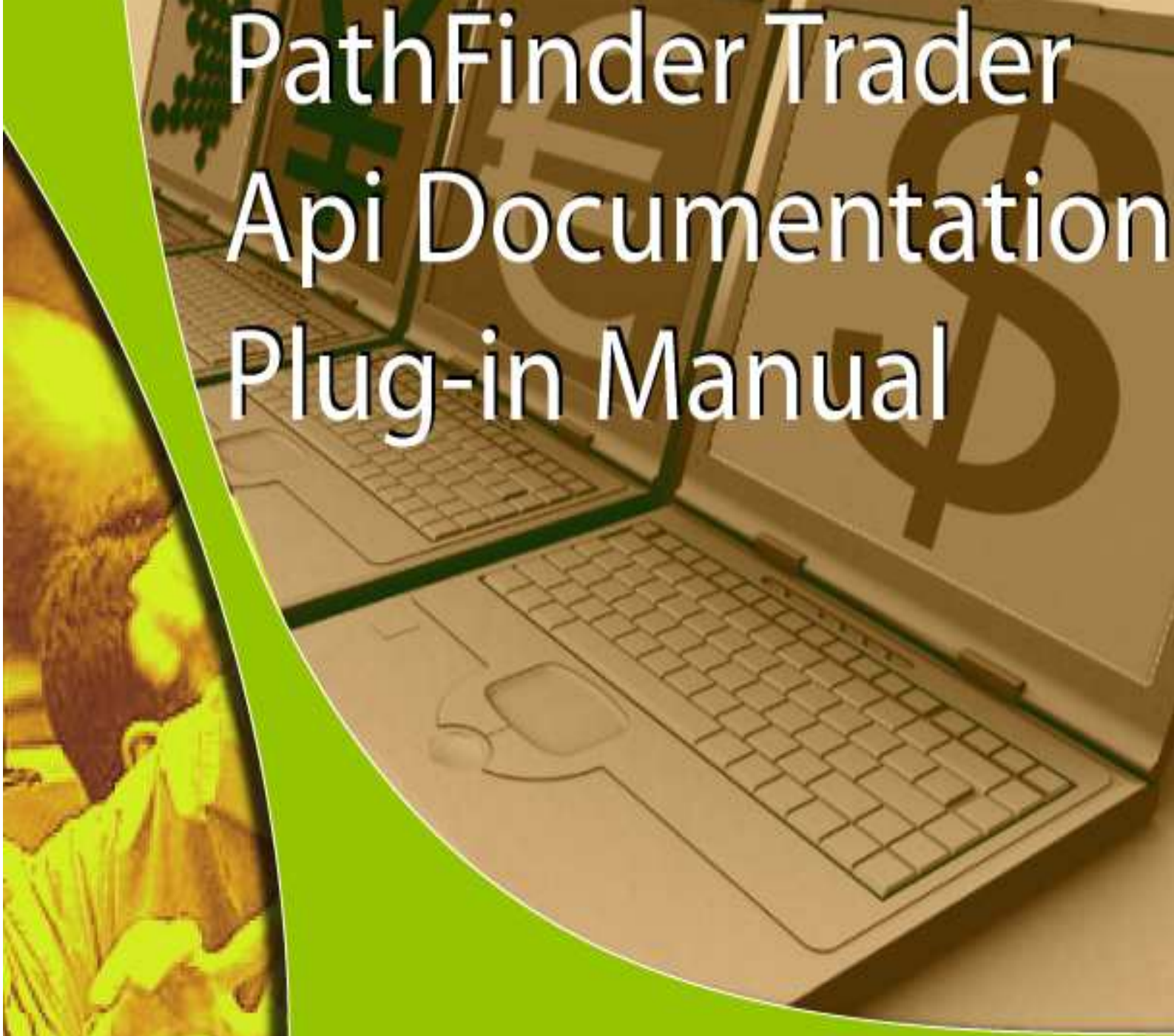
PathFinder Trader Api Documentation Plug-in Manual



