ProfitDLL 64 bits - User Manual

4.0.0.31

Price Book modernization

To better manage theoretical prices during auction, a new mechanism was created to fetch price depth data. When using the new function `SubscribePriceDepth` and the new callback `SetPriceDepthCallback`, price depth data can be fetched with `GetPriceGroup`, which already brings the price depth entries with all calculations done.

New types

- `TConnectorPriceGroup`
- `TConnectorActionType`
- `TConnectorUpdateType`
- `TConnectorBookSideType`
- `TConnectorPriceDepthCallback`

New callbacks

- `SetPriceDepthCallback`

New functions

- `SubscribePriceDepth`
- `UnsubscribePriceDepth`
- `GetPriceDepthSideCount`
- `GetPriceGroup`

Bug Fixes

- Fixed callback `SetTheoreticalPriceCallback` not triggering
- New function to fetch theoretical prices: `GetTheoreticalValues`

4.0.0.30

Added the `AccountType` field to the following structures:

- `TConnectorTradingAccountOut`

4.0.0.28

Position instruments

Added a mechanism to iterate over the instruments that make up an account's position, as well as a mechanism to link orders to a position event.

Modified Structures

Added the EventID field to the following structures:

- `TConnectorTradingAccountPosition`
- `TConnectorOrder`
- `TConnectorOrderOut`

New Types

- `TConnectorAssetPositionListCallback`

New Callbacks

- `SetAssetPositionListCallback`

New Functions

- `EnumerateAllPositionAssets`

Account and sub-account List

Added new mechanisms for retrieving routing accounts and sub-accounts.

New Types

- `TBrokerAccountListCallback`
- `TBrokerSubAccountListCallback`

New Callbacks

- `SetBrokerAccountListChangedCallback`
- `SetBrokerSubAccountListChangedCallback`

New Functions

- `GetAccountCountByBroker`
- `GetAccountsByBroker`

Bug Fixes

- Adjusted the GetAccounts function for cases where the size was larger than the account list.

4.0.0.24

Agent Names

New functions to retrieve agent names.

New Functions

- `GetAgentNameLength`
- `GetAgentName`

Bug Fixes

- Adjusted error codes for GetPositionV2 and SendZeroPositionV2 when there is no position;
- Removed size limitation on Ticker fields;

4.0.0.21

Bug Fixes

- Fixed duplicate order notifications when there are no changes.

4.0.0.20

Trade Callbacks

Added new mechanisms for retrieving trades.

New Types

- `TConnectorTradeCallback`
- `TConnectorTrade`

New Callbacks

- `SetTradeCallbackV2`
- `SetHistoryTradeCallbackV2`

New Functions

- `TranslateTrade`

Order History

To better support downloading order history, an improved mechanism was created for retrieving an account's order history.

New Types

- `TConnectorOrder`
- `TConnectorEnumerateOrdersProc`

New Callbacks

- `SetOrderHistoryCallback`

New Functions

- `HasOrdersInInterval`
- `EnumerateOrdersByInterval`
- `EnumerateAllOrders`

Bug Fixes

- Added a flag to order book callbacks (`SetOfferBookCallback` and `SetOfferBookCallbackV2`)

1. Product Description

The files contained in the zip archive are organized into separate directories for the 64 bit and 32 bit versions. Each directory has the same file structure. In the directory named DLL and Executable, you can find the ProfitDLL.dll file for the 32 bit version and ProfitDLL64.dll for the 64 bit version. Additionally, there is a compiled example in Delphi that can be used to validate the software's functionalities. In the directory named Interface, files containing the declarations of the functions and types necessary to communicate with the DLL in Delphi are provided.

There are also examples for 4 different programming languages in the Example folders

- Delphi
- C#
- C++
- Python

They contain the source code to use the main functionalities of the product.

2. Library Description

The library provides basic communication functions with the Routing and Market Data servers for developing 32 bit or 64 bit applications. The DLL responds to server events and sends them, processed in real time, to the client application, primarily through callbacks that will be described in section `3.2`.

