

## Visual Trading Systems, LLC

<http://www.vtssystems.com>

350 Fifth Avenue, Suite 6420

New York, NY 10118, USA

## Title: VTtrader API version 1.0 documentation

### 1 Document Information

---

Version: 1.0.3

Mod Date: 10 May 2006

Contacts: [api@vtssystems.com](mailto:api@vtssystems.com), [techsupport@vtssystems.com](mailto:techsupport@vtssystems.com)

© Copyright 2004 - 2006. Visual Trading Systems, LLC

### 2 Contents

---

1. Document information
2. Contents
3. General information
4. Functions description (functional style)
  1. GetServerList / Login / Logout / Finalize
  2. Server time
  3. Accounts
  4. Instruments
  5. Open Positions
  6. Orders
  7. Report
  8. Message handling
  9. Constants / Errors / Settings
  10. Error messages
  11. Send functions (general description)
  12. Create Order / Create order with Stop/Limit
  13. Change Order
  14. Remove Order
  15. Close Position
  16. Create Hedge
  17. Set Stop / Limit
  18. Change Stop / Limit
  19. Remove Stop / Limit
  20. Events
  21. GetHistory
  22. GetSetting - **DEPRICATED, use `settings()` function.**
  23. LoggedIn
  24. MarketStatus
  25. GetClosedPositions
  26. LoginInProgress
  27. SendAnnouncement
  28. SSL Disabling
  29. History depth limitation
5. Data types
  1. IServerMessage
  2. IAccount
  3. IInstrument
  4. IPosition
  5. IOrder, IConditionalOrder
  6. IServerMessages
  7. IAccounts
  8. IInstruments
  9. IPositions
  10. IOrders
  11. ISettings

6. Objects description (object-oriented style)
7. Appendix A: Implementing Event model in Visual Basic 6, Visual Basic.NET and C#

### 3 General information

---

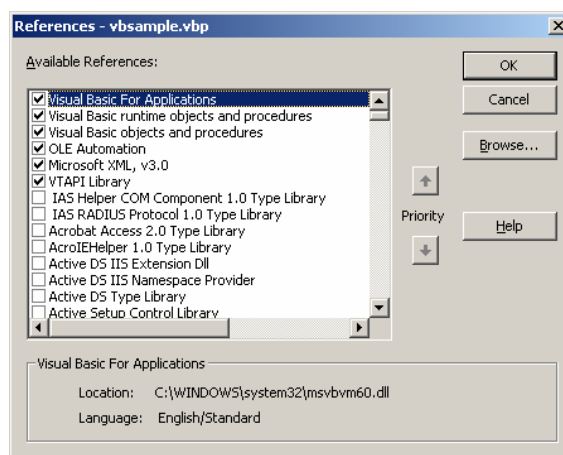
Official FAQ is available at: <http://forum.vtssystems.com/viewtopic.php?t=1953>

VTTrader API (further VTAPI) is a set of COM servers (vtcore.dll and vtapi.dll), configuration file servers.api.ini, SSL DLLs (libeay32.dll, ssleay32.dll) and error messages file – vtapimessages.enu.xml.

VTAPI Samples are located in Samples subfolder. Currently provided: C#, VB.NET, Java, Delphi, VBA, Perl.

Since version 1.0.2 it is now possible to subscribe and receive only selected instrument's rate changes. This can reduce network traffic (on newest VTServer versions). See `IInstrument.set/getSubscribed` and `IInstruments.CommitSubscription` methods. Note, that at least one Instrument should be subscribed.

In order to access VTAPI from VisualBasic, for example, you should check the following checkboxes in the Project\References dialog.



### 4 Functions description (functional style)

---

#### 4.1 GetServerList / Login / Logout / Finalize

❖ `function GetServerList: WideString;`

Function returns XML containing server list as a string. This list is read from servers.ini file. For example:

```
<?xml version="1.0"?>
<servers>
  <server id="0" alias="Demo trading"/>
  <server id="1" alias="Live Trading"/>
  <server id="2" alias="Local Trading"/>
</servers>
```

Attribute "id" of node "server" should be passed to Login function to tell it the server to login to.

❖ `function login(UserName, Password: WideString; ServerID: Integer): Integer;`

Function connects VTAPI to the trading server. Parameters:

`UserName` - User name.

`Password` - Password.

`ServerID` - Server identifier. See function GetServerList.

Returns - error Code (see p.p. 4.9, 4.10).

❖ `function Logout: Integer;`

Function disconnects VTAPI from the trading server.

Returns - error Code (see p.p. 4.9, 4.10).

❖ `procedure finalize;`

Destroys internal objects. Calls logout if logged in. Call it on application (or main form) close.

#### 4.2 Server time

❖ `function GetServerTime: WideString;`

Function returns server time as String. Format: YYYY-MM-DD HH:NN:SS

- ❖ `function GetServerTimeAsDate: TDateTime;`  
Function returns server time as DATE.
- ❖ `function GetServerTimeShift: WideString;`  
Function returns time difference between server and client as String. Format: HH:NN:SS.