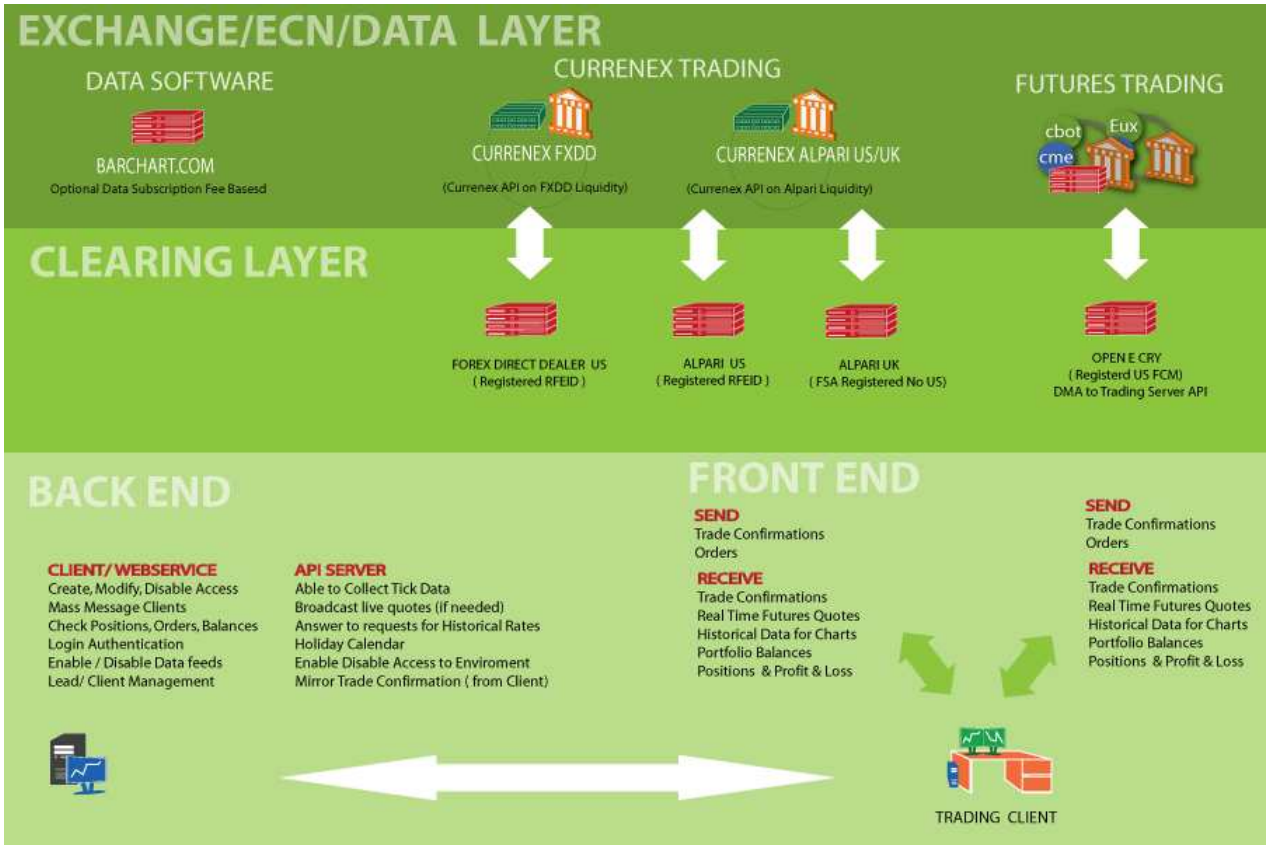
FASTBROKERS

DISCLAIMER:

This document and any files transmitted with it, are confidential and intended solely for the use of the individual or entity to which they are addressed. This document or any portion hereof may not be reprinted, sold or redistributed without the prior written consent of Fast Trading Services LLC. Document is for information purposes only and should not be regarded as an offer to sell or as a solicitation of an offer to buy any financial product. Information cannot be guaranteed to be secure or error-free, therefore, we do not represent that this information is complete or accurate and it should not be relied upon as such.

Risk Disclosure: There is a substantial risk of loss in trading futures and foreign exchange. Please carefully review all risk disclosure documents before opening an account as these financial instruments are not appropriate for all investors.

PathFinder Trader™ Sample Deployment



PathFinder Trader™ API Specifications

INTRODUCTION

For the most demanding trading solutions, ECN2 Technologies offers an Application Protocol Interface (API) which operates through ECN2 Trader to create a direct link with your front end. Our API is a software component intended for the development of applications accessing the services of PathFinder Trader. It is currently available as a set of .NET 3.5 DLL libraries making it simple to access and intuitive to program thanks to its object based characteristics.

The API allows you to take advantage of some of the richest functionalities our designed DMA system has to offer, including: Access to multiple FX ECN and different liquidity landscapes, trading on the most liquid Futures Contracts, access to Contract for Difference, and Data Providers. The API environment will allow your application to interact with the PF infrastructure to perform advanced functions such as: Quote subscription (including Depth of Market), and Data Broadcasting, advanced trading on fully proprietary algorithms, Historical rates queries (including ticks data). Prior to use your application you can run it on our testing environment to finely tune your work.

API Package Install?

The ECN2API library is provided as a set of a zip package containing all required dll files and Resources to make the client API run.
To Run your project, just reproduce these folders structure on build output, setting the api files as application content. To better understand you can download an Example Application Project from ECN2 Support website and take a look at Example Application's code.

What can you do with ECN2API?

The ECN2API library provides a complete set of methods and interfaces to access direct market data and orders front-end of each counterparty and liquidity supported on Pathfinder Trader™.

ECN2API is a multi-feed structure and the architecture of the interface is based on the concept of "multi-data-providing". The implementation of this kind of structure makes ECN2API really flexible yet powerful, giving the final user a way to access and trade on multiple marketplaces and financial instruments through single software.