The following sections describe, in more detail, how the communication between the library and the client application is carried out, as well as present the technical details of each function or callback.

3. Library Interface The library exposes several functions that are directly called by the client application, which make requests to the servers or directly to the internal services and structures of the DLL. The types specified in this documentation are coded in Delphi, with specific examples for other programming languages in their respective example files.

All the structures necessary to define the library's functions are defined below:

Definitions:

```
// Pointer Math
PConnectorAccountIdentifierArrayOut = ^TConnectorAccountIdentifierOut;

TConnectorOrderType = (
    cotMarket    = 1,
    cotLimit     = 2,
    cotStopLimit = 4
);

TConnectorOrderSide = (
    cosBuy  = 1,
    cosSell = 2
);

TConnectorPositionType = (
    cptDayTrade     = 1,
    cptConsolidated = 2
);

TConnectorOrderStatus = (
    cosNew                     = 0,
    cosPartiallyFilled         = 1,
    cosFilled                  = 2,
    cosDoneForDay              = 3,
    cosCanceled                = 4,
    cosReplaced                = 5,
    cosPendingCancel           = 6,
    cosStopped                 = 7,
    cosRejected                = 8,
    cosSuspended               = 9,
    cosPendingNew              = 10,
    cosCalculated              = 11,
    cosExpired                 = 12,
    cosAcceptedForBidding      = 13,
    cosPendingReplace          = 14,
    cosPartiallyFilledCanceled = 15,
    cosReceived                = 16,
    cosPartiallyFilledExpired  = 17,
    cosPartiallyFilledRejected = 18,
    cosUnknown                 = 200,
    cosHadesCreated            = 201,
    cosBrokerSent              = 202,
    cosClientCreated           = 203,
    cosOrderNotCreated         = 204,
    cosCanceledByAdmin         = 205,
    cosDelayFixGateway         = 206,
    cosScheduledOrder          = 207
);

TConnectorActionType = (
    atAdd        = 0,
    atEdit       = 1,
```

```
    atDelete     = 2,
    atDeleteFrom = 3,
    atFullBook   = 4
);

TConnectorUpdateType = (
    utAdd          = 0,
    utEdit         = 1,
    utDelete       = 2,
    utInsert       = 3,
    utFullBook     = 4,
    utPrepare      = 5,
    utFlush        = 6,
    utTheoricPrice = 7,
    utDeleteFrom   = 8
);

TConnectorBookSideType = (
    bsBuy  = 0,
    bsSell = 1,
    bsBoth = 254,
    bsNone = 255
);

TConnectorAccountIdentifier = record
    Version : Byte;

    // V0
    BrokerID     : Integer;
    AccountID    : PWideChar;
    SubAccountID : PWideChar;
    Reserved     : Int64;
end;
PConnectorAccountIdentifier = ^TConnectorAccountIdentifier;

TConnectorAccountIdentifierOut = record
    Version : Byte;

    // V0
    BrokerID           : Integer;
    AccountID          : TString0In;
    AccountIDLength    : Integer;
    SubAccountID       : TString0In;
    SubAccountIDLength : Integer;
    Reserved           : Int64;
end;
PConnectorAccountIdentifierOut = ^TConnectorAccountIdentifierOut;

TConnectorAssetIdentifier = record
    Version : Byte;

    // V0
    Ticker   : PWideChar;
    Exchange : PWideChar;
    FeedType : Byte;
end;
```

```
TConnectorAssetIdentifierOut = record
    Version : Byte;

    // V0
    Ticker         : PWideChar;
    TickerLength   : Integer;
    Exchange       : PWideChar;
    ExchangeLength : Integer;
    FeedType : Byte;
end;

