

Instant Gratification

Bloomberg Tradebook API

In view of the title of this magazine, it's probably safe to assume that a decent percentage of our readers are already exponents of API trading; it therefore seemed sense for us to take a trading API for a spin. So we have; Andy Webb, Automated Trader's Founder, and the Wrecking Crew practise their handbrake turns on the Bloomberg Tradebook API.

When the idea of reviewing the Tradebook API was first discussed, there was something approaching excitement among the members of the Wrecking Crew. (To learn all about the Wrecking Crew, see page 92 of the Q2 2010 issue of *Automated Trader*.) Many of them are or have been Bloomberg users and most of them are involved in automated or algorithmic trading¹ in some way or another, but none of them actually had prior hands-on experience of the Tradebook API. Therefore, once everything was installed and running, there was a certain amount of jostling over who would get first go behind the wheel. There was also rather a lot of speculation regarding the price, but more of that later...

Which API?

One thing that took the review team a moment to get its collective head around was which API we were actually dealing with. The API Developer's Help Site on the Bloomberg terminal actually lists three API developer products - Desktop API, Server API and Managed B-Pipe. However, those are all *Bloomberg* APIs, not the *Bloomberg Tradebook* API. *Bloomberg* APIs enable the extraction of just about any sort of data you could want from Bloomberg - ranging from historical news releases to real time tick data. In order to do any automated

or algorithmic trading, you need the *Bloomberg Tradebook* API, which is for order execution rather than data retrieval.

Having said all that, and despite the fact we're supposed to be reviewing the Tradebook API, we're also including a certain amount on the Bloomberg API (to be specific, the Bloomberg Desktop API) here. The reasoning is hopefully fairly self explanatory; if you want to get up and running fairly quickly with your auto/algo trading program, it makes considerable sense to be using one of the Bloomberg APIs alongside the Bloomberg Tradebook API. Apart from anything else, this will save considerable time, because the two APIs are programmatic cousins. They therefore share a lot of common concepts such as the way in which sessions are established, services are subscribed to and events are handled. That's not to say that you can't use a completely separate data source to feed your models, it's merely that unless there is data Bloomberg doesn't provide that's desperately required or some other completely compelling reason, then the additional effort is unlikely to prove worthwhile.

Getting started - download and installation

Our first thought in relation to downloading and installing the Bloomberg and Tradebook APIs was that a better job could be made of discriminating between

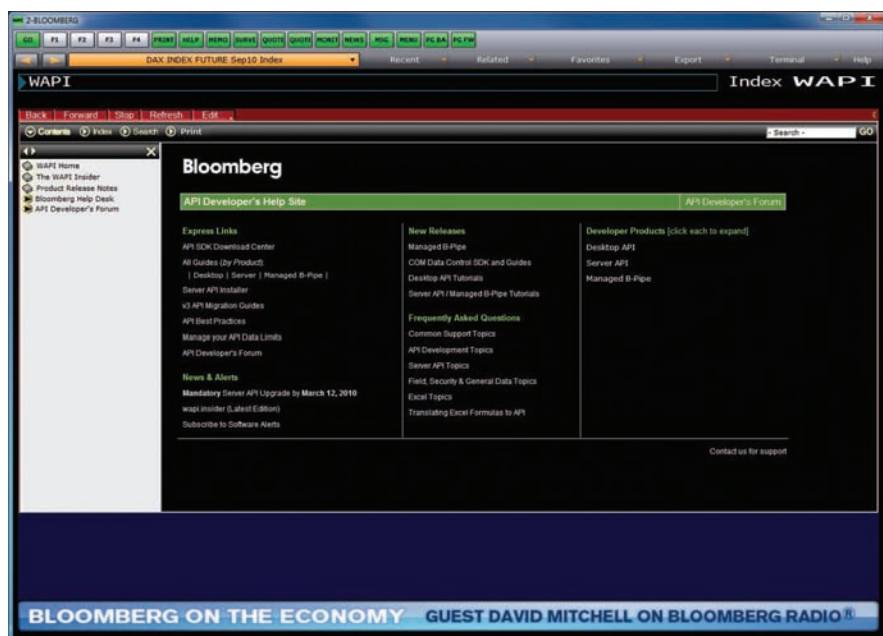


Figure 1

¹ See http://www.automatedtrader.net/Algorithmic_Trading.shtml for the distinction Automated Trader makes between automated and algorithmic trading.