Currently supported data feeds are:

- **Integral, (Forex ECN)**
- **Currenex, (Forex ECN)**
- **OpenECry, (Futures DMA)**
- **LMax, (Forex ECN, CFD)**
- **Nadex, (Binaries)**
- **PatSystems, (Futures DMA)**
- **Barchart (Data)**
- **ECN2 (Proprietary Liquidity Aggregation for Forex ECNs)**

Market Data is accessed by ECN2API through a Fix Protocol Connection transmission using a 128 bit SSL tunnel where supported. The simple and modular structure of ECN2API allows and makes simple the implementation of a new data feed.

Within ECN2API data feeds are organized in lists and dictionaries. Any data feed respects a unique programming interface criteria. Consequently developers will be able to access data and place orders using a single standardized series of methods, being able with few lines to accomplish and adapt to different instrument and marketplaces.

ECN2API is fully equipped to satisfy the needs of the most demanding Algo-trader. A fully API-integrated non procedural scripting language comes with our library and classes to build automated trading strategies and custom executing indicators and Plug-ins. Furthermore, A DDE (Dynamic Data Exchange) server is also part of the library.

- With our ECN2API Data class you can do:
- Access the list of connected feeds
- Ask to connect or disconnect one or a list of feeds
- Receive alerts on connections change events
- Receive alerts on provider messages updates.
- Place any order type supported by the selected provider
- Receive market Price Real Time quotes
- Receive depth of market both aggregated or extended (if provided)
- Query for data organized in full customized timeframes
- Request Ticks and bars history
- Access FB Server to receive market replay data
- Retrieve Margin account information, including Balance history
- Manage orders, send or cancel any kind of order supported by the counterpart
- Receive Order Status updates
- Request orders histories, active orders statuses and portfolio updates

- Access detailed info on any symbol or contract
- Access user account info and activation statuses
- Activate a DDE server to share Realtime data with excel or any other DDE client

1.1 Purpose

The purpose of this documents it to introduce allowed third party programmers to implement financial applications and fully integrated external plug-ins using ECN2API libraries. ECN2API libraries offering a complete set of classes and methods to manage connectivity, price streaming, orders execution and back office information for each supported counterparty. In addition to this document, take a look to latest ECN2 API Documentation (CHM File) to get details on classes and interfaces structure.

1.2 Compatibility

ECN2API library is designed for .Net programming. It is compatible with any .net language and need a .net version greater or equal to 3.5 sp1.

1.3 Interfaces Structure

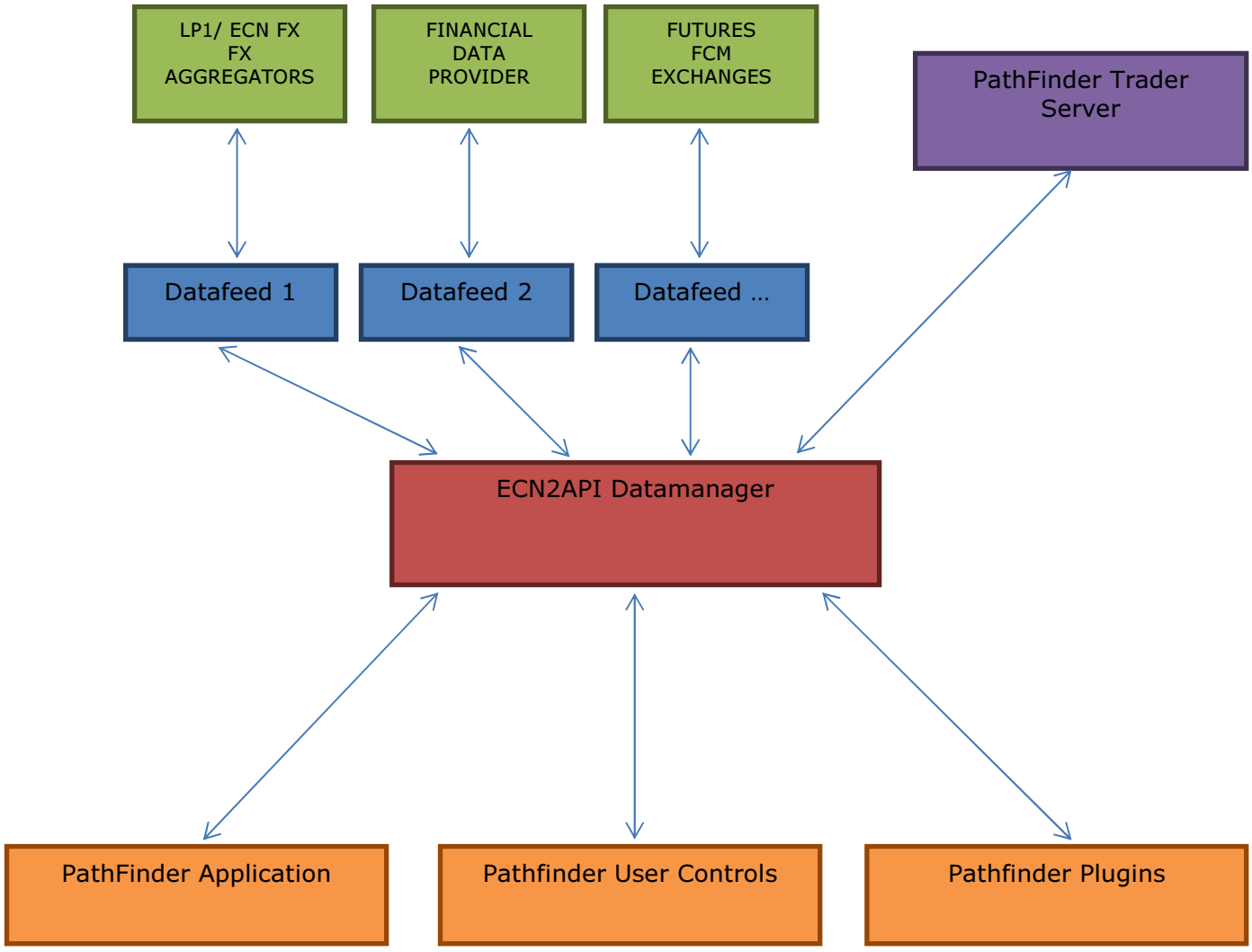
The main class of ECN2API is the Datamanager Class, which provides a container able to connect, send and receive commands & messages from any configured counterparty.

