YellowJacket™ Pricing Service API & SDK

INTRODUCTION

YellowJacket is a powerful communication tool that converts traditional IM conversations into hard-to-obtain OTC market data. This market data is aggregated and presented in a proprietary Quote Console that provides traders a consolidated view of their broker quoted markets . Each row in the Quote Console represents a unique market that was either sent or received and provides relevant information such as bid/offers, counterparty information, and time stamps, etc.

YellowJacket provides an API and SDK to access and interoperate the data the Quote Console displays. This document will describe how to use the API and SDK to do the following:

- Create a real-time market data feed from YellowJacket to any third-party application
- Integrate a custom pricing model/application and display model-calculated theoretical prices and Greeks directly inside the Quote Console
- Optimize the integration to avoid unnecessary and costly re-pricing calculations

PRICING SERVICE API OVERVIEW

TheYellowJacket application can communicate with any third-party application over a local or network TCP socket connection. This communication conforms to an XML API schema provided by YellowJacket. As described above, this connection can be utilized to display real-time price updates inside the Quote Console.

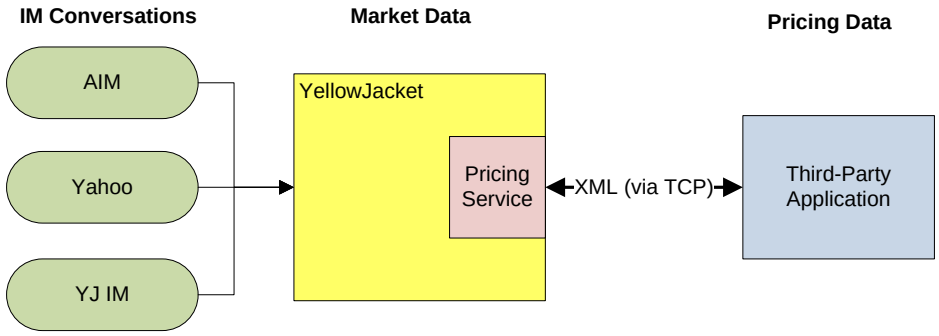


Figure 1: Pricing Service API Overview

By default, YellowJacket will send an XML PriceUpdate message whenever the following events occur:

- A new market is sent or received via IM
- An existing market is updated with a new bid or offer
- The YJ user double clicks the value to request re-pricing

These PriceUpdate messages allow third-party applications to respond with XML PriceResponse messages for each market. A PriceResponse message may return the following information for each leg of the market::

- A unique Id for each leg (sent from YJ)
- Theoretical Price
- Gamma
- Delta

- Vega
- Theta
- Rho
- Vol (The volatility used by the model to calculate a price)

These fields may be extended to support additional data interchange (see custom key/value pairs below).

Once received by YellowJacket an aggregate theoretical price is calculated and displayed in a column next to the applicable market; the Greeks are displayed at the top of the screen when the market is selected.