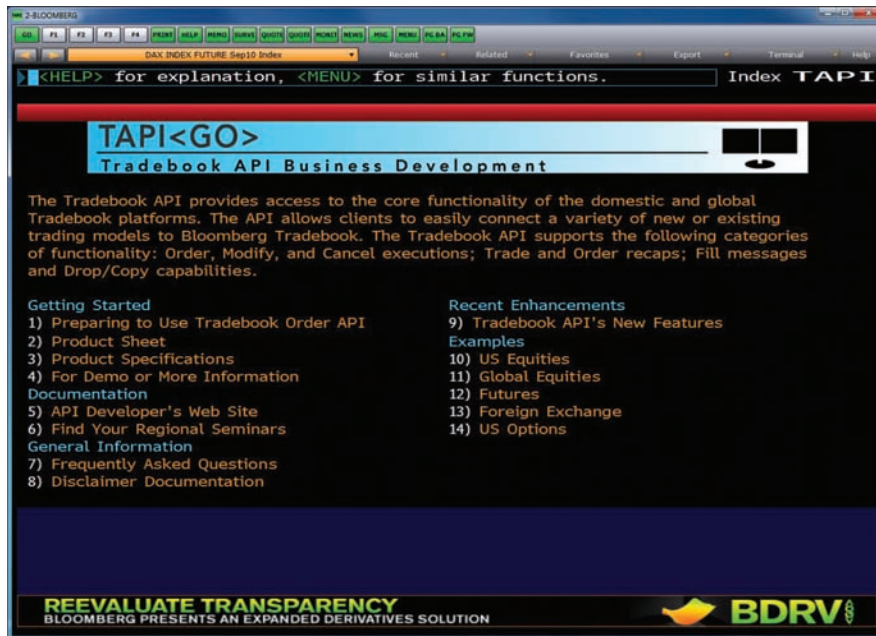


Figure 2

them. Hitting WAPI <GO> in the Bloomberg terminal took us to the API Developer's Help Site (see Figure 1) where we clicked on the API SDK Download Center link. This gave us the option of downloading the Desktop or Server API SDK, the Managed B-Pipe SDK, or the Desktop Pre-3.x SDK. As mentioned earlier, we had decided to focus on the Desktop API, so we downloaded that. As we had seen no reference to the Tradebook API on the API SDK Download Center page, we assumed we had just downloaded the Bloomberg Desktop API.

So we then hit TAPI <GO> in the Bloomberg terminal to go to the Tradebook API home page (see Figure 2). We couldn't see any direct link for "Download Tradebook API here!" - so we clicked the API Developer's Web site link and found ourselves back at Figure 1. Curious.

Hat Tips

Howard Stone is the Product/Business head for the Tradebook API and - to judge by the quality of the Tradebook API sample code he has knocked up - no mean shakes as a programmer either. He also has a very high tolerance level for less than intelligent questions - a level that we are happy to confirm we have thoroughly stress tested over the past few weeks...

Many thanks Howard!

So off we went to find the Tradebook API docs to see if they could shed any light, but they weren't any easier to find than the API itself. A quick email to the Tradebook API Product/Business head and the magic link was revealed: IDOC TRADEBOOK DEVELOPER <GO>. The Tradebook API Developer Guide contained the important sentence: "The Tradebook API Library is built on the Bloomberg Desktop APIv3". After reading that we finally realised we had probably already downloaded the Tradebook API along with the Bloomberg Desktop API and that our search was therefore over.

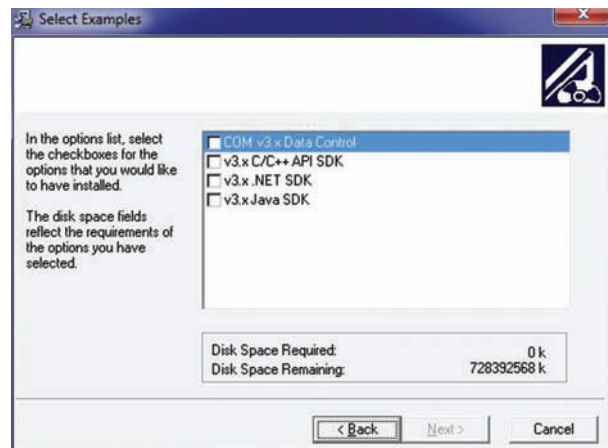


Figure 3

OK, so we're nitpicking, this wasn't a disaster but it did waste a bit of time and could easily be fixed with some rather clearer sign posting. It also rather stood out given that the rest of the install process was so smooth.

The install routine for the APIs gives you the option of installing the files for COM, C/C++, .NET or Java (see Figure 3). We opted to install all four, although most of our subsequent testing focused on COM and .NET. In addition to the API files, the install also includes a slew of example projects for the Bloomberg API for the various environments. These included everything from simple subscription for real time updates, to historical data downloads, to correlations, to VWAP calculations.