An application based on ECN2API must implement the **IPathFinderApplication** Interface to create a Datamanager instance. To subscribe and receive prices data streaming a class must implement the **IDataSubscriber** Interface. To send orders and receive orders status update a class must implement the **IOrderSubscriber** interface.

Refer to latest API Documentation File for a detailed Classe's Reference.

1.4 PathFinder Architecture

This schema illustrates how the data flow is driven through PathFinder components.



1.5 The Datamanager Class

The entire API work around a Datamanager instance. At application startup the Datamanager will do two things, the first is to retrieve our user settings from server; the second thing is to initialize and connect all enabled datafeed instances.

To create a Datamanager instance the user need an ApplicationID and Password needed to authenticate the API application to our servers. A Datamanager instance can also be retrieved using remoting technology or implementing your software as a plugin and not as an application. This concept will be better explained later on this document.

1.6 The FBDatafeed Interface

The Datamanager class provides all needed methods and properties to do everything on a connected counterparty, anyway, pathfinder give the possibility to access the FBDatafeed Object that own the connection directly, on a lower level.

Get the datafeed instance invoking one of GetDatafeedFrom… methods.

The FBDatafeed Interface nest some important type the most important nested types are:

- FBDatafeed.BarData (a structure containing info about a chart bar)
- FBDatafeed.CLPortfolio (Provide a set of methods and properties to manage portfolios)
- FBDatafeed.Order (a class containing info about an order. Used also as order draft.)

INTERACT WITH ECN2API CLASSES

2.1 Obtaining a Datamanager Instance

2.1.1 Create your own Datamanager:

To create a Datamanager instance the user have to give reference to:

```
IpathFinderApplication
```

class to act as parent. Please note that a valid license key is needed. The Datamanager instance will not operate if the parent license key is not valid.

2.1.2 Get Datamanager from Remoting:

If you are connecting to PathFinder using dot-net Remoting, in order to obtain the current running Datamanager instance, you need to call the function:

```
ECN2API.CommunicationServiceClass.ConnectPathFinder (username, password,
licenseKey, clientAssemblyFilePath)
```

Which will return an active Datamanager usable to subscribe Real Time quotes or send orders etc.

Using Remoting, a specific datafeeds reference are not allowed, all commands have to be sent through the Datamanager.

Note that use of events trough Remoting require inheritance from MarshalByRefObject.
To avoid this limitation, in case that you class cannot inherit from MarshalByRefObject, you can use an intermediate object and pass it to Datamanager has subscriber in your requests.