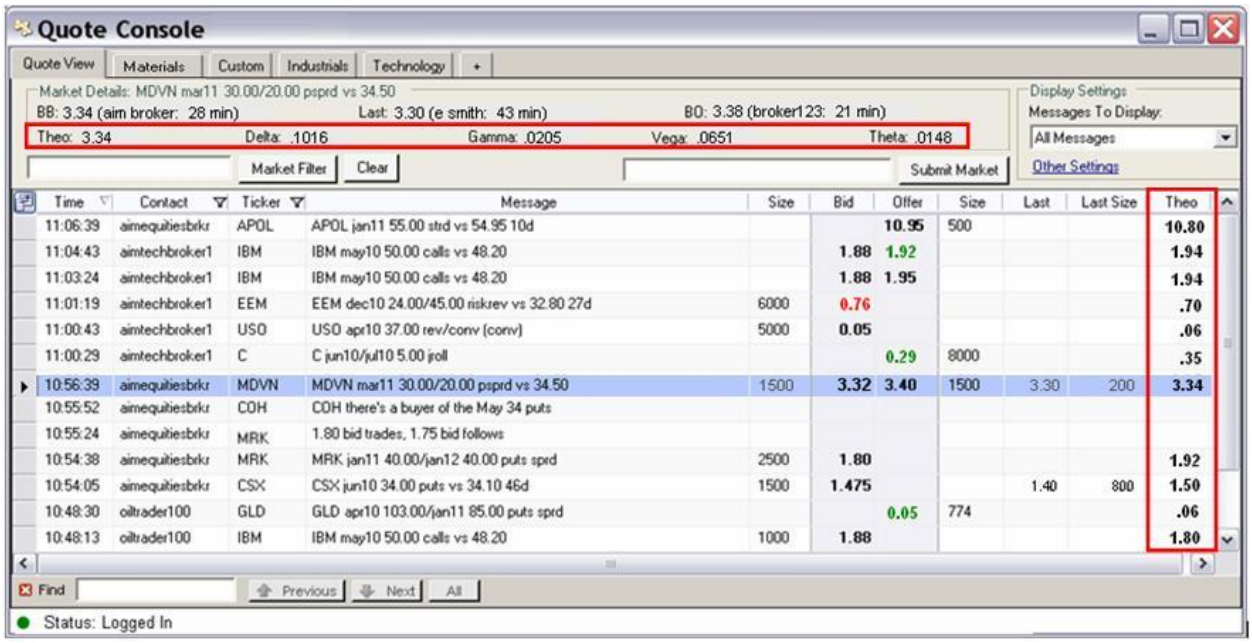


Figure 2: Quote Console with Custom Theoretical and Greeks

PRICING SERVICE API CONFIGURATION DETAILS

To enable the Pricing Service API navigate to *Manage->My Preferences->Analytics* from the YellowJacket Main Console and select the "External" Pricing Service from the dropdown. YJ can be configured to listen for connection requests on a user-specified port; to change the port, or to limit requests to local connections only, use the settings fields at the bottom of the page.

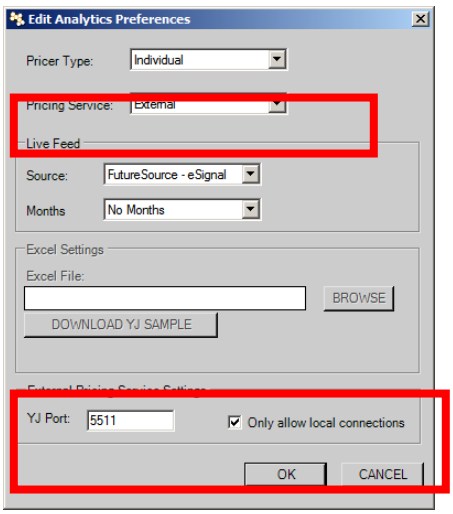


Figure 3: External Pricing Service Configuration

PRICING SERVICE API CONNECTION & MESSAGE DETAILS

Once YellowJacket is properly configured a third party application initiates a TCP socket connection that will establish a session between the two applications. The pricing application then sends and receives messages with YJ as follows:

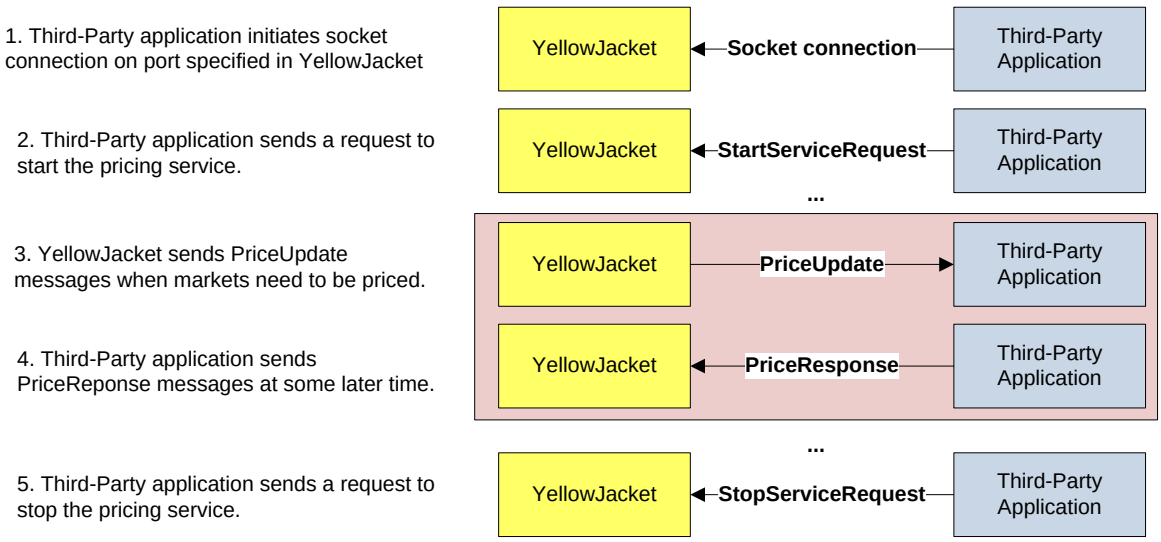


Figure 4: Pricing Service Connection

All messages will be serialized instances of the Envelope API class to ensure a consistent root type for serialization/de-serialization and to ensure consistent message delimiters.

STARTSERVICEREQUEST

Once a socket connection is established the third-party application must send a `StartServiceRequest message`. The `StartServiceRequest` includes the following information:

- Version ("Equities 1.0" should be sent as the current version)
- Service ("Pricing" is the only currently supported service)
- Optional key/value pairs (included to support extensible meta-data)

PRICEUPDATE

A `PriceUpdate` message is sent by YellowJacket whenever new market data is received that may necessitate a re-price of a certain market. A `PriceUpdate` message contains the following information:

- An action field that contains the reason/source of the `PriceUpdate` message. The current actions include "`Price`" and "`Remove`".
- Either of the following (but never both):
  - A `MarketDetail` object (used when requesting a price updateon a market).
  - A market Id (used when removing a market)

To prevent costly re-pricing calculations, the.NET SDK (discussed in detail below) allows the user to filter/ignore `PriceUpdate` messages that do not meet certain requirements.

PRICERESPONSE

A `PriceReponse` is initiated by a third-party application whenever new pricing data (theoretical and/or Greeks) is calculated and needs to be supplied to the YellowJacket application. A `PriceResponse` message contains the following information:

- A required unique Id for a market (originally supplied by a `PriceUpdate` message)
- Information for each leg including:
  - A required unigue Id for each leg (originally supplied in a PriceUpdate message)
  - An optional theoretical price
  - Optional greeks (Gamma, Delta,Vega, Theta, Rho)
  - An optional volatility value which was used to calculate the theoretical price
  - Optional custom key/value pairs
- An optional "Theoretical override" that allows the third-party application to override the YJ aggregate calculated theoretical.
- Optional implied volatility values for the best and/or last quotes on the market.

