conid-exchange pairs separated by a semicolon character. Ex: 8314:SMART;43645865:SMART

Input below:

{ method: 'mdSubscribe', md: <string> }

After a successful subscribe the data callback function willbe called. For market data two types of ACTIONS are relevant: CONTRACT with static data describing the contract properties and MD with prices and other market data attributes. The data callback function will call the correct package based on the ACTION property in the message. See parseMarketDataPackage() function in client.js for detailed info.

{ method: 'data',

message
:{"RESULT":true,"MESSAGE_TYPE":"FORWARD","symbol":"IBM","sec_type":"STK",

"ACTION":"CONTRACT","exchange":"SMART","currency":"USD","conid":8314}}

{ method: 'data', message:

{"MESSAGE_TYPE":"FORWARD","ACTION":"MD","MD":[{"change_price":"-1.72","bid_size":"2","last_price":"145.65","is_delayed":false,"open_price":"147.37","change_percent":"-1.17%","conid":8314,"bid_price":"145.43","ask_price":"146.47","last_trading_day":"20150911","volume":"2.91M","close_price":"147.37","dividend_yield":"3.6%","high":"147.37","low":"145.41","exchange":"SMART","ask_size":"2"}]}}

If you subscribe to an incorrect conid-exchange pair, you get an error like below:

{method: "message", message: "Market Data subscription failed", ERRORS:["WTE020"]}

Potential Error Codes
WTE008
WTE020
WTE028

Data fields returned after subscriptionin MD Action is:

MD Object key value pair of:

con_id

exchange

change_percent

change_price

bid_size

bid_price

ask_size

ask_price

volume (in K, M, B)

high

low

market_value

average_price

symbol

company_name

contract_description_1: contains the symbol in most cases

last_price

yield_last

mark_price

is_delayed(true/false) : If user doesn’t have market data permission for this conid, exchange pair; but we have delayed market data available then this flag is true.

open

close

open_price

close_price

dividend_yield

4.5 Market Data Unsubscribe

To unsubscribe market data the app should call 'mdUnsubscribe' method with a conid-exchange-pairs parameter. This parameter can contain one or more conid-exchange pairs separated by a semicolon character. Ex: 8314:SMART;43645865:SMART

Input below:

{ method: 'mdSubscribe', md: <string> }

After un-subscribe, market data stops ticking for the un-subscribed conid, exchange pair.Only unsubscribe for conid-exchange pair, to which you have subscribed earlier.

If you unsubscribe to incorrect conid-exchange pair, you get an error like below:

{method: "message", message: "Cancelling Market Data subscription failed", ERRORS:["WTE020"]}

Potential Error Codes
WTE008
WTE020

4.6 Chart Data Subscribe

To subscribe to chart data the application should call 'chartSubscribe' method with three parameters. The first parameter contains one (and only one) conid-exchange-pair. The second parameter is the length of the chart. This value can be selected from a predefined list (1d, 1w, 2w, 1m, 6m, 1y, 5y). The last one is a boolean for “Outside Regular Trading Hours”.

Input below:

{ method: 'chartSubscribe', md: <string> , time: <string> , orth: <boolean> }

After a successful Chart subscribe the data callback function called the chart packages is invoked. The CHART type packages contain the data in following format.

The chart data is sent to parent frame every 1 minute.

{ method: 'data', message: {"RESULT":true,"MESSAGE_TYPE":"BOTH","SESSION":"d53d96e0-0095-4ef0-bc33-

2b7117d6ca03","data_points":[{"open":147.37,"time":1442237430000,"volume":607,"high":147.37,"low":147,"close":147.2},{"open":147.21,"time":1442237490000,"volume":19,"high":147.31,"low":147.18,"close":147.31},{"open":147.14,"time":1442237550000,"volume":81,"high":147.31,"low":147.06,"close":147.14},{"open":146.35,"time":1442238150000,"volume":53,"high":146.38,"low":146.24,"close":146.38},{"open":146.29,"time":1442238210000,"volume":23,"high":146.29,"low":146.14,"close":146.14},{"open":145.46,"time":1442257830000,"volume":18,"high":145.48,"low":145.44,"close":145.46},{"open":145.47,"time":1442257890000,"volume":71,"high":145.54,"low":145.47,"close":145.53}],"start_time":"2015091413:30:30","ACTION":"CHART"}}