### 4.3 Accounts

- ❖ `function GetAccountCount: Integer;`  
Function returns number of accounts.
- ❖ `function GetAccountByID(ID: WideString): IAccount;`  
Function returns account (see p. 5.2) by identifier (see IAccount.ID)
- ❖ `function GetAccountByIndex(Index: Integer): IAccount;`  
Function returns account (see p. 5.2) by index (index should be between 0 and GetAccountCount-1)

### 4.4 Instruments

- ❖ `function GetInstrumentCount: Integer;`  
Function returns number of instruments.
- ❖ `function GetInstrumentByID(ID: WideString): IInstrument;`  
Function returns instrument (see p. 5.3) by identifier (see IInstrument.ID)
- ❖ `function GetInstrumentByIndex(Index: Integer): IInstrument;`  
Function returns instrument (see p. 5.3) by index (index should be between 0 and GetInstrumentCount-1)
- ❖ `function GetInstrumentByCurrency(currency1, currency2: WideString): IInstrument;`  
Function returns instrument (see p. 5.3) by currency pair. E.g. `GetInstrumentByCurrency("EUR", "USD")`.

### 4.5 OpenPositions

- ❖ `function GetOpenPositionCount: Integer;`  
Function returns number open positions.
- ❖ `function GetOpenPositionByID(ID: WideString): IPosition;`  
Function returns open position (see p. 5.4) by identifier (see IPosition.ID)
- ❖ `function GetOpenPositionByIndex(Index: Integer): IPosition;`  
Function returns open position (see p. 5.4) by index (index should be between 0 and GetOpenPositionCount-1)

### 4.6 Orders

- ❖ `function GetOrderCount: Integer;`  
Function returns number of orders.
- ❖ `function GetOrderByID(ID: WideString): IOrder;`  
Function returns order (see p. 5.5) by identifier (see IOrder.ID)
- ❖ `function GetOrderByIndex(Index: Integer): IOrder;`  
Function returns order (see p. 5.5) by index (index should be between 0 and GetOrderCount - 1)

### 4.7 Report

- ❖ `function GetReportUrl(Account: OleVariant): WideString;`  
Function returns URL to get a report from server. Parameter Account is optional.
- ❖ `function ShowReport(Account: OleVariant): Integer;`  
Function launches the default browser with server report page. Parameter Account is optional.

### 4.8 Message handling

- ❖ `function GetMessageCount: Integer;`  
Function returns server messages count.
- ❖ `function GetMessageByIndex(Index: Integer): IServerMessage;`  
Function returns server message by index (index should be between 0 and GetMessageCount-1)

- ❖ `function GetMessageByID(ID: Integer): IServerMessage;`  
Function returns server message by identifier. (See `IServerMessage.ID`)
- ❖ `function ShowMessageWindow(OwnerHandle: Integer; StayOnTop: WordBool): Integer;`  
Function shows server messages window. Parameters:  
`OwnerHandle` – Handle of main window of custom application.  
`StayOnTop` - Determines whenever window should be on top.  
Returns - error Code (see p.p. 4.9, 4.10).
- ❖ `function HideMessageWindow: Integer;`  
Function hides server messages window. Returns - error Code (see p.p. 4.9, 4.10).
- ❖ `function SetMsgRefresh(EventSink: IMsgItemsListEvents): Integer;`  
Function sets handler for new messages. Handler class must implement method `OnAddMessage()` of `VTcore.IMsgItemsListEvents` interface. Not supported by VisualBasic 6. Returns - error Code (see p.p. 4.9, 4.10).
- ❖ `function UnSetMsgRefresh: Integer;`  
Function restores handler for new messages to initial state. Returns - error Code (see p.p. 4.9, 4.10).

#### 4.9 Constants / Errors

- ❖ `function constants: ConstRecord;`  
Function returns constant-record. Record field values could be used to pass parameters to several functions and read properties of several objects.  
`constRecord: (dur_Day:True; dur_GTC:False; bs_Buy:False; bs_Sell:True; ord_Initial:True; ord_entry:False; ot_Stop:'S'; ot_Limit:'L'; ot_EntryStop:'ES'; ot_EntryLimit:'EL'; Ask:True, Bid:False);`  
E.g. `If API.Oder("1234563").BuySell=API.Constants.Buy then...`
- ❖ `function settings(): ISettings`  
Use this function to obtain `LimitRateFromMarket`, `StopRateFromMarket`, `HedgingEnabled` settings values. See p. 5.11 for more details.  
E.g. `if API.settings.HedgingEnabled(account) then ...`

The following values are referenced to the error codes.