TConnectorOrderIdentifier = record
    Version : Byte;

    // V0
    LocalOrderID : Int64;
    ClOrderID    : PWideChar;
end;

TConnectorSendOrder = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    AssetID   : TConnectorAssetIdentifier;
    Password  : PWideChar;
    OrderType : Byte; // TConnectorOrderType
    OrderSide : Byte; // TConnectorOrderSide

    Price     : Double;
    StopPrice : Double;
    Quantity  : Int64;
end;
PConnectorSendOrder = ^TConnectorSendOrder;

TConnectorChangeOrder = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    OrderID   : TConnectorOrderIdentifier;
    Password  : PWideChar;

    Price     : Double;
    StopPrice : Double;
    Quantity  : Int64;
end;
PConnectorChangeOrder = ^TConnectorChangeOrder;

TConnectorCancelOrder = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    OrderID   : TConnectorOrderIdentifier;
```

```
    Password  : PWideChar;
end;
PConnectorCancelOrder = ^TConnectorCancelOrder;

TConnectorCancelOrders = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    AssetID   : TConnectorAssetIdentifier;
    Password  : PWideChar;
end;
PConnectorCancelOrders = ^TConnectorCancelOrders;

TConnectorCancelAllOrders = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    Password  : PWideChar;
end;
PConnectorCancelAllOrders = ^TConnectorCancelAllOrders;

TConnectorZeroPosition = record
    Version : Byte;

    // V0
    AccountID : TConnectorAccountIdentifier;
    AssetID   : TConnectorAssetIdentifier;
    Password  : PWideChar;
    Price     : Double;

    // V1
    PositionType : Byte;
end;
PConnectorZeroPosition = ^TConnectorZeroPosition;

TConnectorAccountType = (
  cutOwner       = 0,
  cutAssessor    = 1,
  cutMaster      = 2,
  cutSubAccount  = 3,
  cutRiskMaster  = 4,
  cutPropOffice  = 5,
  cutPropManager = 6
);

TConnectorTradingAccountOut = record
    Version : Byte;

    // In Fields
    AccountID : TConnectorAccountIdentifier;

    // Out fields
    BrokerName         : PWideChar;
    BrokerNameLength   : Integer;
```

```
    OwnerName          : PWideChar;
    OwnerNameLength    : Integer;

    SubOwnerName       : PWideChar;
    SubOwnerNameLength : Integer;

    AccountFlags       : TFlags;

    // V1
    AccountType        : Byte; // TConnectorAccountType
end;
PConnectorTradingAccountOut = ^TConnectorTradingAccountOut;

TConnectorTradingAccountPosition = record
    Version : Byte;

    // In Fields
    AccountID : TConnectorAccountIdentifier;
    AssetID   : TConnectorAssetIdentifier;

    // Out Fields
    OpenQuantity           : Int64;
    OpenAveragePrice       : Double;
    OpenSide               : Byte;

    DailyAverageSellPrice  : Double;
    DailySellQuantity      : Int64;
    DailyAverageBuyPrice   : Double;
    DailyBuyQuantity       : Int64;

    DailyQuantityD1        : Int64;
    DailyQuantityD2        : Int64;
    DailyQuantityD3        : Int64;
    DailyQuantityBlocked   : Int64;
    DailyQuantityPending   : Int64;
    DailyQuantityAlloc     : Int64;
    DailyQuantityProvision : Int64;
    DailyQuantity          : Int64;
    DailyQuantityAvailable : Int64;

    // V1
    PositionType : Byte;

    // V2
    EventID : Int64;
end;
PConnectorTradingAccountPosition = ^TConnectorTradingAccountPosition;

TConnectorOrder = record
    Version : Byte;

    OrderID           : TConnectorOrderIdentifier;
    AccountID         : TConnectorAccountIdentifier;
    AssetID           : TConnectorAssetIdentifier;
```

```
    Quantity          : Int64;