Tradebook API examples (Excel only) came separately and were downloaded via the Examples links on the

Order Sent	2	3	4	5
ClientOrderId	Q3UJFbc97Cq9mH			
StrategyType	SingleOrder			
NumberOfOrders	1			
NumberOfEntryConditions	1			
NumberOfExitConditions	1			
TimeInForce	GFD			
OrderNumber				
ExecutingBroker				
Account				
SecurityId	IXUO			
ContractType	Index			
Side	Buy			
OrderQuantity	10			
OrderType	MKT			
LimitPrice				
StopPrice				
DisplayType				
DisplayQuantity				
Entry ConditionNumber	1			
Entry ConditionType	Target			
NumberOfTargetConditions	1			
TargetCriteria				
SecurityId	IXUO			
ContractType	Comdty			
QuoteAttribute	Last			
ConditionOperator	LessOrEqual			
TargetPrice	128.89			
Exit ConditionNumber	1			
Exit ConditionType	Target			
NumberOfTargetConditions	1			
TargetCriteria				
SecurityId	IXUO			
ContractType	Comdty			
QuoteAttribute	Last			
ConditionOperator	LessOrEqual			
TargetPrice	128.87			
Notes				

Figure 4

Tradebook API home page. These include basic order entry and drop copy functionality, as well as examples of more complex trade types - such as the ability to base entries on certain time or price levels. In the case of price levels, these can be based on instruments other than the one being traded. (Just for the hell of it, we set up and tried a Dax index future buy at market order, based on the Bund future dropping to a certain level, with the trade exiting if the Bund dropped to a further lower level (see Figure 4). Whatever the non-merits of the trade, the order execution worked perfectly.)

The Bloomberg API - nuts and bolts

The Bloomberg API is not the primary object of this review, but because many Tradebook API users will also be using the Bloomberg API for data, we provide here a quick outline of how it works.

Depending upon which language you are developing in, specific Bloomberg API DLL, LIB or JAR files will need to be referenced or included in your code. These are automatically placed in the appropriate locations and registered as necessary by the

installation routine, so any effort relating to these should only involve checking you are using the current version(s).

One of the most important things to understand about the Bloomberg API is that it uses two fundamentally very different paradigms - Request/Response and Subscription. Which you use where in your code depends upon the type of data request being made. The Request/Response paradigm is for static data requests where the data concerned is unlikely to change, while the Subscription paradigm is for requests such as real time price updates.

Therefore the typical programming process for a static data request would be:

1. Create a session object to manage the connection with Bloomberg's data platform. (Multiple sessions can be created for redundancy.)
2. Use the session object to create a service object to open the required Bloomberg services. Four core services are available - Reference Data, API Field Information, Market Data and Custom VWAP. (Each service has an associated schema, which defines the operations that can be performed on it, the events that can be received from it and a list of associated elements, such as requests and messages.)
3. Once the relevant service has been opened, the next step service is to make a request. (See Figure 5, which is a C++ request for the last price traded in France Telecom and Microsoft with the request syntax highlighted.)
4. Process the response to the request (assuming the request is successful).
5. Finally - once the rest of the program has completed - stop the session. ▶

```
Request request = ref.createRequest("ReferenceDataRequest");

// add securities
Element secs = request.getElement("securities");
secs.appendValue("FTE FP Equity");
secs.appendValue("MSFT UW Equity");

// add fields
Element flds = request.getElement("fields");
flds.appendValue("PX_LAST");
flds.appendValue("Name");

session.sendRequest(request, null);
```

Figure 5

```
subscriptions.add("ESUO INDEX",
                  "LAST_PRICE, BID, ASK, VOLUME",
                  "",
                  CorrelationId((long long)0));
```

Figure 6

The typical programming process for a dynamic data request would follow a similar process but in step 3, the request would be framed as in Figure 6, which requests real time updates of the last trade, bid, ask and volume for the S & P E-Mini (with the subscription syntax highlighted).

Getting the data

Following the above basic programming workflow in order to acquire data via the Bloomberg API is relatively straightforward. However, if you're prepared to plagiarise the various examples that come packaged with the Bloomberg API download (which of course we were) it's positively trivial.