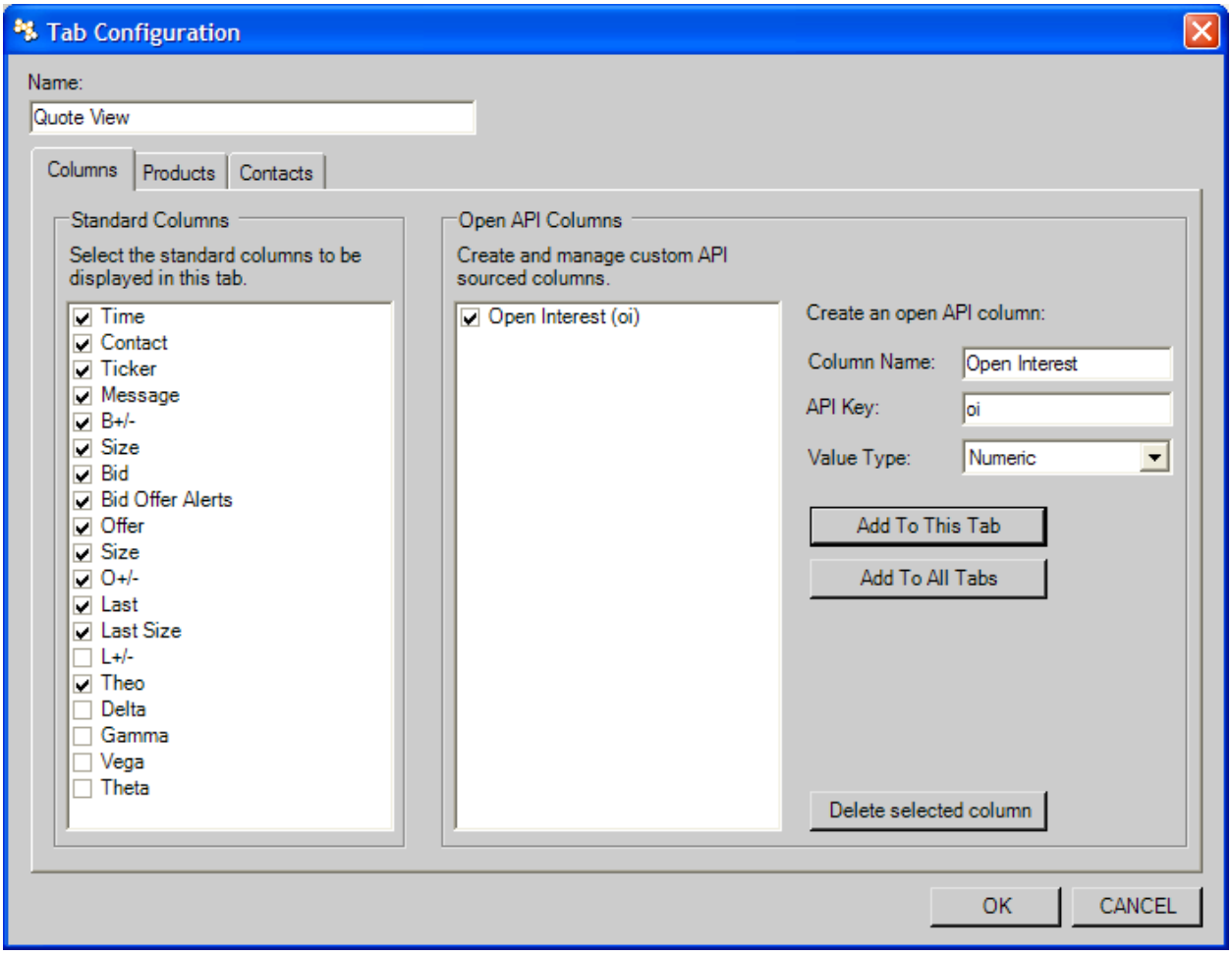
When the YJ application receives a `PriceReponse` message it will calculate an aggregated theoretical and set of Greeks by summing the component pricing information across all legs (taking buy/side direction and ratios into account) and will display the aggregated values in the Quote Console. If the user would like to override the leg-calculated theoretical price they may do so by supplying a "theoretical override".

OPEN API

In addition to the fields defined by the AppLink API, it is possible to send additional data in user defined fields; which can be displayed in user created columns in the Quote Console's Quote View.

To send user defined fields and values, a user must set key/value pairs in the PriceResponse message. To display those values, a user must configure matching columns by editing a Quote View tab (right click on a tab in the Quote Console and select "Edit…"). In the accompanying dialog (see screenshot below) a user has added a new "Open Interest" column. The new column has a key "oi", which must be sent as the key in a key/value pair in a PriceResponse message.

Open API columns can be of Numeric or Text type. Numeric columns will be sorted numerically, and Text columns can be sorted alphabetically.



PriceResponse messages can contain Open API key/value pairs and/or theoretical prices, values can be sent separately or together in one message.

An example PriceResponse message is below, in this case an "oi" value of 100 is being sent for the specified market.

