Peter Gustafson
Interactive Brokers: Connectivity
April 3, 2024

Connectivity

A socket connection between the API client application and TWS is established with the [IBApi.EClientSocket.eConnect](https://interactivebrokers.github.io/tws-api/classIBApi_1_1EClientSocket.html#a315a7f7a34afc504d84c4f0ca462d924) function. TWS acts as a server to receive requests from the API application (the client) and responds by taking appropriate actions.

The first step is for the API client to initiate a connection to TWS on a socket port where TWS is already listening. It is possible to have multiple TWS instances running on the same computer if each is configured with a different API socket port number.

Also, each TWS session can receive up to **32 different client applications** simultaneously. The **client ID** field specified in the API connection is used to distinguish different API clients.

Establishing an API connection

Once our two main objects have been created, EWrapper and ESocketClient, the client application can connect via the [IBApi.EClientSocket](https://interactivebrokers.github.io/tws-api/classIBApi_1_1EClientSocket.html) object:

- clientSocket.eConnect("127.0.0.1", 7497, 0);

eConnect starts by requesting from the operating system that a TCP socket be opened to the specified IP address and socket port. If the socket cannot be opened, the operating system (not TWS) returns an error which is received by the API client as error code 502 to [IBApi.EWrapper.error](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a7dfc221702ca65195609213c984729b8 "Handles errors generated within the API itself. If an exception is thrown within the API code it will...")

(**Note:** since this error is not generated by TWS it is not captured in TWS log files).

Most commonly error 502 will indicate that TWS is not running with the API enabled, or it is listening for connections on a different socket port. If connecting across a network, the error can also occur if there is a firewall or antivirus program blocking connections, or if the router's IP address is not listed in the "Trusted IPs" in TWS.

 - After the socket has been opened, there must be an initial handshake in which information is exchanged about the highest version supported by TWS and the API.
 - This is important because API messages can have different lengths and fields in different versions and it is necessary to have a version number to interpret received messages correctly.

-   For this reason it is important that the main EReader object is not created until after a connection has been established.
- The initial connection results in a negotiated common version between TWS and the API client which will be needed by the EReader thread in interpreting subsequent messages.

After the highest version number which can be used for communication is established, TWS will return certain pieces of data that correspond specifically to the logged-in TWS user's session.

This includes (1) the account number(s) accessible in this TWS session, (2) the next valid order identifier (ID), and (3) the time of connection. In the most common mode of operation the EClient.AsyncEConnect field is set to false and the initial handshake is taken to completion immediately after the socket connection is established. TWS will then immediately provides the API client with this information.

Important: The **IBApi.EWrapper.nextValidID** callback is commonly used to indicate that the connection is completed and other messages can be sent from the API client to TWS. There is the possibility that function calls made prior to this time could be dropped by TWS.

There is an alternative, deprecated mode of connection used in special cases in which the variable A`syncEconnect` is set to true, and the call to startAPI is only called from the `connectAck()` function. All IB samples use the mode `AsyncEconnect = False`.

# The EReader Thread

API programs always have at least two threads of execution. One thread is used for sending messages to TWS, and another thread is used for reading returned messages.

The second thread uses the API EReader class to read from the socket and add messages to a queue. Every time a new message is added to the message queue, a notification flag is triggered to let other threads now that there is a message waiting to be processed.

In the two-thread design of an API program, the message queue is also processed by the first thread. In a three-thread design, an additional thread is created to perform this task.

The thread responsible for the message queue will decode messages and invoke the appropriate functions in EWrapper. The two-threaded design is used in the IB Python sample Program.py and the C++ sample TestCppClient, while the 'Testbed' samples in the other languages use a three-threaded design. Commonly in a Python asynchronous network application, the [asyncio module](https://docs.python.org/3/library/asyncio.ht) will be used to create a more sequential looking code design.

The class which has functionality for reading and parsing raw messages from TWS is the [IBApi.EReader](https://interactivebrokers.github.io/tws-api/classIBApi_1_1EReader.html) class.

-   //Create a reader to consume messages from the TWS. The EReader will consume the incoming messages and put them in a queue

```
    > var reader = new EReader(clientSocket, readerSignal);
    >
    > reader.Start();
    >
    > //Once the messages are in the queue, an additional thread can be
    > created to fetch them
    >
    > new Thread(() => { while (clientSocket.IsConnected()) {
    > readerSignal.waitForSignal(); reader.processMsgs(); } }) {
    > IsBackground = true }.Start();
```

Now it is time to revisit the role of [IBApi.EReaderSignal](https://interactivebrokers.github.io/tws-api/

interfaceIBApi_1_1EReaderSignal.html) initially introduced in [The EClientSocket Class](https://interactivebrokers.github.io/tws-api/client_wrapper.html#client_socket). As mentioned in the previous paragraph, after the EReader thread places a message in the queue, a notification is issued to make known that a message is ready for processing. In the (C++, C#/.NET, Java) APIs, this is done via the [IBApi.EReaderSignal](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EReaderSignal.html) object we initiated within the [IBApi.EWrapper](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html)'s implementer. In the Python API, it is handled automatically by the [Queue class](https://docs.python.org/3/library/queue.html).