Just for the hell of it, we modified the C# sample code "Simple Intraday Ticks Example" provided with the API, so that rather than just displaying the ticks but being otherwise inaccessible, we could write the data to file for later use in building models. Figure 7 shows the first twenty or so lines of output for the Dax index future for a week's worth of tick data. What it doesn't show is the other approximately 250,000 rows of data below it...

time	type	value	size	conditionCodes	exchangeCode
2010-07-05T06:00:00	TRADE	5834	210		
2010-07-05T06:00:00	TRADE	5835	1		
2010-07-05T06:00:00	TRADE	5835	1		
2010-07-05T06:00:00	TRADE	5834	1		
2010-07-05T06:00:00	TRADE	5837.5	1		
2010-07-05T06:00:00	TRADE	5838	1		
2010-07-05T06:00:00	TRADE	5837	2		
2010-07-05T06:00:00	TRADE	5837	1		
2010-07-05T06:00:00	TRADE	5837	1		
2010-07-05T06:00:00	TRADE	5837	1		
2010-07-05T06:00:00	TRADE	5837.5	1		
2010-07-05T06:00:00	TRADE	5837.5	2		
2010-07-05T06:00:00	TRADE	5836.5	1		
2010-07-05T06:00:00	TRADE	5838	2		
2010-07-05T06:00:00	TRADE	5838	1		
2010-07-05T06:00:00	TRADE	5838	2		
2010-07-05T06:00:00	TRADE	5838	1		
2010-07-05T06:00:00	TRADE	5838	4		
2010-07-05T06:00:00	TRADE	5838.5	8		
2010-07-05T06:00:00	TRADE	5839	7		
2010-07-05T06:00:00	TRADE	5840	3		
2010-07-05T06:00:00	TRADE	5837.5	2		

Figure 7

We thought it might be instructive to run a series of repeated requests for data to see how quickly the API could serve it up, so we shoved an instance of the C# Stopwatch class into the sample code and off we went. To make things reasonably realistic, we opted to run the first batch of tests between 9am and 11am EST when both US and European markets would be active and the Bloomberg servers would be reasonably busy. We varied the periods requested to keep it interesting, but even in active weeks where the Dax was knocking

out approaching 400,000 transactions we found the API seldom took more than 35 seconds to deliver the data. In periods like that illustrated in Figure 7, it was nearer 25 seconds.

We also ran a series of tests on the S & P E-Mini future at the same time of day, which were - to say the least - intriguing. For example, on July 9th 2010 the contract posted 388,223 ticks (about the same as the Dax in a reasonably busy week) but these took on average around 105 seconds (rather than approximately 35 seconds for the Dax) to download. So it would appear that there are plenty of traders hitting the Bloomberg servers for historical E-Mini tick data...

The Tradebook API - nuts and bolts

As mentioned earlier, the Tradebook API follows a broadly similar programming model to its Bloomberg cousin. In the case of the Tradebook API, a service is directly tied to the individual market it covers - Futures, FX, Options, US Equities and Global Equities (but no cash bonds, which caused a certain moodiness among certain members of the Wrecking Crew - see *Steve K's grumbling in Crew Views below*).

The Tradebook API also replicates these services/markets in beta versions. "Beta" in this sense does not imply they are pre-production versions of forthcoming Tradebook API software releases, but that they are services/markets in which users can beta test their own trading models by firing blanks, as opposed to live rounds (more on the realism of this later).

So, if you connect to the `ftbapisvc` service and start sending orders you'll be trading live on Bloomberg Futures Tradebook. By contrast, if you connect to the `otapisvc.beta` service and start spraying orders about, you'll be doing so in the beta environment of the Bloomberg Options Tradebook.

As with the Bloomberg API, service requests and any ensuing responses and events are schema-based. In a nutshell if your code doesn't conform to the appropriate schema for a requested service - it ain't gonna work!

Once connected to a service, you have to establish a subscription (which effectively creates a communication channel) in order to actually do anything practical. To do this you also have to be logged into your Bloomberg terminal and already be

```
SubscriptionList subscriptions;
subscriptions.add("TOPICSTRING");
session.subscribe(subscriptions);
```

Figure 7a

```
SubscriptionList subscriptions;
subscriptions.add("//blp/ftxbapisvc.beta/user/7118543
?mode=Trade")
session.subscribe(subscriptions);
```

Figure 7b

enabled for trading on the Tradebook asset classes for which you are attempting to subscribe. Figure 7a shows the generic C++ code for accomplishing this; a specific example is shown in Figure 7b where “TOPICSTRING” is replaced by a real topic string.

To be valid, the topic string has to contain three pieces of information:

1. The name of the service to which you wish to subscribe - in Figure 7b this is `ftxbapisvc.beta`, the beta FX Tradebook market.
2. Your Bloomberg unique user ID (UUID) - in Figure 7b this is `7118534` (the UUID assigned to us for the review period).
3. The mode of the subscription request - in Figure 7b this is `Trade` (because we wanted to trade!), the other two available mode values here are `Modify` and `View`.