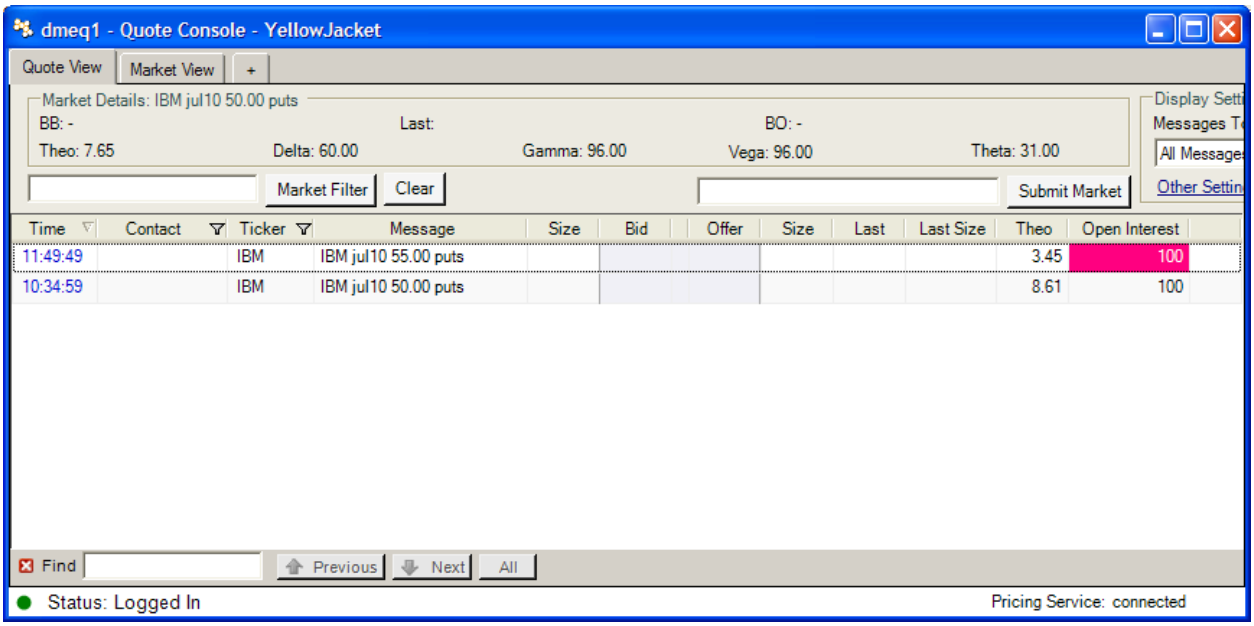
```
<?xml version="1.0" encoding="utf-16"?>
<Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Content xsi:type="PriceResponse">
    <MarketDetailId>Market-262903564</MarketDetailId>
    <KeyValues>
      <Key>oi</Key>
      <Value>100</Value>
    </KeyValues>
  </Content>
</Envelope>
```

The Open API also supports sending cell formatting information in addition to a value. A background and foreground color can be sent – this can be useful to alert a user to a particular value. To send display formatting information use the following format for the Value: <Value>value=yourValue,foreground=color,background=color</Value>

Color's can be specified in hexadecimal format or using standard HTML color names, an example is as follows:

```
<Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Content xsi:type="PriceResponse">
    <MarketDetailId>Market-262903564</MarketDetailId>
    <KeyValues>
      <Key>oi</Key>
      <Value>value=100,foreground=White,background=#FF0080</Value>
    </KeyValues>
  </Content>
</Envelope>
```

This message resulted in following Quote View update:



The SDK contains methods that help simplify sending Open API values, this is described in a section below.

STOPSERVICEREQUEST

Once a service has been started it can be stopped using the `StopServiceRequest` message. The `StopServiceRequest` message includes the following information:

- Service (”Pricing” is the only currently supported service)
- Optional key/value pairs (included to support extensible meta-data)

Communication between YellowJacket and the third-party application is asynchronous; a PriceReponse can be supplied to the YellowJacket application at any time and a PriceUpdate message does not require a PriceResponse.

APPLINK PRICING SERVICE SDK

To expedite integration efforts YellowJacket provides a .Net 2.0 SDK (YJ.AppLink.dll), a sample application that uses the SDK (YJ.Sample.exe), and all accompanying source code. The SDK provides a number of data structures, methods and events that can be used to simplify and manage the interaction between your application and YellowJacket.

Furthermore, the SDK allows you to keep a record of all OTC markets and selectively update specific markets in the Quote Console based on triggers or events driven by your application. Using the SDK, you can:

- Update the Quote Console with theoretical values from your model automatically as new prices are received from your brokers and counterparties.
- Update or modify your model and then revalue all or some markets in the Quote Console automatically.
- Filter out markets that do not apply to your business and only price and update the ones that do.
- Update or modify the swap level for a specific month, and then re-price and update only the markets that contain a leg with that month.
- Update your model with pricing from your OTC brokers and counterparties in real time

Using the SDK is optional; it is possible to connect directly to the YellowJacket application without it.

SETTING UP A CONNECTION USING THE APPLINK SDK

The AppLink Pricing Service SDK provides methods to connect and disconnect from YellowJacket's TCP pricing feed. Once connected, the SDK provides events that trigger whenever a `PriceUpdate` message is received from YJ.