ERROR enumeration:

```

NO_ERROR = 0,
UNKNOWN_ERROR = 1,
HTTP_ERROR = 2,
PARSE_ERROR = 3,
TIMEOUT_ERROR = 4,
DICTIONARY_ERROR = 5,
ALREADY_LOGGED_ERROR = 6,
NOT_LOGGED_ERROR = 7,
PROTOCOL_ERROR = 8,
INTERFACE_ALREADY_SET_ERROR = 9,
INTERFACE_NOT_SET_ERROR = 10,
XML_CONTENT_ERROR = 11,
WRONG_PASSWORD = 12,
LOGIN_IN_PROGRESS = 13,
INVALID_REFRESH_RATE = 14,
INVALID_CLOSE_RATE = 15,
INVALID_HISTORY_DEPTH = 16,
INVALID_INSTRUMENT = 17,
INVALID_SERVER_INDEX = 18,
INVALID_USERID = 19,
INVALID_AMOUNT = 20,
GETOPENCANDLES_ERROR = 21,
INVALID_SETTING = 22,
INVALID_INTERVAL = 23,
INVALID_STOPORDER_RATE = 24, - reserved
INVALID_LIMITORDER_RATE = 25, - reserved
INVALID_STOPORDER_RANGE = 26,
NOT_IMPLEMENTED = 27,
INVALID_ORDER_RATE = 28,
COMMAND_UNSUPPORTED = 29,
NO_INSTRUMENTS_SUBSCRIBED = 30,
INVALID_ACCOUNT = 31,
INVALID_POSITION = 32,

```

```
INVALID_ORDER = 33,  
HEDGING_DISABLED = 34
```

#### MessageLevel enumeration:

```
ML_ERROR = 0,  
ML_EXCEPTION = 1,  
ML_PARSER_ERROR = 2,  
ML_MESSAGE = 5,  
ML_USER_REQUEST = 6,  
ML_CONFIRM = 7
```

#### MessageType enumeration:

```
MT_NEW_POSITION = 0,  
MT_CLOSE_POSITION = 1,  
MT_CREATE_STOP = 2,  
MT_CREATE_LIMIT = 3,  
MT_CREATE_ORDER = 4,  
MT_UPDATE_STOP = 5,  
MT_UPDATE_LIMIT = 6,  
MT_UPDATE_ORDER = 7,  
MT_REMOVE_STOP = 8,  
MT_REMOVE_LIMIT = 9,  
MT_REMOVE_ORDER = 10,  
MT_ERROR = 11,  
MT_CREATE_INITIAL_ORDER = 12,  
MT_REMOVE_INITIAL_ORDER = 13,  
MT_UNKNOWN_TYPE = 14
```

#### AccountMessageType enumeration:

```
A_ACCOUNT_UPDATE = 0,  
A_MARGIN_CALL = 1,  
A_EQUITY_CALL = 2,  
A_SHORT_MARGIN_CALL = 3,  
A_MARGIN_CALL_LEVEL1 = 4,  
A_MARGIN_CALL_LEVEL2 = 5,  
A_MARGIN_CALL_LEVEL3 = 6,  
A_NEW_ACCOUNT = 7,  
A_ACCOUNT_RATE_UPDATE = 8
```

#### TimeInterval enumeration:

```
T_TICKS = 0,  
T_1_MIN = 1,  
T_5_MIN = 5,  
T_10_MIN = 10,  
T_15_MIN = 15,  
T_30_MIN = 30,  
T_1_HOUR = 60,  
T_2_HOURS = 120,  
T_4_HOURS = 240,  
T_1_DAY = 24,  
T_1_WEEK = 7,  
T_1_MONTH = 31
```

#### Settings enumeration:

```
LIMIT_RATE_FROM_MARKET = 0,  
STOP_RATE_FROM_MARKET = 1
```

**Settings enumeration: - DEPRICATED, use `settings()` function.**

```
LIMIT_RATE_FROM_MARKET = 0,  
STOP_RATE_FROM_MARKET = 1
```

#### MarketStatus enumeration:

```
MS_OPEN = 0,  
MS_CLOSE = 1,  
MS_UNKNOWN = 2
```

These values are automatically imported from COM server libraries – you can use their names instead of values to make your source code more understandable.

All functions that return an error code return one of these values. For example: to determine success of Logout function use the following code:

```
Dim err As Integer
err = API.Logout
If err = NO_ERROR Then
    MsgBox "Logged out OK."
Else
    MsgBox "Logout error: " + API.GetErrorMessage(err) + "code: " + str(err)
End If
```

#### 4.10 Error messages

The following functions are used to get text for error code:

- ❖ `function SetErrorMessageLanguage(lang: WideString; silent: OleVariant): Integer;`  
Function set text language. Parameters:  
`Lang` – name of language. Only "ENU" is supported in current version.  
`silent` – should the function suppress error messages (True/False). Parameter is optional – True by default.
- ❖ `function GetErrorMessage(_id: Integer): WideString;`  
Function returns text for error code. Parameter `id` – is a error code. See example for p. 4.9.

#### 4.11 Send-functions (general description)

All send-functions return a command identifier. When VTAPI has a response from the server, a message is placed in `ServerMessage` object, and the command identifier is assigned to the "Sender" property of `ServerMessage` object.

As a parameter of `OleVariant` type you can pass identifier (string), index (numeric) or object. But for Instrument String value can represent currency, like: "EUR/USD" or "EURusd" or even "eUR any text UsD" value – first and last 3 symbols are used - VTAPI tries to convert it to identifier (by converting to integer) and than tries to find Instrument by currency.