Once a valid subscription is established, order submission, modification and cancellation can be undertaken by sending `OrderRequest`, `ModifyOrderRequest` and `CancelOrderRequest` objects to the desired service. The Tradebook API will respond to these with an `OrderResponse`, `ModifyOrderResponse` or `CancelOrderResponse` if successful - and a `Reject` if not.

The functionality available via these three basic activities varies considerably from market (service) to market. For example, the `CustomerOrderFieldsType` for the FX Tradebook has fifteen possible elements, but that for the US Equity Tradebook has thirty-one. This disparity is partly related to the number of order types or algos available for each market². In the case of the FX Tradebook you have a simple choice of limit, stop limit and sweep - but for the US Equity Tradebook you can pick from a selection of sixteen, ranging from simple market or limit orders to Tradebook execution algos such as `BSmart™`, `HideAndBSmartAuto™` and `HideAndFire™`.

One thing that struck the review team was the

practical resilience built into the way that the Tradebook API operates. A good example of this is the channel reset event, which is triggered whenever Tradebook detects that your connected trading application has died. Incorporating this event into an error handling routine means that you can immediately re-subscribe to the service concerned. In the same routine you can also include an order recap request that will resynchronise your application's order status with Tradebook's. So if your application has (for example) missed some order fills or rejects while it has been out of action, it would automatically be updated and therefore display your correct overall order position. While you would logically hope to find such functionality in a trading API, it's nevertheless reassuring to see.

Incidentally, it's worth noting that because Tradebook API services are tied to individual market segments they operate independently as regards the channel reset event. So if you have one application handling your automated trading models for FX and another application for futures models, the demise of one application would not affect the other's connectivity to the Tradebook API.

Building and deploying the models

Having downloaded a selection of historical data and glued it together with real time updates from the Bloomberg API and taken a cursory glance at the Tradebook API Developer Guide, we thought we'd better put together a few trading models to see how it all worked out in practice.

At the trivial end of the spectrum, we had a very basic (irrational) model (mentioned above under *Getting started - download and installation*). While this showed that the functionality in that particular sample spreadsheet worked very well, this was hardly a realistic model - despite the protestations of the Wrecking Crew member who dreamt (snorted?) it up.

Another member of the review team came up with a rather more realistic (though hardly revolutionary) model based upon variable period (volatility and time weighted) Donchian channel breakouts for a basket of global futures markets. His original code was written in MATLAB, so obviously we should have used the COM interface to interact directly with the Tradebook and Bloomberg APIs. In actual fact we ended up performing a really horrible kludge and updating a trigger value in a modified version of one of the Tradebook sample Excel workbooks in real time from MATLAB. And as if that wasn't bad enough, we were ▶

² Other considerations simply relate to the market concerned - for example, the `TradingSystemCounterparty` element in FX Tradebook would clearly not be applicable to the US Equity Tradebook.

also feeding real time data to the MATLAB model from another Excel workbook connected to the Bloomberg API.

Of course this was all totally justified by the scientific need to know how the APIs would react to truly awful and incompetent programming (cough).

Admittedly this model was only putting through fairly sporadic orders of indifferent size, but the short answer is that the APIs both reacted rather well. The Bloomberg API kept the data flowing and the Tradebook API kept the trades happening.

Next into bat was a rather more wholesome effort consisting of a set of FX models in a C# app written by one of the Wrecking Crew (Martin S again - see Crew Views below). At present, the FX Tradebook API service offers limit and stop limit order types, as well as a sweep order type that attempts to deliver the average value requested.

Unfortunately we didn't get the opportunity to road test the stop limit and sweep orders as this particular model exclusively used limit orders. However, even with just these, some curious anomalies arose; for while the Tradebook API work flawlessly, we were a little puzzled by some of the non-fills we were getting on the beta Tradebook FX market. *(As mentioned, in the Q2 issue The Editor is legendarily mean with his pages, but he's even meaner with his editorial budget - which apparently didn't extend to funding a live trading account, so beta mode was all we could afford.)*

The FX models weren't particularly high frequency, so we were able to watch events fairly easily and after a while we noticed that a feature (bug) in the EURUSD model was causing it to post limit orders to sell significantly below the current price quoted on the Tradebook FX beta market. What was puzzling was that on quite a few occasions these limit orders weren't immediately clobbered. We also confirmed this behaviour by posting a few similarly off market limit orders manually via the basic FX template supplied with the Tradebook API. We don't know the techniques Tradebook uses to synthesise the beta market for FX, so we weren't sure if this meant that the Tradebook FX live market was extremely thin at the time the orders were being submitted (around 4pm EST) or if it