To create a pricing service connection using the SDK do the following:

1. Create and configure a `YJ.AppLink.Session` instance:
   ```
   YJ.AppLink.Session session = new YJ.AppLink.Session();
   ```
2. Optionally set a non-default IP address and port by setting `session.IpAddress` and `session.Port` respectively
3. Establish the connection:
   ```
   bool connected = session.Connect();
   ```
4. If the connection is successful, configure the `session.PricingService` by setting the `PriceUpdateRequested`:
   ```
   session.PricingService.PriceUpdateRequested +=new
   PriceUpdateRequestedHandler(PricingService_PriceUpdateRequested);
   ```
5. Call `session.PricingService.Start()`

After the steps outlined above are completed, the `PriceUpdateRequested` listener will be invoked whenever a PriceRequest is received. The user can choose to ignore the request, immediately send a response back to YJ, or price the market at some later time.

PRICE A MARKET USING THE SDK

As described above, sending prices back to YellowJacket occurs independent of price requests. To send a price back to YJ (at any time) using the SDK do the following:

1. Initiate a successful connection to the pricing service using the steps outlined above
2. Get a MarketData instance to price from a PriceUpdateRequestedEvent, or by finding market(s) in the cache (described below).
3. Iterate through the MarketData's `Options` array.. Calculate and set any or all values on each option leg (`TheoreticalPrice`, `Delta`, `Gamma`, `Vega`, `Theta`, `Rho`, `ImpliedVol`, `Vol`).
4. Iterate through the `Stocks` array and set the price using the `MyPrice` property.
5. Call `marketData.SendPriceResponse()` to send the response to YJ.

SEND OPEN API VALUES WITH THE SDK.

The SDK contains methods that simplify sending Open API values. The following methods are defined in the MarketData class:

```
/// <summary>
/// Adds a key/value pair to a market, along with foreground and background
/// cell formatting information.
/// Specify System.Drawing.Color.Empty to clear to omit a foreground or
/// background color.
///
/// Note: this method adds the key/value pair to the market's PriceResponse,
/// but does not send the response message
/// </summary>
public void AddOpenAPIKeyValue(string key, string value,
            System.Drawing.Color foreground, System.Drawing.Color background)


/// <summary>
/// Adds a key/value pair to a market, without formatting information
///
/// Note: this method adds the key/value pair to the market's PriceResponse,
/// but does not send the response message
/// </summary>
public void AddOpenAPIKeyValue(string key, string value)


/// <summary>
```

```
/// This method builds and sends a PriceResponse message with only Open API
/// key/value pairs,
/// it does not send or update theoretical price information.
///
/// </summary>
public void SendOpenApiValues()
```

SEARCHING THE MARKET CACHE USING THE SDK

When the Pricing Service SDK receives a `PriceUpdate` message, a `MarketData` object is created and stored in the SDK's cache. The cache will store all markets that have been sent from YellowJacket. The SDK also provides methods to search the cache for specific markets.

To get info on a certain market call `YJ.AppLink.Pricing.PricingService.GetMarketData (string id)`.

To get all markets call `YJ.AppLink.Pricing.PricingService.GetAllMarkets()`.

To search for specific markets you must first create a filter; filters can check a single condition or can be composites of multiple filters. To create a filter simply implement the `YJ.AppLink.Pricing.IMarketFilter` interface—this interface only has one method `PassesFilter(MarketData mkt)` that must return a `bool` value.

A sample HasTermFilter (checks if a market includes a particular term), filter is included in the SDK. Furthermore, some composite filters are also included: `AndMarketFilter`, `OrMarketFilter` and `NotMarketFilter`.

Once a filter is implemented, simply call `YJ.AppLink.Pricing.PricingService.GetSelectedMarkets(IMarketFilter selectionFilter)` to receive an array of `MarketData` objects that match the selected filter.

IGNORING CERTAIN PRICE REQUESTS

The SDK can be configured to pass all PriceUpdate messages through to your application, or you can set specific filters to minimize pricing requests to your model. To ignore certain price requests, simply create and install a filter on the pricing service:

1. Create the appropriate filter or composite filter by implementing the `YJ.AppLink.Pricing.IMarketFilter` interface
2. Install the filter by calling `SetMarketFilter(IMarketFilter filter)` on the `YJ.AppLink.Pricing.PricingService`

PRICING SERVICE SAMPLE APPLICATION

YellowJacket also includes a sample application (YJ.Sample.exeto illustrate how to best use the SDK. The AppLink Sample code is included in the `Sample` directory. It consists of two C# files: `SampleForm.cs` and `SamplePricingModel.cs`. To use the AppLink Sample you must do the following:

1. First, ensure the "External" pricing service is configured and enabled in YellowJacket. In YJ, Navigate to *Manage->My Preferences->Analytics* and select the "*External*" option from the dropdown. Next, ensure the **YJ Port** is set appropriately or use the default value.
2. Run `YJ.Sample.exe, or compile and run the source code project.`
3. Ensure the **YJ IP Address** is set properly. This will be `127.0.0.1` unless YellowJacket is running on a separate computer.
4. Ensure the **YJ Port** is set to match the value set in the YellowJacket application.
5. Press the **Connect To YJ** button.