If your chart subscribe fails, you get an error like below:

{ method: 'data', message:
{"RESULT":false,"MESSAGE_TYPE":"BOTH","SESSION":"6008e42c-6e22-4c08-97a9-
a092c6201517","ERRORS":["WTE021"],"ACTION":"CHART_UNSUBSCRIBE"}}

Potential Error Codes
WTE008
WTE021
WTE022
WTE028

4.7 Chart Data Unsubscribe

To unsubscribe chart data the app should call 'chartUnsubscribe' method with a one (and only one) conid-exchange-pair. Once unsubscribe is called, the chart details stop being sent to parent every 1 minute.

Input:

{ method: 'chartUnsubscribe', md: <string> }

If chart unsubscribe fails, you will a get message like the below:

{ method: 'data', message:
{"RESULT":false,"MESSAGE_TYPE":"BOTH","SESSION":"6008e42c-6e22-4c08-97a9-
a092c6201517","ERRORS":["WTE021"],"ACTION":"CHART_UNSUBSCRIBE"}}

Potential Error Codes
WTE008
WTE021

4.8 BookTrader Data Subscribe

To subscribe to BookTrader the application should call 'booktraderSubscribe' method with one parameter. The first parameter contains one (and only one) conid-exchange-pair.

Input below:

{ method: 'booktraderSubscribe', md: <string> }

After a successful BookTrader subscribe, the data callback function called the book traderpackages is invoked. The BOOKTRADERtype packages contain the data in following format.

The chart data is sent to parent frame every 1 minute.

{"data":[{"price":"0.10","row_number":0},{"price":"0.09","row_number":1},{"price":"0.08",
"row_number":2},{"price":"0.07","row_number":3},{"price":"0.06","row_number":4},{"price":

"0.05","row_number":5},{"price":"0.04","row_number":6},{"price":"0.03","row_number":7}, {"price": "0.02","row_number":8}, {"price":"0.01","row_number":9}, {"price":"0.00","row_number":10}], "ACTION":"BOOKTRADER","MESSAGE_TYPE":"BOTH","SESSION":"6008e42c-6e22-4c08-97a9-a092c6201517","RESULT":true}

If your BookTrader subscribe fails, you get an error like below:

{"ERRORS":["WTE027"],"ACTION":"BOOKTRADER","MESSAGE_TYPE":"BOTH","SESSION":"6008e42c-6e22-4c08-97a9-a092c6201517","REASON":"no sec defs returned forSecDefreqId=book for [8314/@SMART]ReqByConid10749 for 8314/@SMART","RESULT":false}

Potential Error Codes
WTE008
WTE026
WTE027
WTE028

4.9 BookTrader Data Unsubscribe

To unsubscribe BookTrader data the app should call 'booktraderUnsubscribe' method with a one (and only one) conid-exchange-pair. Once unsubscribe is called, the book trader details stop being sent to parent.

Input:

{ method: 'booktraderUnsubscribe', md: <string> }

If BookTrader unsubscribe fails, you will get a message like the below:

{ method: 'data', message: {"RESULT":false,"MESSAGE_TYPE":"BOTH","SESSION":"6008e42c-6e22-4c08-97a9-a092c6201517","ERRORS":["WTE021"],"ACTION":""BT_UNSUBSCRIBE"}}

Potential Error Codes
WTE008
WTE027

4.10 Logout

To end interaction with the iframe and to clean up resources on the server, call the'logout' method.

Input below:

{ method: 'logout' }

On successful logout, the response is:

{ method: 'message', message: 'Logged out' }

After a successful logout the parent should unload the iframe (null the iframe source).

4.11 Competition

If the user is logged into any other IB trading application like Webtrader, TWS or another WT Web APIi version with the same username and tries to login into WT Web API, there will be a competition message. This competition callback function is called with a message 'competition'. The callback function should display a popup message (with the native confirm or a custom one like jQuery Dialog) asking the user to continue the session or stop it.