The following example illustrate how to declare an intermediate object class that forward events coming from ECN2 Trader:

```
Public Class EventsForwarder
    Inherits MarshalByRefObject
    Implements ECN2API.IDataSubscriber, ECN2API.IOrdersSubscriber

    Private m_ControlID As Integer = Rnd() * 1000000

    Public Event ConnectionsChange(ByVal feed As ECN2API.FBDatafeed, ByVal unexpected As Boolean)
    Public Event BarUpdate(ByVal Symbol As String, ByVal BarType As ECN2API.FBDatafeed.Periodicity,
ByVal Bar As ECN2API.FBDatafeed.BarData)
    Public Event PriceUpdate(ByVal NewQuote As ECN2API.FBDatafeed.PriceUpdate)
    Public Event VolumeUpdate(ByVal symbol As String, ByVal datafeed As
ECN2API.FBDatafeed.DatafeedType, ByVal price As Double, ByVal volume As Double)
    Public Event OrderUpdate(ByRef _Order As ECN2API.FBDatafeed.Order)

    Public Sub ConnectionsChanged(ByVal feed As ECN2API.FBDatafeed, ByVal unexpected As Boolean)
        RaiseEvent ConnectionsChange(feed, unexpected)
    End Sub

    Public Sub onBarUpdate(ByVal Symbol As String, ByVal BarType As ECN2API.FBDatafeed.Periodicity,
ByVal Bar As ECN2API.FBDatafeed.BarData) Implements ECN2API.IDataSubscriber.BarUpdate
        RaiseEvent BarUpdate(Symbol, BarType, Bar)
    End Sub

    Public ReadOnly Property ControlID() As Integer Implements ECN2API.IDataSubscriber.ControlID,
ECN2API.IOrdersSubscriber.ControlID
        Get
            Return m_ControlID
        End Get
    End Property

    Public ReadOnly Property MarketDepthNeeded() As ECN2API.FBDatafeed.PriceSubscriptionType
Implements ECN2API.IDataSubscriber.MarketDepthNeeded
        Get
            Return ECN2API.FBDatafeed.PriceSubscriptionType.BestPrice
        End Get
    End Property

    Public Sub onPriceUpdate(ByVal NewQuote As ECN2API.FBDatafeed.PriceUpdate) Implements
ECN2API.IDataSubscriber.PriceUpdate
        RaiseEvent PriceUpdate(NewQuote)
    End Sub

    Public Sub onVolumeUpdate(ByVal symbol As String, ByVal datafeed As
ECN2API.FBDatafeed.DatafeedType, ByVal price As Double, ByVal volume As Double) Implements
ECN2API.IDataSubscriber.VolumeUpdate
        RaiseEvent VolumeUpdate(symbol, datafeed, price, volume)
    End Sub

    Public Sub onOrderUpdate(ByRef _Order As ECN2API.FBDatafeed.Order) Implements
ECN2API.IOrdersSubscriber.OrderUpdate
        RaiseEvent OrderUpdate(_Order)
    End Sub

End Class
```

Note: *Remoting can be enabled from the Software Main Options. Refer Microsoft documentation to get more details about .net Remoting at http://msdn.microsoft.com/en-en/library/kwdt6w2k%28v=vs.80%29.aspx*

2.1.3 Get Datamanager Class within a custom Plug-in:

If you are programming a custom Plug-in to integrate with Pathfinder Trader™, you will notice that you don't need any specific instance to get the Datamanager class: in fact an

instance of **ECN2API.Datamanager** will be already in the **PFPlugin.Datamanager** property of your Plug-in control.

2.2 Building a Plug-in, the easy way to start.

PathFinder Trader easily integrates with complex third parties Plug-ins. Programmers which want to develop their own applications to execute commands through Pathfinder Trader, simply need a valid license of the software installed on the same machine, and an IDE/Compiler to create a .Net library.

2.2.1 Create a class library project

- Add Reference to **ECN2API.dll** into PathFinder Installation Folder
- Within your project, create a main class and make such class derive from **PFPlugin Class** (this will be the user interface of your plugin);
- If your Plug-in needs to subscribe streaming real time Quotes make sure to implement the **IDataSubscriber** interface within your class;
- If your Plug-in needs to execute orders make sure to implement the **IOrderSubscriber** Interface within your class;
- Assign a unique guid to the PluginID property and ask for his activation to our support service;
- Implement your own plugin code;

NOTE: An advanced plugin implementation tutorial "ExecutorPlugin" can be downloaded as Visual Studio 2008 Solution.

2.2.2 Plug-in Sample (VB.net)

The following example shows how to create a Plug-in that subscribe **EUR/USD** rates requested to a virtual datafeed named "**X"**. The final purpose of this Plug-in is to display and show any price update by creating a new line in a textbox:

```
Public Class MyPlugin
    Inherits PFPlugin
    Implements ECN2API.IDataSubscriber, ECN2API.IOrdersSubscriber

    Dim m_Datamanager As ECN2API.DataManager
    Dim txtBox As new TextBox()

    Public Sub New()
      txtBox = new TextBox()
      txtBox.Multiline = True
      txtBox.Dok = Fill
      Me.Controls.Add(txtBox)

      'Set the assigned plugin ID
      MyBase.PluginID = New Guid("00000000-0000-0000-0000-000000000000")
    End Sub

    Public Overrides Sub SetDatamanager(ByRef dataman As ECN2API.DataManager)
        MyBase.SetDatamanager(dataman)

      Dim feed As FBDatafeed = Me.Datamanager.GetDatafeedFromString("FeedName")
      Me.Datamanager.SubscribeForPrices(Me, "EUR/USD", feed , False)

    End Sub



  'Invoked on new bars if the control is registered using SubScribeForBars
    Public Sub BarUpdate(ByVal Symbol As String, ByVal BarType As
ECN2API.FBDatafeed.Periodicity, ByVal Bar As ECN2API.FBDatafeed.BarData)
Implements ECN2API.IDataSubscriber.BarUpdate

    End Sub

    'Used from API to access the control
    Public ReadOnly Property ControlID() As Integer Implements
ECN2API.IDataSubscriber.ControlID, ECN2API.IOrdersSubscriber.ControlID
        Get
            Return Me.Handle
        End Get
    End Property

    'The market depth needed by this control
    Public ReadOnly Property MarketDepthNeeded() As
ECN2API.FBDatafeed.PriceSubscriptionType Implements
ECN2API.IDataSubscriber.MarketDepthNeeded
        Get
            Return ECN2API.FBDatafeed.PriceSubscriptionType.BestPrice
        End Get
    End Property

    'Invoked on new prices if the control is registered using SubScribeForPrices
    Public Sub PriceUpdate(ByVal NewQuote As ECN2API.FBDatafeed.PriceUpdate)
Implements ECN2API.IDataSubscriber.PriceUpdate

       txtBox.Text &= VbCrLf & "New Eur/Usd Price received: " & NewQuote.Price

    End Sub

    'Invoked on volume histogram updates if the control is registered using
```

```
SubScribeVolumes
    Public Sub VolumeUpdate(ByVal symbol As String, ByVal datafeed As
ECN2API.FBDatafeed.DatafeedType, ByVal price As Double, ByVal volume As Long)
Implements ECN2API.IDataSubscriber.VolumeUpdate

    End Sub

    'Invoked on order updates to any IOrdersSubscriber control
    Public Sub OrderUpdate(ByRef _Order As ECN2API.FBDatafeed.Order) Implements
ECN2API.IOrdersSubscriber.OrderUpdate

    End Sub
End Class
```