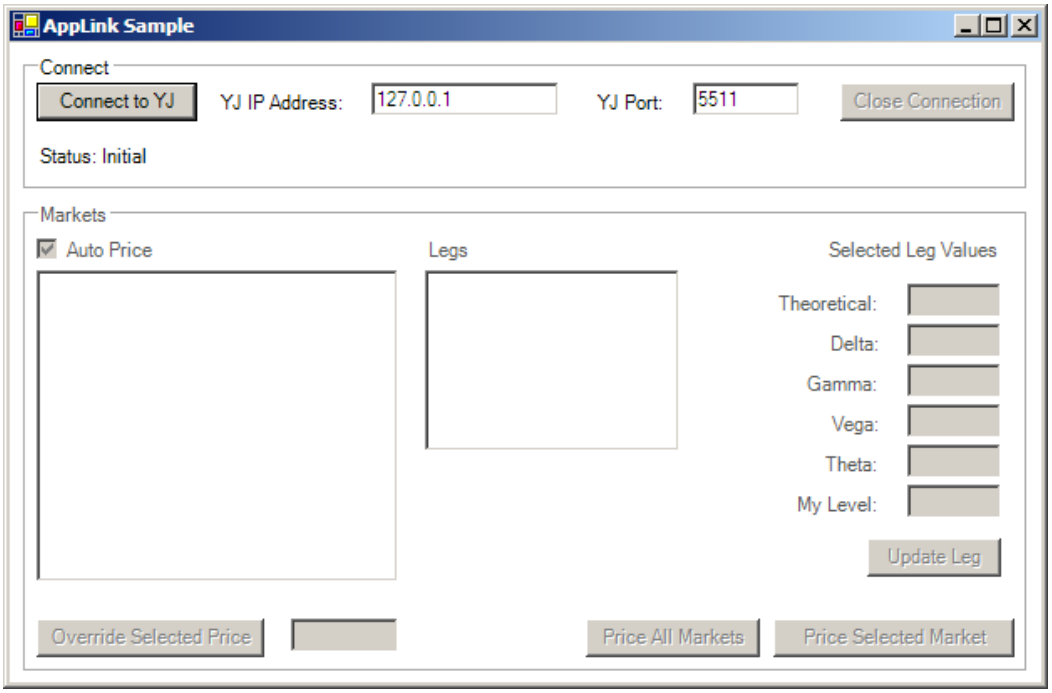


Figure 5: AppLink Sample when First Loaded

USING THE SAMPLE APPLICATION

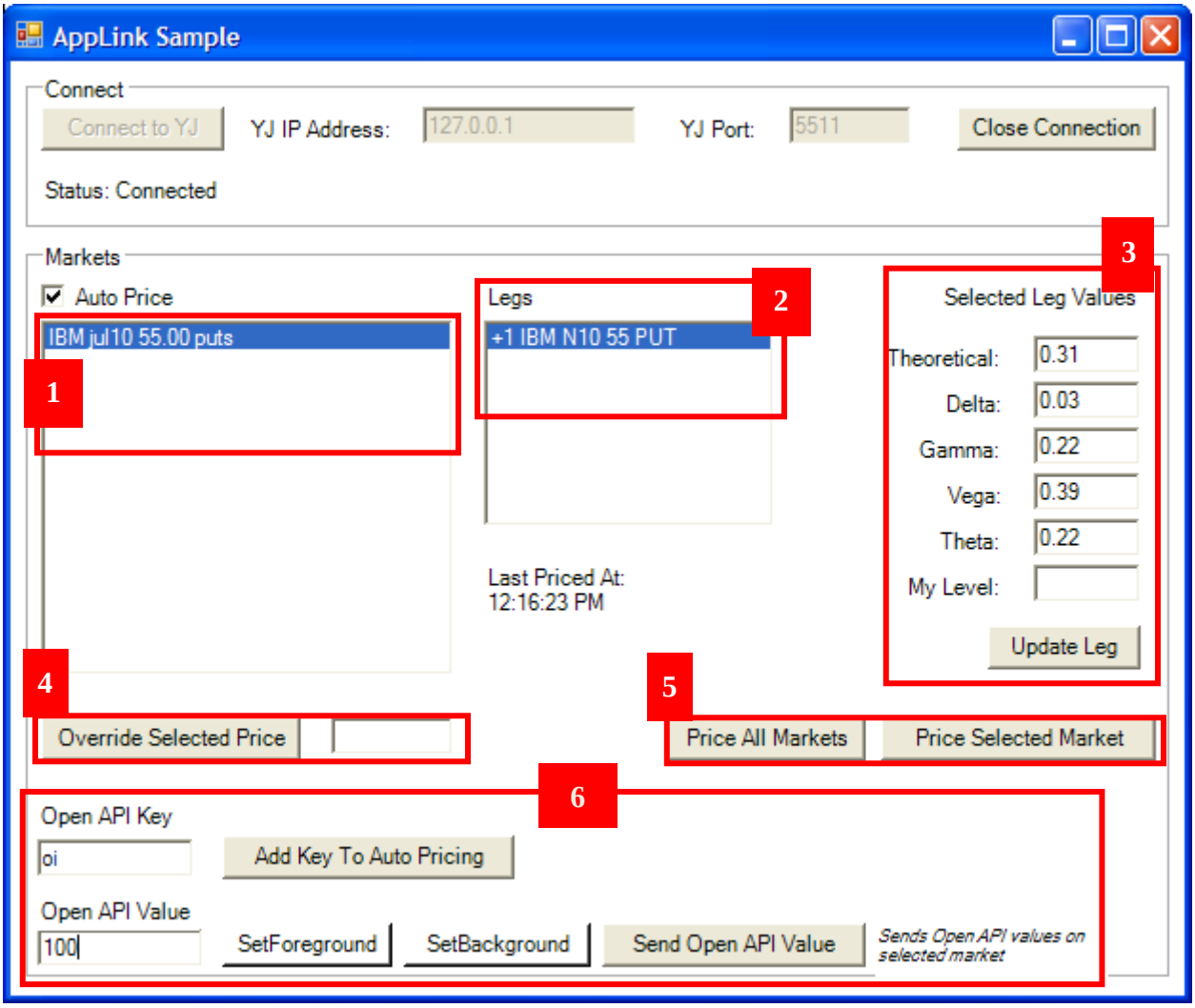


Figure 6: AppLink Sample with a Few Markets

1. MARKETS AREA

As markets are received by YellowJacket they will be displayed in this listbox. Selecting a market will list that market's legs in the Legs Area (#2). Checking the **Auto Price** checkbox will automatically price a market as it is received by the AppLink sample; if the Auto Price checkbox is not pressed then the user must manually select the market and press the Price Selected Market button.

2. LEGS AREA

After selecting a market (using #1) that market's legs will be listed in this listbox.

3. PRICING FIELDS

Selecting a leg (using #2) will display the calculated theoretical and Greeks for that leg. In the sample application these values are simply random numbers generated every time the market is priced. To override a particular value, change the value in any of the text fields and press the **Update Leg** button (the price will automatically be re-calculated and displayed in YJ).

4. OVERRIDE SELECTED PRICE

As described in the `PriceReponse` section of this document the API includes a method to "override" the leg-calculated theoretical price. This area of AppLink sample exposes this functionality for demonstration purposes.

5. PRICE SELECTED OR ALL MARKETS

Pressing the **Price Selected Market** button will price the market that is selected in Markets listbox. Pressing the **Price All Markets** button will price all markets that the AppLink Sample is aware of.

6. OPEN API CONTROLS

Enter an Open API key in the and click the "Add Key To Auto Pricing" button to add a randomly generated value each time a market is auto priced.

Use the Open API Value field and the color buttons and "Send Open API Value" buttons to send a specified value on the selected market.

ADDITIONAL INFORMATION OR QUESTIONS

For additional information or questions please email support@yjenergy.com, IM yjshelp, or call (770) 738-2101 and choose option 5.

APPENDIX I: XML SCHEMA

see Classes.xsd file