Competition message:
{ method: 'competition', message: <string> }

If the user wants to continue the current session, the app should call:
{ method: 'keepSession' }
To kill the current session, call the method:
{ method: 'logout' }

5. Unsolicited Messages from IB

All market data messages are sent to the parent once they arrive. On subscription of a market data message, the server pushes every tick to the client, and the parent does not need to poll for it. If websocket is enabled for the browser and it is regular trading hours, market data ticks ever 250 ms. If browser does not support websocket or if websocket is disconnected for any reason, market data is sent to the client every three seconds.

{ method: 'data',
message:{"MESSAGE_TYPE":"FORWARD","ACTION":"MD","MD":[{"symbol":"IB
KR","conidex":"43645865","company_name":"INTERACTIVE BROKERS GRO-CL
A","expiry_type":"0","sec_type":"STK","listing_exchange":"NASDAQ.NMS","contract
_description_1":"IBKR","exchange":"SMART","conid":43645865}]}}
{ method: 'data',
message:{"MESSAGE_TYPE":"FORWARD","ACTION":"MD","MD":[{"change_price
":"-
1.90","bid_size":"3","last_price":"145.47","is_delayed":false,"open_price":"147.37","cha
nge_percent":"-
1.29%","market_value":"1,018","conid":8314,"mark_price":"145.47","bid_price":"145.4
7","ask_price":"145.48","last_trading_day":"20150911","volume":"1.61M","close_price"
:"147.37","dividend_yield":"3.6%","high":"147.37","low":"145.41","exchange":"SMAR
T","ask_size":"8"}]}}

Additionally, the Chart message is sent to parent frame every 1 minute by IB Server.

The following messages are sent from the server if there are any application problems. The client should either try to reconnect, or show login window again.

WT Error	What to do
WTE002	Show the login window
WTE003	Show the login window
WTE006	Let user decide to end current session or disconnect other session; call force initialization accordingly
WTE008	Show the login window

6. Signed data and its verification from calling program

Once a user has logged in, there will be three messages sent to the client. The data will be formatted like the below:

DATA 1:

{"BACKEND_SESSION_ID":"581c0b12.00000280","ERRORS":[],"ACCOUNTS":"DU91474","MESSAGE_TYPE":"FORWARD","ID":"":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","CURRENT_TIME":1478290612110,"RESULT":true}

DATA 2:

{"SIGNATURE_TYPE":"USER_SESSION","MESSAGE_TYPE":"FORWARD","ID":"":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","RESULT":true,"SIGNATURE":"OBi7YFp3lBY1d+jMvGg7NwJtsJpNVYqyD67WbQR+wAfRzHvwLVGTEd3EeIaITw3tOtTMzzKuQR4/BJKp1qcLryGPHo5r3sJcgnNEXOqal9g8P/plNpjkgLVaVd0uDgp0aiis+EXW2Z38LKhOEP/3vdGvR4Lvin9jjtmEts724aUnMfVg9ZiNK/dbWd1ADgXgid/f1fkRekDDbDmGphYaQ3jEv0bSkCfg6d8rc/jW+gDAiBGMbPnu9Jxvx1vjbT4vkGPL9PkcpgEJHgFKDXbw6IlNDJ6sHvX41WXZuv/6o49DpgXyITQQhgPwIf7HbKs5uAA4IJSoNEfHVB+sHyv/hnONZIE1vNPjwewtRC84Sob1y7pblYY3eUusfzy756y63qTVBfOoLynUEj12pihs6zvghZd26kMGE1TexO+hs4EkepY1Fnq5jzR0IA5cp0dBXpehGiXbdXp1FwEFFDjWlNUlDpLSgIwcy4SKy6xstPviiasZktlUKbQP1ruTXw4ANcUXLcdggbhABhXWjqsFz8WN9M65hHTCx2CpM3gZn2uLRXz4gg2aRUL4ilXlgMRKi2ILDgWSxVn7R80Q1lhRalt1jmmUBjNY7pQJvGsSOQ3+66i4hiedctWcVugDHsF3IsxVgrp04PmIUjd4XNYKfuhrFsJeQVjCYHAfxduC+bo="}

DATA 3:

{"SIGNATURE_TYPE":"USER_ACCOUNTS","MESSAGE_TYPE":"FORWARD","ID":"":"d53d96e0-0095-4ef0-bc33-2b7117d6ca03","USER":"qapap908","RESULT":true,"SIGNATURE":"REJSCdTG23xmiPOgACV8uyoeq3vIiTnGo/3EygQ/C3R1FAiXn3e59tAEm6FER15LikWjNw9IXLcsEdiRsKbc+MHZ6iFsU7UrPY/ZORdWL0KkXEqu1swo2ZE4KaT3O0Dujnh+GFGSQcNdYP2I7lSWXS5442Bub06pShXuGOmC1Dd9FdmwKWCrR1kTwm9RsylhLB32OHaNtdaxMYgULPbDH8WVUjAYheRZ389F9yHoSqSRlzSQUdZIch66/747vABMuFzaAYqxxbM0LL0k1fpXpifuFAAqOTMjwtZB6HmSbpgwH45NRD5b7WRrwyFcm8HLpa2P2isc6lCk/AxY42Ay8IQUvluFzSmE3N+4vCbiWf9IiB2jcovZoZb0J/twm2E98J9ryP76IvgdULdMBDciy7J728dSoHr0SOPmBkjGWHrF8sLrKpVFm6YJ3t6UrOhMboOiPIIUe/m/e3WAUn48NOYaOpwiKjjkmBfi5H05Z+UtMMLZErt7w4vRdgbfEb+IgsLX75HBSUwcOhyH

P0/Tm/DljFRKNYhTJ2yPE+oKqiC2M8iP8XNChurxFQKqIgT2CaIqCRSAd48C6hBEZqacX4CUGbeKDaptnRwkaIopzlqIQ+NWVLLlPulV3aIOS81W8+oq73dNa6/aXnR8fEoIKk9acvM7Sjy1NbSPumnYw+Q="}

DATA2 is the **username|backend_session_id|time**which has been signed

DATA3 is the **username|accounts|time**which has been signed

When the calling application gets the data, they will take the Signature and verify it against the unsigned data using the downloaded certificate.

The public certificate to be used for verifying the data can be downloaded from: https://zh.wgw.clientam.com/wtapiqa/downloadCertificate

The steps to verify (to be done by the calling application):

1. Download and save the public certificate from IB (at a certain file location on your end). This doesn't need to be done every day. It can be once, and then every time the key changes (IB will notify application team).
2. Ensure that the current_time is within a few minutes of the application's server time. This ensures that no one is playing back old messages.
3. Run Verify (sample java code attached). If Verify is true and 2 is true, then you can proceed knowing that the user has logged into the IB system in the last few minutes.
4. Send the backend_session_id on orders for additional security.

7. ERROR Codes

All error codes with descriptions. The last column indicates if the message can be seen in the local login page, is sent to parent, or both.

WT Error	Description	Where
WTE001	Not Supported Message	Both
WTE002	Generic Error	Both
WTE003	Disconnected	Both
WTE004	Unable to Connect	Both
WTE005	Invalid username or password	Local
WTE006	Competition: user connected on another machine	Both

WT Error	Description	Where
WTE007	Unable to connect to api	Both
WTE008	Not connected to server	Both
WTE009	Restricted IP	Local
WTE010	User not specified	Local
WTE011	User locked out	Local
WTE012	Password expired	Local
WTE013	Session Token Authentication Failed	Local
WTE014	Session Token Authentication Passed	Local
WTE015	AUTHENTICATED	Local
WTE016	NEED_AUTHENTICATION	Local
WTE017	Guaranteed Dollar user no supported	Local
WTE018	Invalid Whitebranded user	Parent
WTE019	Invalid Parent domain	Parent
WTE020	Invalid subscription	Parent
WTE021	Invalid chart request	Parent
WTE022	Chart Result failed	Parent
WTE023	Invalid Timeout	Parent
WTE024	Invalid contract	Parent
WTE025	Invalid Login Message	Both
WTE026	Invalid Book Trader request	Parent

WT Error	Description	Where
WTE027	Invalid Book Trader results	Both
WTE028	Invalid access	Parent
WTE099	Min Version	Parent

8. Security Checks

Only valid white branded users and valid domains are allowed to use this application. Applications will need to register their test and prod domains with IB (this will be discussed during setup).