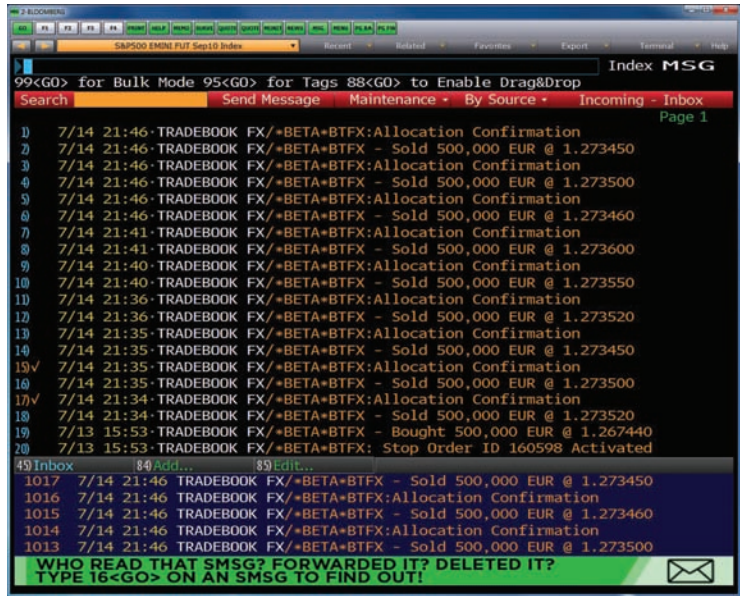


Figure 8

was another issue entirely. Whatever the case, on the matter of executing the orders transmitted by the application, the Tradebook API delivered. (Incidentally, if you can't be bothered to write your own handling code for FX fill reporting we discovered that you don't have to bother - FX fills are automatically dumped into the Bloomberg Terminal - see Figure 8.)

As we were keen to access some of Tradebook's execution algos via the API, our final automated trading model was for synthetic pairs in large cap US equities. In the end however, we ended up using just one - Tradebook's Algo BSmartAuto™ algo. Like the BSmart™ algo, this uses intelligent posting to break up orders and submit individual slices to the most active venues based upon real-time market information and a proprietary probing model. However, in addition

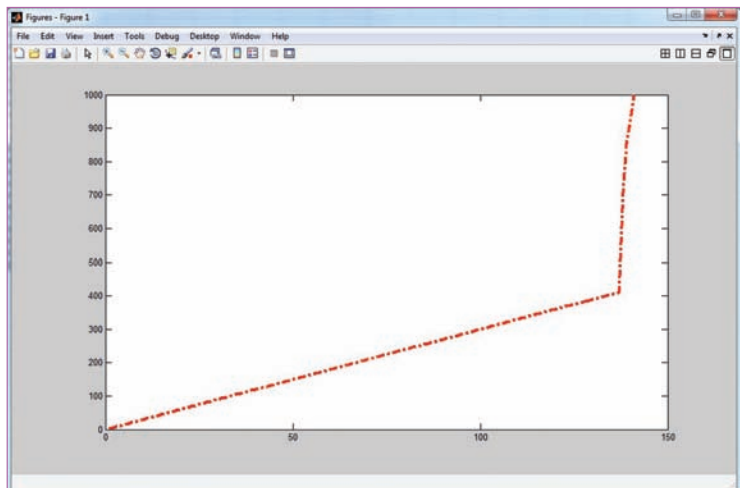


Figure 9

Support

In an environment like the Automated Trader review lab, where we are continually swapping machines in/out and reconfiguring IP addresses and secure connections etc, software and hardware glitches are something of a fact of life. Having said all that, we've had until recently a fairly painless few months. Unfortunately some idiot must have remarked on this out loud instead of keeping their fat mouth shut. Needless to say, the gods of technology overheard and clearly decided that condign punishment was in order - and boy did they ever deliver...

We started the review on a 32 bit Windows XP workstation and for a few days all went well. Then one morning it wouldn't boot - no POST screen appeared, no opportunity to go into the BIOS. Nothing, rien, nichts, nada, niets, nic - apart from a black screen and whirring fans. The only good thing that arose from this was that it gave us a chance to try out Bloomberg global support. As a security measure, Bloomberg locks the install of its software to a particular machine, so if you want (need!) to reinstall on another machine you need a new license key. This went both fast and smoothly, the phone was answered immediately and we were quickly put through to the relevant support team who cheerfully dished out the requisite number straight away. They probably wouldn't have been quite so cheerful, had they realised how frequently they would be hearing from us...