2.3 Interacting with the Data Manager. Few Samples.

The Pricing Stream

Prices update mechanism is handled by the **Datamanager.** Subscription to a quote stream is achieved by invoking the Methods **SubscribeForPrices** *(IDataSubscriber subscriber Instance, String Symbol, DatafeedType Datafeed, Bool WaitPrice)* after having successfully initiated Datamanager instance.

The **subscriberinstance**, is the instance belonging to the control class instance that will receive updates. Immediately after subscription, any price update will be notified by the interface IDataSubscriber which will invoke the method **PriceUpdate** (*FBDatafeed.PriceUpdate* quote) method on the subscriber instance.

The Quote Class (FBDatafeed.PriceUpdate data type)

The object returned by the Datamanager to the prices subscriber, is a nested type of the **FBDatafeed** Interface called **PriceUpdate**.

Below is a short sample structure of this class:

```
Public TradeDateTime As Date
Public Symbol As String
Public Price As Double
Public LastVolume As Long
Public TotalVolume As Long
Public Bid As New List(Of Double)
Public BidSize As New List(Of Long)
Public Ask As New List(Of Double)
Public AskSize As New List(Of Long)
Public DataFeedType As FBDatafeed.DatafeedType
```

It seems obvious that the class above contains basic information related to a financial instrument. This choice to keep the class light and usable to keep this class light and usable. Thus more detailed information (which are not strictly related to prices changes) is stored on classes called **SymbolsDetails.** To get an Instrument detail, it is necessary to invoke the **GetSymbolDetails** (*String symbol, DatafeedType datafeed*). The output will have the following structure:

```
        Public Product As String
        Public Provider As FBDatafeed
        Public Exchange As String
        Public Group As String
        Public ClearingHouse As String
        Public Symbol As String
        Public Description As String
        Public ReadOnly Property TickValue() As Double
        Public ReadOnly Property ContractSize() As Double
        Public ReadOnly Property InitialMargin() As Double
        Public TradingHours As String
        Public ReadOnly Property Swap() As Double
        Public Commissions As Double
        Public BaseCurrency As String
        Public Expiration As String
        Public DaySettle As Double
        Public DayOpen As Double
        Public DayHigh As Double
        Public DayLow As Double
```

2.3.1 The Quote Streaming in fact, a class skeleton

In this example we will focus on the part of code which represents the basic of a quote subscriber class:

```
Public Class  MyQuoteReceiverClass
    Implements ECN2API.IDataSubscriber


           'The datamanager instance to invoke, set this var to an active instance
somewhere in your class
           'Refer to Chapter 1.7 “How to Obtain a datamanager instance”
    Dim m_Datamanager As ECN2API.DataManager

  'Invoked on new bars if the control is registered using SubScribeForBars
    Public Sub BarUpdate(ByVal Symbol As String, ByVal BarType As
ECN2API.FBDatafeed.Periodicity, ByVal Bar As ECN2API.FBDatafeed.BarData) Implements
ECN2API.IDataSubscriber.BarUpdate

       'Invoked on bars update only if the client subscribe bars using SubscribeForBars
mothod

    End Sub

    'Used from API to access the control
    Public ReadOnly Property ControlID() As Integer Implements
ECN2API.IDataSubscriber.ControlID        Get
            Return 12398592874 ‘A unique identifier
        End Get
    End Property

    'The market depth needed by this control
    Public ReadOnly Property MarketDepthNeeded() As
ECN2API.FBDatafeed.PriceSubscriptionType Implements
ECN2API.IDataSubscriber.MarketDepthNeeded
```

```
        Get
            Return ECN2API.FBDatafeed.PriceSubscriptionType.BestPrice
          'Return ECN2API.FBDatafeed.PriceSubscriptionType.MarketDepth

          'The commented line show how to subscribe a full book quote straming to
receive prices L2
        End Get
    End Property



    'Invoked on new prices if the control is registered using SubScribeForPrices
    Public Sub PriceUpdate(ByVal NewQuote As ECN2API.FBDatafeed.PriceUpdate) Implements
ECN2API.IDataSubscriber.PriceUpdate

       'Invoked on price update also for book prices

    End Sub

    'Invoked on volume histogram updates if the control is registered using
SubScribeVolumes
    Public Sub VolumeUpdate(ByVal symbol As String, ByVal datafeed As
ECN2API.FBDatafeed.DatafeedType, ByVal price As Double, ByVal volume As Long) Implements
ECN2API.IDataSubscriber.VolumeUpdate

       'Invoked on current session volume histogram update

    End Sub

End Class
```