Also check with IB for the correct iframe domain to be used for PostMessage validations on the parent side. This could be different for test and production environments.

Each institution will need to specify which set of functionality they will need, which will need to be configured on IB's side. Some institutions might need to use this for login only and others for both login and market data access.

9. Style Sheet for Each Institution

A Custom CSS style sheet to alter the look of the login page can be used for each institution. These styles sheets will be hosted by IB, separately from the application. WT Web API will contain the default configuration but will allow different style sheets per white branding ID to be loaded after the default style sheet. This style sheet for the white branded user will be served from a different location so that style sheet changes will not require a release.

The custom style sheet can overwrite the colors, background colors, font family, font size, etc. Recommended selectors to use are: body, input.text, button.submit, h1, .title. Institutionswill have to test their modified style sheet on every major supported browser before sending to IB. IB will not be held responsible for the broken layout of a login page after incorporating the custom style sheet.

Style sheets to be incorporated for each white branded user (if any) will be confirmed during setup.

10. Translations

Institutions can be provided with the Error codes for translations (as needed). The process followed would be the same as for other IB applications and would be through CLAMS. Users could choose from a list of languages that we offer for translation and translated text, error codes would be shown on the login page.

Please contact IB Sales Engineering (salesengineering@interactivebrokers.com**)** for additional translation needs.

11. Authentication Plugin for IBroker Android app

The plugin enables IBroker Android app to provide user authentication using credentials at IB and second factor. Authentication will be done in following steps:

- User requests to login from IBroker app
- IBroker app makes a plugin call to start a session
- User is directed to web browser to enter IB credentials
- After successful authentication, user is directed back to IBroker app
- Plugin provides IBroker app the IBApiSessionDescription object that contains username, list of accounts, session id, and signatures information
- IBroker backend verifies that signatures are IB authentic and approves the login

Requirements:

- Android 4.0+ (API level 14+)
- Internet permission

11.1 Plugin Installation

IBroker app needs to include the plugin .jar file in build path. Additionally, iBroker app is required to have a dedicated activity to host the plugin. Let us assume this activity is called LoginActivity in the rest of this document.

AndroidManifest.xml is required to contain following settings:

- configure LoginActivity intent-filter to accept URL from ib.wtapi.android.client://ibkr.com
- install IBApiClientService as a service

For example (see demo app for complete source code):

```
<activity android:name=".LoginActivity" android:launchMode="singleTask">
<intent-filter>
<data android:scheme="ib.wtapi.android.client" android:host="ibkr.com" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<action android:name="android.intent.action.VIEW" />
</intent-filter>
</activity>
<service android:name="com.ib.wtapi.android.pub.IBApiClientService" />
```

If IBroker app is using gradle build tool, and target sdk Android 6.0 (API Level 23) or higher, app `build.gradle` (see demo app for example) needs to include the following line:

```
useLibrary 'org.apache.http.legacy'
```

11.2 Plugin Lifecycle

The plugin will start a background service that communicates with IB backend. It is important that plugin follows LoginActivity's (the dedicated activity that hosts the plugin) lifecycle closely, so that the background service can attach-to/detach-from LoginActivity properly.

- LoginActivity'sonCreate() method needs to initialize and obtain an IBApiClient object by calling IBApiClientFactory.createIBApiClient().
- LoginActivity'sonStart(), onStop(), onSaveInstanceState() methods need to make similar plugin calls accordingly.

IBroker app can call IBApiClient.startSession() to start the authentication process. In order to ensure plugin background service is initialized and attached properly, the following prerequisites need to be met:

- IBroker app is running on the front (not in background)
- LoginActivity is fully resumed
- It is triggered by user action, e.g. user click Login button to request login

IBroker app can call IBApiClient.logoutSession() to log out current session.