Our next machine was running XP 64 bit and that lasted all of 24 hours before the PSU exploded. Needless to say it was some esoteric top of the range mission critical server PSU (supposedly), which meant that a replacement would take at least a week to arrive. So it was time for machine number three and

another call to Bloomberg support. They were again efficient and fast and we were soon up and running on a Windows 7 64 bit workstation. Happily this time the hardware didn't break - but Windows 7 did. To be more precise, its networking components went AWOL. One Windows 7 reinstall and another call to our increasingly close friends at Bloomberg global support later and we were finally up and running on a more permanent basis.

In addition to bugging the Product/Business head for the Tradebook API on a fairly regular basis, we thought we'd better share the grief around by pestering the Tradebook API support desk as well. We had an opportunity to do this when the API refused to transmit orders via any of the beta markets and instead threw us a load of "account not authorized" messages.

Interestingly, each of the five markets currently available via the Tradebook API appears to have its own separate support team. We did wonder if that might not pose problems in a live trading situation for traders running inter market strategies who would have to contact multiple support teams. Our particular problem actually involved *all* Tradebook markets (none of them would work) but we were impressed with the speed with which we were passed among the teams and the issue resolved. It emerged that while we had originally been enabled for the beta markets at a high level, we had not been enabled for any individual instruments within those markets, so each API market support segment had to turn the relevant instruments on. While the Tradebook API support teams were all fast, efficient and helpful, we couldn't help feeling that a single frontline support team to handle basic issues (like our enablement problem) might be worthwhile.

it also uses a short term price forecasting model to automatically adjust its aggression levels.

This model was again written in MATLAB, but on this occasion we connected directly to the Tradebook API rather than hacking about in the rough via Excel. While we were of course only accessing the beta rather than the live market, the results were nevertheless interesting. Figure 9 shows the cumulative execution profile for one of a set of small initial orders - in this case for Bank of America Merrill Lynch. The average trade price achieved was 15.56 during a period (from 14:48:36 to 15:11:31 on July 14th) when the stock was typically trading two to four cents higher on the primary market (*see the area between the red and green vertical lines Figure 10*). Just for the entertainment value, we also ran the model with far larger order sizes in the 500K to 1m shares range. Apart from BAC,

symbols traded included PEP, AAPL, MSFT, KO, GE, BA, JPM, T and XOM. While the .MAT file recording the individual order slices reached a fairly generous size, the Tradebook API just kept plugging away. On that point, it should be said that we didn't manage to blow up the APIs or the Bloomberg terminal once during the review period - Excel fell over a fair few times, but we didn't feel we could fairly lay that at either of the APIs' doors. Considering the abuse to which we were subjecting the technology, we were rather impressed about that.

Beta realism

If you're planning on using the Tradebook API, you'll be spending a fair bit of your time using the Tradebook beta services. Therefore one question we felt compelled to consider was how realistic were the fills achieved on the beta markets? (Apart from checking code, these



Figure 10

markets represent a crucial step in evaluating your models' viability after in and out of sample testing on historical data has been completed and before live deployment - so realism is literally vital here.)

As mentioned earlier, none of the Wrecking Crew had prior knowledge of the Tradebook API, but fortunately several of them have been or currently are Tradebook users. We were therefore able to tap into their experience of using Tradebook for live trading as a rough benchmark for our beta trade executions. (The word "rough" in that previous sentence is significant at several levels - especially if you've met the Wrecking Crew members in question.) Their general verdict on the realism of fills we were getting on the beta market was "pretty good". We didn't unfortunately have a Tradebook options user to hand, but those with futures, FX and global equities experience felt that the fills were reasonably in line with what they would expect to see in live markets. US equities received a similarly positive response, though one user of Tradebook algos felt that a few of the BSmartAuto™ fills for buy orders were possibly a touch optimistic, but that some of the sell order fills were actually slightly pessimistic.

Documentation and examples

As Martin S says (see "Crew Views" box below), the documentation for both Bloomberg and Tradebook APIs is impressive. PDF linking is used extensively, which means that drilling down through the programming hierarchy of both APIs is easy. As Martin mentions, the Tradebook APIs looked like a work in progress with fewer code samples and only in C++, but from a practical perspective we didn't find this a huge problem as the basic functionality of each programming element was explained.

The code samples and example applications/workbooks are really outstanding - some of the very best we've seen to date. From the review team's perspective they

added a lot of value - both in terms of saving implementation time and obtaining a clearer understanding of how the APIs worked.

The \$64K question

So is it any good? Well, as you can see from page 92 of the Q2 edition of *Automated Trader* ("The Automated Trader review process and team"), if we've bothered to write this far then it's at least half decent. In fact it's quite a bit more than that...