For example:

```
API.SendCreateOrder("1233532","4"... - by identifiers,
API.SendCreateOrder("1342344","eur/usd"... - by identifier and currency,
API.SendCreateOrder(0,1... - by indexes – 0th account, 1st instrument,
API.SendCreateOrder(GetAccountByID("1233532"))... – by object.
```

#### 4.12 CreateOrder / CreateOrderWithSL

- ❖ `function SendCreateOrder (Account, Instrument: OleVariant; Duration, TraderRange: Integer; SellBuy: WordBool; Rate, Amount: Double; Hedge, Initial: WordBool): Integer;`  
Function sends order creation request. Parameters:  
`Account` – Account ID, Index or object  
`Instrument` –Instrument ID, Index, or object  
`Duration` – execution period (see constants `dur_GTC/dur_Day` p. 4.9)  
`TraderRange` –  
`SellBuy` – Sell / Buy (see constants `bs_sell/bs_buy` p. 4.9)  
`Rate` – Rate of order. Can be passed value of 0 for Initial orders – current rate will be used automatically.  
`Amount` – Lot count  
`Hedge` – create hedge (True/False)  
`Initial` – execution (see constants `ord_initial/ord_entry` n. 4.9)  
Returns command identifier.
- ❖ `function CreateOrderWithSL(Account, Instrument: OleVariant; Duration, TraderRange: Integer; SellBuy: WordBool; Rate, Amount: Double; Hedge, Initial: WordBool; LimitRate, StopRate: Double; StopRange: Integer): Integer;` - Same as `SendCreateOrder` function, but allows to create order with Stop or Limit (or both). Pass `LimitRate` = value and `StopRate` = 0 to create with just Limit. Currently only NON Initial orders are allowed.

#### 4.13 ChangeOrder

- ❖ `function SendChangeOrder(Order: OleVariant; NewRate: Double): Integer;`  
Function sends order change request. Parameters:  
`Order` – Order ID, Index or object  
`NewRate` – new rate.  
Returns command identifier.

#### 4.14 RemoveOrder

❖ `function SendRemoveOrder(Order: OLEVariant): Integer;`  
Function sends order remove request. Parameters:  
`Order` – Order ID, Index or object  
Returns command identifier.

#### 4.15 ClosePosition

❖ `function SendClosePosition(OpenPosition: OLEVariant; Amount, Rate: Double; TraderRange: Integer; Hedge: WordBool): Integer;`  
Function sends close position request. Parameters:  
`OpenPosition` – open position ID, Index or object  
`Amount` – lot count  
`Rate` – Rate to close. Must be equal to corresponding position `CloseRate`. But value of 0 can be passed – `CloseRate` will be used automatically.  
`TraderRange` –  
`Hedge` – Close hedge True/False  
Returns command identifier.

#### 4.16 Create hedge

❖ `function SendCreateHedge(Account, OpenPosition: OLEVariant; Amount: Double): Integer;`  
Function sends hedge creation request. Parameters:  
`Account` – Account ID, Index or object  
`OpenPosition` – Open position ID, Index or object  
`Amount` – Lot count  
Returns command identifier.

#### 4.17 Set Stop / Limit

❖ `function SendSetStopForOrder(Order: OLEVariant; Rate: Double): Integer;`  
❖ `function SendSetLimitForOrder(Order: OLEVariant; Rate: Double): Integer;`  
❖ `function SendSetStopForOpenPosition(OpenPosition: OLEVariant; Rate: Double): Integer;`  
❖ `function SendSetLimitForOpenPosition(OpenPosition: OLEVariant; Rate: Double): Integer;`  
Functions set Stop/Limit for Orders or Open Positions Parameters:  
`Order / OpenPosition` – Order or Open position ID, Index or object  
`Rate` –  
Returns command identifier.

#### 4.18 Change Stop / Limit

❖ `function SendChangeStopForOrder(Order: OLEVariant; NewRate: Double): Integer;`  
❖ `function SendChangeLimitForOrder(Order: OLEVariant; NewRate: Double): Integer;`  
❖ `function SendChangeStopForOpenPosition(OpenPosition: OLEVariant; NewRate: Double): Integer;`  
❖ `function SendChangeLimitForOpenPosition(OpenPosition: OLEVariant; NewRate: Double): Integer;`  
Functions change Stop/Limit for Orders or Open Positions Parameters:  
`Order / OpenPosition` – Order or Open position ID, Index or object  
`NewRate` –  
Returns command identifier.

#### 4.19 Remove Stop / Limit

❖ `function SendRemoveLimitForOrder(Order: OLEVariant): Integer;`  
❖ `function SendRemoveStopForOrder(Order: OLEVariant): Integer;`  
❖ `function SendRemoveStopForOpenPosition(OpenPosition: OLEVariant): Integer;`  
❖ `function SendRemoveLimitForOpenPosition(OpenPosition: OLEVariant): Integer;`  
Functions remove Stop/Limit for Orders or Open Positions Parameters:  
`Order / OpenPosition` – Order or Open position ID, Index or object  
Returns command identifier.

#### 4.20 Events

VTAPI supports events. For more information about event handling see samples and "VTAPI documentation. Implementing Event model in Visual Basic.pdf" file.

❖ `OnNewServerMessage(MsgID: Integer)`

This event occurs when a message is get from server. Use `MsgId` parameter to access message. (e.g. `API.ServerMessages.ItemByID(MsgID)` or `API.GetMessageByID(MsgID)`).