11.3 IBApiSessionDescription

IBApiSessionDescription contains username, list of accounts, session id, and signatures information. After successful authentication, IBroker app can obtain IBApiSessionDescription object in one of the following ways:

- callback method IBApiClient.LoginCallback.onSuccess(IBApiSessionDescription)
- IBApiClient.getSessionDescription()

11.4 Signature Verification

For security purpose, IBroker is required to verify signatures before granting user account access. The steps are:

- IBroker app sends IBApiSessionDescription object data to iBroker backend
- IBroker backend verifies logged-in-time is LESS than 2 minute old
- IBroker backend verifies session signature is from IB
- IBroker backend verifies accounts signature is from IB

The demo app includes IB public key and example code (see SimulatedServer.java) that is doing the above signature verification.

11.5 Error Codes

If there is an error during authentication, IBroker app can get the error codes in one of the following ways:

- callback method IBApiClient.LoginCallback.onFailure(String)
- IBApiClient.getFailureReason()

11.6 Plugin Method Detail

IBApiClientFactory

- **createIBApiClient**

```
public static IBApiClient createIBApiClient(String whiteBrandingId,
                            String language,
                            Activity activity,
                            Bundle savedInstanceState,
                            String defaultIbBackendURL,
                            IBApiClient.LoginCallback loginCallBack,
                            [TradingMode tradingMode]
                                                )
```

Create an IB Api Client

Parameters:
whiteBrandingId - iBroker id
language - pick one from IBApiClient.LANGUAGE_* constants
activity - caller (host) activity
savedInstanceState - from activity's onCreate(Bundle savedInstanceState)
defaultIbBackendURL - the IB backend URL, use IBApiClientFactory.IB_BACKEND_URL_BETA for beta testing, use IBApiClientFactory.IB_BACKEND_URL_PROD for production; Or use special URL provided by IB
loginCallBack - callback to handle login results
[tradingMode] - optional parameter, use TradingMode.LIVE for production

Returns:
an IB Api Client object

IBApiClient

- **startSession**

```
void startSession()
```

Initiate a login session

- **isValidSession**

```
boolean isValidSession(String sessionID)
```

Check if session is valid

Parameters:
`sessionID` - session ID

Returns:
true if session ID is valid

- **logoutSession**

```
void logoutSession(String sessionID)
```

Log out session

Parameters:
`sessionID` - session ID

- **getSessionDescription**

```
IBApiSessionDescription getSessionDescription()
```

Get session description

Returns:
IBApiSessionDescription object if last login is successful; otherwise return null

- **getFailureReason**

```
String getFailureReason()
```

Get failure reason

Returns:
failure reason if last login fails; otherwise return null

- **onStart**

```
void onStart()
```

iBroker activity's onStart() method should call this

- **onStop**

```
void onStop()
```

iBroker activity's onStop() method should call this

- **onSaveInstanceState**

```
void onSaveInstanceState(Bundle outState)
```

iBroker activity's onSaveInstanceState(Bundle) method should call this

- **getMessageLog**

```
String getMessageLog()
```

Get message log; this is used for debug only

IBApiClient.LoginCallback

- **onSuccess**

```
void onSuccess(IBApiSessionDescription sessionDesc)
```

called after successful login, and provide IBApiSessionDescription object

- **onFailure**

```
void onFailure(String reason)
```

called when login failed

11.7 Android Demo App

The demo app project can be imported into Android Studio. The requirements are:

- Android Studio 1.5+
- Minimum Android 4.1 (API Level 16), Target Android 6.0 (API Level 23)

The demo app has the following classes:

- LoginActivity : main login activity that host the plugin
- SimulatedServer : demonstrates how to verify IB signatures on iBroker backend
- LauncherActivity : simply redirect to LoginActivity

IB public key is located in path app\src\main\res\raw\ib_selfsigned_cert.dat

AndroidManifest.xml demonstrates necessary elements to install plugin.

The demo app is connecting to `IBApiClientFactory.IB_BACKEND_URL_BETA` that is for beta testing only. In production, please use `IBApiClientFactory.IB_BACKEND_URL_PROD`; Or use special URL provided by IB.