2.3.2 The Order Class (FBDatafeed.Order data type)

The object given by the Datamanager to the orders subscriber is a nested type of FBDatafeed Interface called Order, the structure of this class is the following:

```
Implements IEqualityComparer(Of Order), IComparable(Of DateTime), IComparable(Of
Integer), ICloneable

        Public Enum OrderStatus
            Unspecified = 0 'Order Update arrive but no status has specified
            Sending = 1     'Order is being transmitted
            Received = 2    'Order has been received
            Working = 3    'Order is ready to be filled
            CancelSending = 4 'Cancel request is being sent
            Canceled = 5 ' Cancel request has been received
            Filled = 6     'Order has been filled
            PartialFilled = 7 'Your order has been partially filled
            Rejected = 8   'Order was rejected
            Parked = 9    'Order is waiting his parent fill
            Unknown = 10    'Indicates a problem - call your broker!
        End Enum

        Public Enum OrderSide
            [Long] = 1
            [Short] = 2
            Flat = 3
```

```
            Unknown = 4
        End Enum
        Public Enum OrderType
            Unknown = 0
            Limit = 1 'Limit
            Market = 2
            StopLimit = 3
            [Stop] = 4
            Iceberg = 5
            TrailingStopLimit = 6
            TrailingStop = 7
        End Enum
        Public Enum Expiration
            Day = 0
            GoodTillCanceled = 1
            ImmediateOrCancel = 2
            FillOrKill = 3
            GoodTillDate = 4
            Unknown = 5
            AllOrNothing = 6
        End Enum
        Public Class ExecutionItem
            Public ReadOnly Property TransactTime() As DateTime
            Public ReadOnly Property ExecutionStatus() As Order.OrderStatus
            Public ReadOnly Property LeavesQty() As Double
            Public ReadOnly Property FilledQty() As Double
            Public ReadOnly Property ExecPrice() As Double
            Public ReadOnly Property ExecutionID() As String
        End Class

        Public ReadOnly Property CheckSum() As Long
        Public Quantity As Double = 0
        Public LeavesQty As Double = 0
        Public CumQty As Double = 0
        Public MaxShow As Double = 0
        Public MinQty As Double = 0
        Public OrderID As Long = -1
        Public DFOrderID As String = ""
        Public ExecutionID As String = ""
        Public Provider As DatafeedType = Nothing
        Public Counterpart As String = ""
        Private ExecsHistory As New List(Of ExecutionItem)
        Public Property ExecutionsHistory() As List(Of ExecutionItem)
        Public Side As Order.OrderSide = OrderSide.Flat
        Public Exchange As String = ""
        Public Symbol As String = ""
        Public OrdType As OrderType = OrderType.Unknown
        Public ExecPrice As Double = 0
        Public StopPrice As Double = 0
        Public StopLoss As Double = 0
        Public TakeProfit As Double = 0
        Public TrailingStop As Double = 0
        Public Expires As Expiration = Expiration.Unknown
        Public ExpireTime As Date
        Public Slippage As Integer = 0
        Public EntryTime As Date
        Public ExecTime As Date
        Public LocalExecTime As Date
        Public Swap As Double = 0

        Public ReadOnly Property AvgPrice() As Double
```

```
            Get
                Dim GeomSum As Double
                Dim totQuantities As Double = 0
                Dim retValue As Double

                For i As Integer = 0 To ExecutionsHistory.Count - 1

                    Dim exec As ExecutionItem = ExecutionsHistory(i)

                    If exec.ExecutionStatus = OrderStatus.Filled OrElse
exec.ExecutionStatus = OrderStatus.PartialFilled Then
                        GeomSum += exec.ExecPrice * exec.FilledQty
                        If exec.ExecPrice <> 0 Then totQuantities +=
exec.FilledQty
                    End If

                Next

                If GeomSum = 0 OrElse totQuantities = 0 Then
                    retValue = ExecPrice
                Else
                    retValue = (GeomSum / totQuantities)
                End If

                Return retValue
            End Get

        End Property
        Public Property Status() As Order.OrderStatus
            Get
                Return OrdStatus
            End Get
            Set(ByVal value As Order.OrderStatus)
                OrdStatus = value
            End Set
        End Property
        Private OrdStatus As Order.OrderStatus
        Public SymbolCurrency As String
        Public Commissions As Double
        Public Details As String
        Public Comments As String

        Public Function GetInitialMargin(ByRef ParentDataManager As FBDatafeed)
As Double
        Public Function GetOpenPL(ByRef ParentDataManager As FBDatafeed) As
Double
        Public Function GetSwap(ByRef ParentDataManager As FBDatafeed) As Double
        Public Function GetContractValue(ByRef ParentDataManager As FBDatafeed)
As Double
        Public Function GetActiveQty(ByVal posEntry As FBDatafeed.PositionEntry)
As Double
        Public Function GetFilledQty() As Double
        Public Shared Function CreateOrderID() As Long
        Public Function EqualsTo(ByVal x As Order, ByVal y As Order) As Boolean
        Public Function GetHashCode1(ByVal obj As Order) As
        Public Function CompareTo(ByVal other As Date) As Integer Implements
System.IComparable(Of Date).CompareTo
        Public Function CompareTo(ByVal other As Integer) As Integer Implements
System.IComparable(Of Integer).CompareTo
        Public Shared Function OrdExpireFromString(ByVal expire As String) As
Expiration
```

```
        Public Shared Function ShortExpire(ByVal expire As Expiration) As String
        Public Shared Function OrdTypeFromString(ByVal ordType As String) As
OrderType
        Public Shared Function OrdSideFromString(ByVal ordSide As String) As
OrderSide
        Public Shared Function OrdStatusFromString(ByVal ordStatus As String) As
OrderStatus
        Public Function Clone() As Object Implements System.ICloneable.Clone
        Public Shared Function Serialize(_Order As FBDatafeed.Order ) As String
        Public Shared Function DeSerialize(ByVal orderString As String) As Order
```