- ❖ `OnOrderChange(OrderID: WideString, Action: Integer)`  
This event occurs when Order changed. Use OrderID parameter to access order. (e.g. `API.Orders.ItemByID(OrderID)` or `API.GetOrderByID(OrderID)`). Action parameter represents change type. Use `MessageType` enumeration to decode (see p. 4.9). **NOTE:** If Action is `MT_REMOVE_ORDER` you will not be able to get order because it is already removed from memory.
- ❖ `OnPositionChange(PositionID: WideString, Action: Integer)`  
This event occurs when Open Position changed. Use PositionID parameter to access open position. (e.g. `API.Positions.ItemByID(PositionID)` or `API.GetOpenPositionByID(PositionID)`). Action parameter represents change type. Use `MessageType` enumeration to decode (see p. 4.9). **NOTE:** If Action is `MT_CLOSE_POSITION` you will not be able to get position because it is already closed and removed from memory.
- ❖ `OnInstrumentChange(InstrumentID: WideString)`  
This event occurs when new dealing rate received. Use InstrumentID parameter to access instrument. (e.g. `API.Instruments.ItemByID(InstrumentID)` or `API.getinstrumentByID(InstrumentID)`).
- ❖ `OnInstrumentChangeEx(InstrumentID: WideString; Bid, Ask: Double; Time: TDateTime)`  
Same as `OnInstrumentChange`, but contains bid, ask and update time.
- ❖ `OnCandleUpdated(InstrumentID: WideString; Interval: TimeInterval; isNewCandle: WordBool);`  
This event occurs when candle is updated. `isNewCandle` indicates if candle was updated or closed. Use `IInstrument.set/getHistorySubscription` to get this event.
- ❖ `OnAccountChange(AccountID: WideString, Action: Integer)`  
This event occurs when Account is changed. Use AccountID parameter to access account. (e.g. `API.accounts.ItemByID(AccountID)` or `API.getAccountByID(AccountID)`). Action parameter represents change type. Use `AccountMessageType` enumeration to decode (see p. 4.9).
- ❖ `OnOrdersChange()`  
This event occurs when orders change completed. (See `VTtraderMINI` and `VBAExcelTrader` example).
- ❖ `OnPositionsChange()`  
This event occurs when positions change completed. (See `VTtraderMINI` and `VBAExcelTrader` example).
- ❖ `OnInstrumentsChange()`  
This event occurs when instruments change completed. Use this event to show dealing rates (see `VTtraderMINI`, `VBAExcelTrader` and `VBAExcelSample` for more details).
- ❖ `OnAccountsChange()`  
This event occurs when accounts change completed. (See `VBAExcelTrader` for more details).
- ❖ `OnConnectionLost()`  
This event occurs when connection with server is lost. (See `VBSample`, `VBAExcelTrader` and `VTtraderMINI` for more details).
- ❖ `OnChangesStart()`  
This event occurs when changes pending. (See `VBAExcelTrader` for more details).
- ❖ `OnChangesComplete()`  
This event occurs when changes done. (See `VBAExcelTrader` for more details).
- ❖ `OnMarketStatusChange()`  
This event occurs when market is opened/closed. Use `MarketStatus` property to determine current status.
- ❖ `OnTradingDayClosed()`  
This event occurs when trading day is closed.
- ❖ `OnNewAnnouncement()`  
Reserved for future server compatibility.

Events order:

`OnChangesStart` – if some changes pending – one or more following events will happen.

`OnInstrumentChangeEx` – possibly several times.  
`OnInstrumentChange` – possibly several times.  
`OnCandleUpdated` – possibly several times.  
`OnInstrumentsChange` – if happened at least one `OnInstrumentChange`.  
`OnPositionChange` – possibly several times.  
`OnPositionsChange` – if happened at least one `OnPositionChange`.  
`OnAccountChange` – possibly several times.  
`OnAccountsChange` – if happened at least one `OnAccountChange`.  
`OnOrderChange` – possibly several times.  
`OnOrdersChange` – if happened at least one `OnOrderChange`.  
`OnChangesComplete` – if some changes done – one or more following events will happened.

#### NOTE:

Delphi 6 users: read notes about event handling in VTtraderMINI source code (MainUnit.pas).

#### 4.21 getHistory

`❖ GetHistory(Instrument: OleVariant; Interval: Integer; BidAsk: WordBool; Count: Integer): WideString;`  
 Returns Instrument history as XML. Format may differ depending on Interval parameter (if T\_TICKS). If error occurs XML contains node "error". Parameters:  
`Instrument` – Instrument ID, Index or object  
`Interval` – history interval. Use Time Interval enumeration (p. 4.9).  
`BidAsk` – BID / ASK  
`Count` – Count of history data. Optional. 500 by default. 2000 max. If 0 - returns all history cached.

#### 4.22 getSetting

`❖ GetSetting(Setting: Interval): OLEVariant; - DEPRICATED. Use settings() function.`  
 Returns specified setting value. Returns -1 if invalid setting specified or error happen.  
`Setting` – Member of Setting enumeration.