12. Authentication Plugin for iBroker iOS app

The plugin enables iBroker iOS app to provide user authentication using credentials at IB and second-factor. Authentication will be done in the following steps:

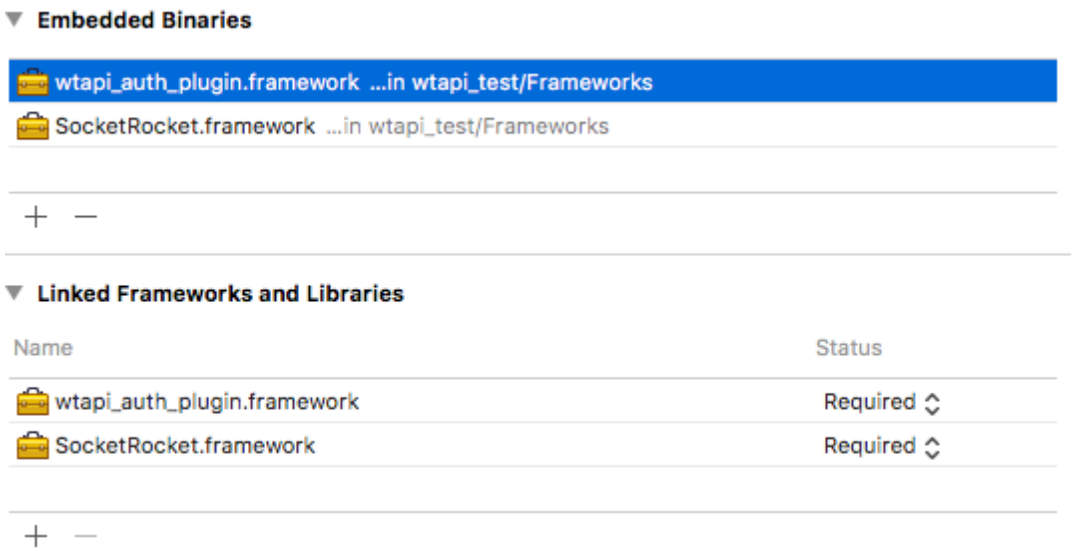
- User requests to login from iBroker app
- IBroker app makes a plugin call to start login
- User is directed to a modally presented web browser to enter IB credentials
- After successful authentication, web browser is dismissed
- Plugin provides iBroker app the IBApiSessionDescription object that contains username, list of accounts, session id, and signatures information
- IBroker backend verifies that signatures are IB authentic and approve the login

Requirements:

- iOS 8.0 or higher

12.1 Plugin Installation

The plugin is delivered as a dynamic library: wtapi_auth_plugin.framework. The plugin depends on the SocketRocket framework, which is also included. On your application targets' “General” settings tab, in the “Embedded Binaries” and the “Linked Frameworks and Libraries” section, add both frameworks.



12.2 Plugin Lifecycle

Create an IBApiClient instance. Make sure the returned IBApiClient instance is kept alive during the whole process by assigning it to a strong property. Parameter viewController is a UIViewController on which the login web browser ViewController can be modally presented. Make sure this view controller is not already presenting another modal view controller.

12.3 Plugin APIs

IBApiClientFactory

- **createIBApiClientWithWhiteBrandingId**

+ (IBApiClient*) createIBApiClientWithWhiteBrandingId:(NSString*)whiteBrandingId language:(IBApiClientLanguage)language

backendURL:(NSURL*)backendURL

tradingMode:(TradingMode)tradingMode

viewController:(UIViewController*)viewController;

Parameters:

whiteBrandingId - iBroker id

language - pick one from the IBApiClientLanguage constants

backendURL - the IB backend URL, use [IBApiClientFactorydefaultIBBackendURL], or usespecial URL provided by IB

tradingMode - can be paper or live. TradingModeShowSelector displays a selector UI.

viewController - a ViewController on which the login web browser ViewController can be modally presented

Returns:

an IB Api Client object

defaultIBBackendURL

+ (NSString*) defaultIBBackendURL

Returns:

Default URL to connect to IB.

IBApiClient

- **loginWithCallback**

(void) loginWithCallback:(LoginCallback)callback;
Initiates login.
Parameters:
callback - a block with log in results.

- **LoginCallback**

  typedef void (^LoginCallback) (BOOL success, NSError *error, IBSessionDescription *sessionDescription);

  Called when login is finished.