The consensus among the review team was that the Tradebook API had pretty much hit the sweet spot in terms of combining functionality and ease of use. As one crew member remarked: "If I've got models to deploy, I want to get on and deploy them ASAP - and I think I can do that with this." Partly we felt that this time to market advantage was down to the straightforward architecture of the APIs, but that the example applications and spreadsheets also made a huge difference because of the opportunity to "plagiarise and play". (The only improvement we could suggest here was that the Tradebook API would benefit from a few .NET, Java and C++ example apps, as provided with the Bloomberg API.)

When it came to making a value judgement about the Tradebook API, we had a bit of shock when we asked about the price tag. If you're already a Tradebook user - it's free; there are no additional charges over and above the commissions you'd be paying anyway as a Tradebook client. The Bloomberg API is included as part of the Bloomberg Professional service subscription and (depending upon the relationship) Tradebook *may* even be able to accommodate the cost of that as well. Since several of the review team are currently paying between GBP1000 and GBP1500 per month for FIX or other trading APIs, this news caused a bit of a stir. Obviously, your individual value judgement on the Tradebook API will ultimately depend upon how good you are at negotiating your commissions with Tradebook.

While we have no insight on that, we can say the following five things about the Tradebook API:

- It works (very well)
- It's easy to learn how to use
- You can use it to realistically test the real time performance of trading models
- ...and deploy them rapidly...
- ... and (if you're any use at commission haggling) - it's a steal

Crew Views

Since they spend most of their time bickering and appear incapable of agreeing on just about anything, it's often difficult to synthesise all the diverse views of the regular team plus the Wrecking Crew into the main body of the review (I do my best, but it's really a pretty hopeless task). This box therefore contains some of their individual views in all their forthright glory. (In addition to one of the review team members introduced in the Q2 edition of *Automated Trader*, there are views from a couple of Wrecking Crew guest artistes - kept anonymous to protect their current/future employment prospects.)



The Review Team at work - from left to right, The Founder, Steve K, Horace, Martin S, Peter Farrow.

Peter Farrow (*Linux-bore server room guru*): Software installation of both the Bloomberg application and the APIs was straightforward. However, Bloomberg could do themselves and their users a favour by clearly publishing the firewall ports that need to be opened for everything to function. We expected to have to open some, but when we called Bloomberg support because the product didn't connect, we were told that our firewall was blocking it (which we had figured out for ourselves) - but we weren't told which ports were involved! It wasn't a major issue, as we simply figured out the relevant port numbers by tracking the blocked connections in the firewall log and cross-correlating those with known Bloomberg IP addresses. Nevertheless, it would hardly be rocket science to put a splash screen in the install routine, warning that you will have to open these X ports for these Y IP addresses.

Steve K (*prop trader, European bank*): The API functionality looked solid to me, but it's frustrating that you can't trade cash bonds through it - especially in view of some of the inter market trades I typically do. None of the automated models I run are particularly high frequency, so the convenience factor of being able to trade everything through one API would be an outright winner, rather than having to code to different interfaces as at present. I realise that adding cash bonds to the API would be a lot easier said than done, given the fragmentation of the bond market, but it would almost certainly make using the Tradebook API an absolute no-brainer for me. The beta environment is great - simulated executions were realistic and the ability to access Tradebook execution algorithms (especially the decent selection on offer for US equities) through the API is particularly useful.

Martin S (*programmer/quant, US proprietary trading firm*): From a programmer's perspective, both the Bloomberg and Tradebook APIs are straightforward to deal with. Nice to have consistency between the two, though there are a few key differences to be aware of. For example, you can re-subscribe to a topic with a different event set via the Bloomberg API, but not the Tradebook API, where instead you have to unsubscribe from the original topic before initiating a new subscription request.

You also need to remember that just because the Bloomberg API happily feeds you a price, that doesn't necessarily mean that the Tradebook API will let you trade on it. For example, the Bloomberg API will quote EURGBP to five decimal places, but if you try to trade on that you'll get a "Rate XXXX does not conform to the pip increment of 0.5 pip" message from the Tradebook API. Similarly, the Bloomberg API will quote you GBPEUR all day long but the Tradebook API will say "Currency Pair not supported". Obviously neither of these represent major obstacles to using the Tradebook API, as they can easily be worked around - they're just things you need to remember.

Docs were very good, although the Tradebook API docs looked slightly like a work in progress when compared to the Bloomberg API docs. While the Bloomberg API docs included Java, .NET, C++ and C code examples, the Tradebook API docs offer fewer examples and only in C++. That isn't a big problem, because all the syntax and available functionality are listed and explained anyway and the excellent range of example Excel sheets cover most of the possibilities across the markets accessible to the Tradebook API.