#### 4.23 LoggedIn

`❖ LoggedIn: WordBool;`  
 Returns if logged in to server.

#### 4.24 MarketStatus

`❖ LoggedIn: MarketStatus enumeration;`  
 Returns current market status. Use MarketStatus enumeration. (p. 4.9).

#### 4.25 GetClosedPositions

`❖ GetClosedPisitions(DateFrom, DateTo: TDateTime): WideString;`  
 Returns Closed Position list in XML format. Parameters:  
`DateFrom` – Date to select positions from.  
`DateTo` – Date to select positions till.

#### 4.26 LoginInProgress

`❖ LoginInProgress: WordBool;`  
 Returns if login in progress.

#### 4.27 SendAnnouncement

`❖ SendAnnouncement(...);`  
 Reserved for future server compatibility.

#### 4.28 SSL Disabling

`❖ setForceDisableSSL(Value: WordBool);`  
 Sets Disable SLL mode on/off. Must be called BEFORE login.  
  
`❖ getForceDisableSSL: WordBool;`  
 Returns mode on/off.

#### 4.29 History depth limitation

`❖ setHistoryCacheLimit(Value: Integer);`

Allow set limit number of candles and ticks stored in local History. 0 by default – no history limit. However if you call some `GetHistory(... count=N)` this Instrument and Interval history will be limited by N. You can set it to 1 to be able to store only open candles and last rate. Must be called BEFORE login.

❖ `getHistoryCacheLimit: Integer;`  
Returns current depth value.

## 5 Data types

---

### 5.1 IServerMessage

IServerMessage represents an instance of a server message.

Properties:

`Index: Integer -`

`ID: Integer - identifier`

`Sender: Integer - command identifier`

`Level: Integer - Message level. See MessageLevel enumeration (p. 4.9).`

`Time: TDateTime - receive date and time`

`Kind: Integer - Message Type. See MessageType enumeration (p. 4.9).`

`DetailsCount: Integer - number of message lines`

`Text: WideString - full message text`

Methods:

`Details(Index: Integer): WideString - message text line by index`

`GetRefreshRate: Integer - returns refresh rate measured in seconds (3 by default).`

`SetRefreshRate(rate: Integer) - set refresh rate measured in seconds. Values from 3 to 10 accepted.`

Increase refresh rate to minimize network traffic. Decrease refresh rate to get Dealing rates more frequently.

### 5.2 IAccount

IAccount represents an instance of an account.

Properties:

`Currency: WideString - default currency name`

`DefaultAmount: Double - default lot size`

`Group: Integer - account group`

`ID: WideString - identifier`

`Index: Integer: -`

`NetPL: Double - Net Profit / Loss measured in default currency`

`Owner: WideString - owner name`

`OwnerId: WideString - owner id`

`UsebleMargin: Double -`

`UsedMargin: Double -`

`Trader: WideString - trader name`

`Premium: Double -`

`Balance: Double -`

`Commision: Double -`

`Equity: Double -`

`OpenTrade: Double -`

### 5.3 IInstrument

IInstrument represents an instance of an instrument.

Properties:

`Currency1: WideString - first currency name`

`Currency2: WideString - second currency name`

`Id: WideString - identifier`

`Index: Integer -`

`Time: TDateTime - update date and time`

`BID: Double - last sell rate`

`ASK: Double - last buy rate`

`Max: Double - maximal rate`

`Min: Double - minimal rate`

`BuyInterest: Double` – interest for buy  
`SellInterest: Double` – interest for sell  
`points: Integer` – number of digits after decimal point (2 or 4). E.g. EUR/USD has 4 - 1.2789, USD/JPY has 2 - 107.33.  
`pipValue: Double` – value of 1 pips measured in USD.  
`pointSize: Double` – fractional value of points. E.g. EUR/USD has 0.0001 - 1.2789, USD/JPY has 0.01 - 107.33.

#### Methods:

`GetHistory(Interval: Integer; BidAsk: WordBool; Count: Integer): WideString` – returns instrument history. (See p. 4.21)

`setThrowsEvent(Value: WordBool)` – switches on/off `OnInstrumentChange` event for this instrument. Set to True (on) at login. This value has no effect if `IInstruments.getThrowsItemsEvent` is False – event will be switched off.

`getThrowsEvent: WordBool` – returns `OnInstrumentChange` event on/off. All instruments `ThrowsEvent` is set ON at login.

`setSubscribed(value: WordBool);` - switches subscription on/off. `IInstruments.CommitSubscription` method must be called to make changes to take effect.

`getSubscribed: WordBool;` - returns subscription on/off. All instruments subscription is set ON at login.

Difference between `ThrowsEvent` and `Subscription` is: `ThrowsEvent` ONLY switches on/off `OnInstrumentChange[Ex]` event (fields of Instrument s.a. as BID, ASK, etc will be changed from data feed), `Subscription` switches on/off data feed from server.

`setHistorySubscription(Interval: TimeInterval; value: WordBool);` - switches history subscription on/off. History subscription allows to get `OnCandleUpdated` event, indicates if candle was updated, or closed.