  Parameters:
  success - Shows if login was successful
  error - Optional error during login
  sessionDescription - Object containing login data in case of a successful login

IBApiClientDelegate

- **messageLogged**

  (void) messageLogged:(NSString*)message;

  Returns log messages as they are received.

  Parameters:

  message: received message

IBSessionDescription

Contains username, list of accounts, session id, and signatures information:
@property (nonatomic,readonly) NSString *sessionId;
@property (nonatomic,readonly) NSString *accounts;
@property (nonatomic,readonly) NSString *username;
@property (nonatomic,readonly) NSString *loggedInTime;
@property (nonatomic,readonly) NSString *sessionSignature;
@property (nonatomic,readonly) NSString *accountsSignature;

12.4 Error codes

If there is an error during authentication, iBroker app receives error codes.

Error domain: IBApiClientErrorDomain

IBAPICLIENT_ERROR_CODE_MESSAGING code: 1 WT Web API error codes are forwarded in this error. WTE error codes are in the error's localized description. See Section 7 for full list of error codes and descriptions.

IBAPICLIENT_ERROR_CODE_SOCKET_CLOSED code: 2 Server unexpectedly closed socket connection.

IBAPICLIENT_ERROR_CODE_USER_CANCELLED code: 3 User aborted login.

IBAPICLIENT_ERROR_CODE_FAILED_OPEN_URL code: 4 Web browser failed to open login url.

12.5 Signature Verification

For security purposes, iBroker is required to verify signatures before granting user account access.

The steps are:

- IBroker app sends IBApiSessionDescription object data to iBroker backend
- IBroker backend verifies logged-in-time is LESS than 2 minute old
- IBroker backend verifies session signature is from IB
- IBroker backend verifies accounts signature is from IB
- The iOS demo app includes IB public key and example code that is doing the above signature verification.

12.6 App Store Submission

Due to a bug in the App Store submission process (www.openradar.me/23318511) projects containing fat, multi-platform frameworks can't be submitted to the store. To solve this, the plugin is delivered in two different versions:

1. ARM only framework: contains armv7 and arm64 binaries. Can be used to upload to the App Storeand to test on real devices.
2. Multiplatform framework: contains i386, x86_64, armv7 and arm64 binaries. Can be used to teston the simulator and real devices, but can't be uploaded to the App Store.

The Demo app initially contains the fat, multiplatform framework files. The framework files can be found in the Frameworks directory (socketrocket_arm, socketrocket_fat, wtapi_auth_plugin_arm,wtapi_auth_plugin_fat). Make sure to replace the multiplatform framework files with the ARM only framework files before submitting to the App Store.

cd<PARENT DIRECTORY>/Client\ project

rm -rf ./wtapi_test/wtapi_test/Frameworks/SocketRocket.framework/

rm -rf ./wtapi_test/wtapi_test/Frameworks/wtapi_auth_plugin.framework/

cp -R ./Frameworks/socketrocket_arm/SocketRocket.framework
./wtapi_test/wtapi_test/Frameworks/

cp -R ./Frameworks/wtapi_auth_plugin_arm/wtapi_auth_plugin.framework
./wtapi_test/wtapi_test/Frameworks/

13. Required from Institutions

Institutions must provide the following information to IB before they start:

- Is functionality provided using Mobile App and/or Browser based app?
- Is application to be used for Authentication, market data, or both?
- Provide a white branded ID (string)
- Tell us how long server should keep the session alive (in minutes, integer between 1 and 30).
- Provide the Test and Prod domains from which you will call wtapi
- Number of users who will use the application in production, by region
- Number of users during peak time
- Additional translation needs, if any

Please contact IB Sales Engineering (salesengineering@interactivebrokers.com) with the above information, and confirm that white-branded users are set up for testing before proceeding with development/testing.

Appendix

The demo url is:

https://zh.wgw.clientam.com/wtapibeta/frame/index.jsp

The sample wtapi.js can be found at:

https://zh.wgw.clientam.com/wtapibeta/frame/wtapi.js

The sample client.js can be found at:

https://zh.wgw.clientam.com/wtapibeta/frame/clientam.js

Please copy these 2 files (wtapi.js and client.js) and customize as needed in your application.

Sample Java program to verify signature:

VerifySignature.java