To send orders it is necessary to instantiate a copy of this class, set all needed parameters, and send it through the Datamanager using the method **SendNewOrder**(ByRef subscriber As IOrdersSubscriber, ByVal _order As FBDatafeed.Order). The order will be processed; any updates related to the status will be notified by an updated copy of the order instance on the method **OrderUpdate** of IOrdersSubscriber interface.

2.4 Send a market order, a simple example

The following example illustrates how to send a market order to a selected Counterparty.

```
Public Class MyOrderSubscriberClass
    Implements ECN2API.IOrdersSubscriber

    'The datamanager instance to set
    Dim m_Datamanager As ECN2API.DataManager

    'Used from API to access the control
    Public ReadOnly Property ControlID() As Integer Implements
ECN2API.IOrdersSubscriber.ControlID
        Get
            Return 1234567235 'My Custon unique id
        End Get
    End Property

    Private Sub SendMarketOrder()
        'A simple example on how to send a Buy Market Order
        Dim MyOrder As New FBDatafeed.Order

        MyOrder.Provider = m_Datamanager.GetDatafeedTypeFromString("FeedName")
        MyOrder.OrdType = FBDatafeed.Order.OrderType.Market
        MyOrder.Symbol = "EUR/USD"
        MyOrder.Comments = ""
        MyOrder.Quantity = 100000

        MyOrder.OrderID = FBDatafeed.Order.CreateOrderID()
        MyOrder.Side = FBDatafeed.Order.OrderSide.Long
        MyOrder.Exchange = m_Datamanager.GetSymbolExchange(MyOrder.Symbol,
```

```
MyOrder.Provider)
        MyOrder.Slippage = 0
        MyOrder.MaxShow = 0
        MyOrder.StopPrice = 0
        MyOrder.ExecPrice = 0
        MyOrder.TakeProfit = 0
        MyOrder.TrailingStop = 0
        MyOrder.StopLoss = 0
        MyOrder.Expires = FBDatafeed.Order.Expiration.GoodTillCanceled
        MyOrder.LocalExecTime = Now
        MyOrder.EntryTime = Now.ToUniversalTime
        MyOrder.Status = FBDatafeed.Order.OrderStatus.Sending

        m_Datamanager.SendNewOrder(Me, MyOrder)
    End Sub
    'Invoked on order updates to any IOrdersSubscriber control
    Public Sub OrderUpdate(ByRef _Order As ECN2API.FBDatafeed.Order) Implements
ECN2API.IOrdersSubscriber.OrderUpdate
        'The updated order will be received by this function
        'Check the value of _Order.Status to see how it change from Sending to
Working and from Working to Filled
        'Obviously, if the order will be rejected the returned status will not
be Working but Rejected
    End Sub

      End Class
```

3.0 The ExecutorPlugin Tutorial

The ExecutorPlugin example provide a fully functional plugin usable as framework for your how code. The source code can be used in two different modes:

1- Setting the value of useOnlySkeleton to true before compile; to write your own code inside the "UserCode" function call. In this way the plugin will be only a monitor and start/stop controller for your vb .net code.
2- Setting the value of useOnlySkeleton to false before compile; In this way another tab will appair to write your own code that will be compiled on the fly at strategy start. Using this mode you could change the code of the strategy on the fly without restart the platform.

The example code placed in the "UserCode" Sub, generate a sound alert with a message on the bull cross of two moving averages (14,34). The code to send a market order is also commented inside the same method.