`getHistorySubscription(Interval: TimeInterval): WordBool;` - returns history subscription on/off. All instruments history subscription is set OFF at login.

## 5.4 IPosition

`IPosition` represents an instance of an open position.

#### Properties:

`Account: IAccount;` - account object

`BuySell: WordBool` – buy or sell (see constants `bs_sell/bs_buy` p. 4.9)

`CloseRate: Double` –

`Commision: Double` –

`GrossPL: Double` – Gross Profit / Loss measured in default currency

`Id: WideString` – identifier

`Index : integer` –

`Instrument: IInstrument` – instrument object

`Interest: Double` –

`LimitOrder: IConditionalOrder` – stop order (if nil then not assigned)

`Amount: Double` – размер лота

`NetPL: Double` – Net Profit / Loss measured in default currency

`OpenRate: Double` –

`pl: Double` – Profit / Loss measured in pips

`StopOrder: IConditionalOrder` – limit order (if nil then not assigned)

`Time: TDateTime` – creation date and time

#### Methods:

`function Close(Amount, Rate: Double; TraderRange: Integer; Hedge: WordBool): Integer` –

Same as `SendClosePosition` function

`function CreateLimitOrder(Rate: Double): Integer` – Same as `SendSetLimitOpenPosition` function

`function CreateStopOrder(Rate: Double): Integer` – Same as `SendSetStopOpenPosition` function

`function CreateHedge(Account: OLEVariant; Amount: Double): Integer` – Same as

`SendCreateHedge` function

## 5.5 IOrder, IConditionalOrder

`IOrder` represents an instance of an order. `IConditionalOrder` represents an instance of a Stop/Limit order.

#### Common properties:

`Account: IAccount` – account object

`BuySell: WordBool` – Buy or sell (see constants `bs_sell/bs_buy` p. 4.9)

`Duration: WordBool` – execution period (see constants `dur_GTC/dur_Day` p. 4.9)

`Hedge: WideString` –

**ID:** WideString - identifier  
**Index :** integer -  
**Instrument:** IInstrument - instrument object  
**Amount:** Double - Lot count  
**OrderType:** WideString - order type (see constants ot\_Stop/ot\_Limit/ot\_EntryStop/ot\_EntryLimit p. 4.9)  
**Rate:** Double -  
**Time:** TDateTime - creation date and  
**Common methods:**  
**function** Change(NewRate: Double): Integer - Same as SendChangeOrder function  
**function** Remove: Integer - Same as SendRemoveOrder function  
 The following properties and methods only IOrder has:  
**StopOrder:** IConditionalOrder - stop order (if nil then not assigned)  
**LimitOrder:** IConditionalOrder - limit order (if nil then not assigned)  
**function** CreateLimitOrder(Rate: Double): Integer - Same as SendSetLimitForOrder function  
**function** CreateStopOrder(Rate: Double): Integer - Same as SendSetStopForOrder function

## 5.6 IServerMessages

IServerMessages represents object to access the collection of server messages.

Properties:

**Count:** Integer - server messages count (Same as GetMessageCount function)  
**Items(Index: Integer):** IServerMessage - get server message by index (Same as GetMessage function)  
**ItemByID(id: WideString):** IAccount - get by identifier.  
**Capacity:** Integer - set maximal number of server messages stored.  
**CurrentItemID:** Integer - last message identifier.

## 5.7 IAccounts

IAccounts represents object to access the collection of accounts.

Properties:

**Count:** Integer - accounts count (Same as GetAccountCount function)  
**Items(index: integer):** IAccount - get account index (Same as GetAccountByIndex function)  
**ItemByID(id: WideString):** IAccount - get by identifier (Same as GetAccountByID function)

## 5.8 IInstruments

IInstrument represents object to access the collection of instruments.

Properties:

**Count:** Integer - instruments count (Same as GetInstrumentCount function)  
**Items(index: integer):** IInstrument - get instrument by index (Same as GetInstrumentByIndex function)  
**ItemByID(id: WideString):** IInstrument - get instrument by identifier (Same as GetInstrumentByID function)  
**ItemByCurrency(currency1, currency2: WideString):** IInstrument - returns instrument by currency pair (Same as GetInstrumentByCurrency function)  
**setThrowsGroupEvent(Value: WordBool)** - switches on/off OnInstrumentsChange event. Set to True (on) at login. Set to ON at login.  
**getThrowsGroupEvent:** WordBool - returns OnInstrumentsChange event on/off. Set to ON at login.  
**setThrowsItemsEvent(Value: WordBool)** - switches on/off OnInstrumentChange event for ALL instruments. Set to True (on) at login.  
**getThrowsItemsEvent:** WordBool - returns OnInstrumentChange event on/off for ALL instruments.  
**CommitSubscription: Integer;** - posts current set of subscribed instruments to server. Returns command identifier. See also: IInstrument.set/getSubscribed.  
**setSubscriptionForAll(Value: WordBool);** - sets all instrument's subscribed value to provided one.

## 5.9 IPositions

IPositions represents object to access the collection of open positions.