The client application is now ready to work with the Trader Workstation! At the completion of the connection, the API program will start receiving events such as [IBApi.EWrapper.nextValidId](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a09c07727efd297e438690ab42838d332) and [IBApi.EWrapper.managedAccounts](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#abd7e561f313bcc4c860074906199a46c). In TWS (_not IB Gateway_) if there is an active network connection, there will also immediately be callbacks to [IBApi::EWrapper::error](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a7dfc221702ca65195609213c984729b8 "Handles errors generated within the API itself. If an exception is thrown within the API code it will...") with errorId as -1 and errorCode=_2104_,_2106_, errorMsg = "Market Data Server is ok" to indicate there is an active connection to the IB market data server. Callbacks to [IBApi::EWrapper::error](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a7dfc221702ca65195609213c984729b8 "Handles errors generated within the API

itself. If an exception is thrown within the API code
it will...") with errorId as -1 do not represent true
'errors' but only notifications that a connection has
been made successfully to the IB market data farms.

IB Gateway by contrast will not make connections to
market data farms until a request is made by the IB
client. Until this time the connection indicator in the
IB Gateway GUI will show a yellow color of 'inactive'
rather than an 'active' green indication.

When initially making requests from an API application
it is important that the verifies that a response is
received rather than proceeding assuming that the
network connection is ok and the subscription request
(portfolio updates, account information, etc) was made
successfully.

# Accepting an API connection from TWS

For security reasons, by default the API is not
configured to automatically accept connection requests
from API applications. After a connection attempt, a
dialogue will appear in TWS asking the user to manually
confirm that a connection can be made:

![conn_prompt.png](https://
interactivebrokers.github.io/tws-api/conn_prompt.png)

To prevent the TWS from asking the end user to accept
the connection, it is possible to configure it to
automatically accept the connection from a trusted IP
address and/or the local machine. This can easily be
done via the TWS API settings:

![tws_allow_connections.png](https://
interactivebrokers.github.io/tws-api/
tws_allow_connections.png)

Note: you have to make sure the connection has
been fully established before attempting to do any

requests to the TWS. Failure to do so will result in the TWS closing the connection. Typically this can be done by waiting for a callback from an event and the end of the initial connection handshake, such as [IBApi.EWrapper.nextValidId](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a09c07727efd297e438690ab42838d332) or [IBApi.EWrapper.managedAccounts](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#abd7e561f313bcc4c860074906199a46c).

In rare cases in which IB Gateway or TWS has a momentarily delay in establishing connecting to the IB servers, messages sent immediately after receiving the nextValidId could be dropped and would need to be resent. If the API client has not receive the expected callbacks from issued requests, it should not proceed assumming the connection is ok.

# Broken API socket connection

If there is a problem with the socket connection between TWS and the API client, for instance if TWS suddenly closes, this will trigger an exception in the EReader thread which is reading from the socket. This exception will also occur if an API client attempts to connect with a client ID that is already in use.

The socket EOF is handled slightly differently in different API languages. For instance in Java, it is caught and sent to the client application to [IBApi::EWrapper::error](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a7dfc221702ca65195609213c984729b8 "Handles errors generated within the API itself. If an exception is thrown within the API code it will...") with errorCode 507: "Bad Message". In C# it is caught and sent to [IBApi::EWrapper::error](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a7dfc221702ca65195609213

3c984729b8 "Handles errors generated within the API itself. If an exception is thrown within the API code it will...") with errorCode -1. The client application needs to handle this error message and use it to indicate that an exception has been thrown in the socket connection. Associated functions such as [IBApi::EWrapper::connectionClosed](https://interactivebrokers.github.io/tws-api/interfaceIBApi_1_1EWrapper.html#a9b0f099dc421e5a48ec290cab67a8ad1 "Callback to indicate the API connection has closed. Following a API <-> TWS broken socket connection...") and [IBApi::EClient::IsConnected](https://interactivebrokers.github.io/tws-api/classIBApi_1_1EClient.html#ab8e2702adca8f47228f9754f4963455d "Indicates whether the API-TWS connection has been closed. Note: This function is not automatically in...") functions are not called automatically by the API code but need to be handled at the API client-level*.