Properties:

**Count:** Integer - open position count (Same as GetOpenPositionCount)  
**Items(index: integer):** IPosition - get open position by index (Same as GetOpenPositionByIndex function)

`ItemByID(id: WideString): IPosition` – get open position by identifier (Same as `GetOpenPositionByID` function)

`GetClosed(DateFrom, DateTo: TDateTime): WideString;` - Returns Closed Position list in XML format. (Same as `GetClosedPositions` function)

### 5.10 IOOrders

IOOrders represents object to access the collection of orders.

Properties:

`Count: Integer` – order (Same as `GetOrderCount`)

`Items(index: integer): IOrder` – get order by index (Same as `GetOrderByIndex` function)

`ItemByID(id: WideString): IOrder` – get order by identifier (Same as `GetOrderByID` function)

Methods:

`function CreateOrder(Account, Instrument: OLEVariant; Duration, TraderRange: Integer; SellBuy: WordBool; Rate, Amount: Double; Hedge, Initial: WordBool): Integer;` - Same as `SendCreateOrder` function)

`function CreateOrderWithSL(Account, Instrument: OLEVariant; Duration, TraderRange: Integer; SellBuy: WordBool; Rate, Amount: Double; Hedge, Initial: WordBool; LimitRate, StopRate: Double; StopRange: Integer): Integer;` - Same as `SendCreateOrderWithSL` function)

### 5.11 ISettings

ISettings represents a set of settings. This interface is only returned by `settings()` function.

Properties:

`LimitRateFromMarket: Integer` – number of pips Limit can be placed close to market.

`StopRateFromMarket: Integer` – number of pips Stop can be placed close to market.

`Function HedgingEnabled(Account: OLEVariant): Boolean` – indicates if Hedging enabled for account.

## 6 Objects description (object-oriented style)

---

Instead of get-functions one can use object-oriented style:

`property ServerMessages: IServerMessages` – to access the collection of server messages

`property Accounts: IAccounts` – to access the collection of account

`property Instruments: IInstruments` – to access the collection of instruments

`property Positions: IPositions` – to access the collection of open positions

`property Orders: IOOrders` – to access the collection of orders

Then you should use `count`, `Items`, `ItemById` properties (see p.p. 5.6 – 5.10) to access objects in collections. Then you should use object's properties (see p.p. 5.1 – 5.5)

For example:

`Orders.Count` – number of orders

`Orders.Items(12)` – order by index

`Orders.ItemById("123143").Rate` – order rate by identifier

`Orders.ItemById("123143").Instrument` – order's instrument by identifier

`Orders.ItemById("123143").Instrument.Currency1` – order's instrument's 1<sup>st</sup> currency by identifier... etc.

`Orders.CreateOrder(Accounts.Items(0), Instruments.ItemByCurrency("EUR", "USD"), constants.dur_GTC, 1; constants.bs_Sell; 1.26, 0.9, false, constants.ord_initial)` – create order for account by index and instrument by currency pair

`Orders.ItemById("5434534").StopOrder.Remove` – remove stop order for order by identifier

`Positions.ItemById("23423423").Close(1.0, 1.27, 1, false)` – close position by identifier

`Positions.Items(5).CreateHedge(Accounts("234234"), 0.9)` – create hedge for position by index for account by identifier... etc.

## 7 Appendix A: Implementing Event model in Visual Basic 6, Visual Basic.NET and C#

---

### Visual Basic 6:

Create the following definition in order to subscribe to event:

```
Public WithEvents API As VTAPI.VT_API
```

You may then define an event; for example, when a message is received by ServerMessage list from the server as a confirmation or rejection of your actions, you should write:

```
Sub API_OnNewServerMessage(ByVal MsgId As Long)
    ' put your code here
End Sub
```

Where MsgID is the identifier of the incoming message.

### **Visual Basic .NET:**

This is a classic connection point model that assumes all COM classes provide IConnectionPointInterface. However, the part that finds connection point is hidden from the user in DLL code. After the usual definition and initialization of VT\_API:

```
Public API As New VTAPI.VT_API ' NO WITHEVENTS
Public eventsHandler As New MyEventsSink
```

You should define the class that modifies the dispatch interface for dual interface IVT\_API:

```
Public Class MyEventsSink
    Implements VTAPI.IVT_APIEvents
    Public APIref As VTAPI.VT_API
    Sub OnNewServerMessage(ByVal MsgID As Integer) Implements VTAPI.IVT_APIEvents.OnNewServerMessage
        ' put your code here like:
        MsgBox(APIref.ServerMessages.Items(MsgID).Text)
    End Sub
End Class
```

Followed by:

```
eventsHandler.APIref = API
API.SetMsgRefresh(eventsHandler)
```

If you want to turn off event support or connect to another implementation of the event interface you should use `API.UnSetMsgRefresh`

### **C#:**

Absolutely like VB.NET:

```
private VTAPIEventHandler events;
```

You should define the class that modifies the dispatch interface for dual interface IVT\_API:

```
public class VTAPIEventHandler : VTAPI.IVT_APIEvents
{
    public void OnNewServerMessage(int msg_index)
    {
        // your code here
    }
    // rest of handlers...
}
```

Followed by:

```
this.events = new VTAPIEventHandler();
vtapi.SetMsgRefresh(this.events);
```