    TradedQuantity    : Int64;
    LeavesQuantity    : Int64;

    Price             : Double;
    StopPrice         : Double;
    AveragePrice      : Double;

    OrderSide         : Byte; // TConnectorOrderSide
    OrderType         : Byte; // TConnectorOrderType
    OrderStatus       : Byte;
    ValidityType      : Byte;

    Date              : TSystemTime;
    LastUpdate        : TSystemTime;
    CloseDate         : TSystemTime;
    ValidityDate      : TSystemTime;

    TextMessage       : PWideChar;

    // V1
    EventID : Int64;
end;
PConnectorOrder = ^TConnectorOrder;

TConnectorOrderOut = record
    Version : Byte;

    // In Fields
    OrderID           : TConnectorOrderIdentifier;

    // Out Fields
    AccountID         : TConnectorAccountIdentifierOut;
    AssetID           : TConnectorAssetIdentifierOut;

    Quantity          : Int64;
    TradedQuantity    : Int64;
    LeavesQuantity    : Int64;

    Price             : Double;
    StopPrice         : Double;
    AveragePrice      : Double;

    OrderSide         : Byte; // TConnectorOrderSide
    OrderType         : Byte; // TConnectorOrderType
    OrderStatus       : Byte;
    ValidityType      : Byte;

    Date              : TSystemTime;
    LastUpdate        : TSystemTime;
    CloseDate         : TSystemTime;
    ValidityDate      : TSystemTime;

    TextMessage       : PWideChar;
    TextMessageLength : Integer;
```

```
    // V1
    EventID : Int64;
end;
PConnectorOrderOut = ^TConnectorOrderOut;

TConnectorTrade = record
    Version     : Byte;
    TradeDate   : TSystemTime;
    TradeNumber : Cardinal;
    Price       : Double;
    Quantity    : Int64;
    Volume      : Double;
    BuyAgent    : Integer;
    SellAgent   : Integer;
    TradeType   : Byte; //TTradeType
end;
PConnectorTrade = ^TConnectorTrade;

TAssetIDRec = packed record
    pwcTicker : PWideChar;  // Represents the asset name e.g.: "WDOFUT".
    pwcBolsa  : PWideChar;  // Represents the exchange the asset belongs to e.g. (for
Bovespa): "B".
    nFeed     : Integer;    // Data source 0 (Nelogica), 255 (Other).
end;
PAssetIDRec = ^TAssetIDRec;

TAccountRec = packed record
    pwhAccountID     : PWideChar;  // Account Identifier
    pwhTitular       : PWideChar;  // Account Holder Name
    pwhNomeCorretora : PWideChar;  // Broker Name
    nCorretoraID     : Integer;    // Broker Identifier
end;
PAccountRec = ^TAccountRec;

// Delegates
TConnectorEnumerateOrdersProc = function(
    const a_Order : PConnectorOrder;
    const a_Param : LPARAM
) : BOOL; stdcall;

// TConnectorTradingAccountOut.Flags
CA_IS_SUB_ACCOUNT : Cardinal = 1;
CA_IS_ENABLED     : Cardinal = 2;

// TConnectorMarketDataLibrary.Flags
CM_IS_SHORT_NAME  : Cardinal = 1;

// TConnectorPriceGroup.PriceGroupFlags
PG_IS_THEORIC     : Cardinal = 1;

// Exchanges
gc_bvBCB            = 65; // A
gc_bvBovespa        = 66; // B
gc_bvCambio         = 68; // D
gc_bvEconomic       = 69; // E
gc_bvBMF            = 70; // F
```

```
gc_bvMetrics        = 75; // K
gc_bvCME            = 77; // M
gc_bvNasdaq         = 78; // N
gc_bvOXR            = 79; // O
gc_bvPioneer        = 80; // P
gc_bvDowJones       = 88; // X
gc_bvNyse           = 89; // Y

// Status
CONNECTION_STATE_LOGIN        = 0;  // Connection to login server
CONNECTION_STATE_ROTEAMENTO   = 1;  // Connection to routing server
CONNECTION_STATE_MARKET_DATA  = 2;  // Connection to market data server
CONNECTION_STATE_MARKET_LOGIN = 3;  // Login to market data server

LOGIN_CONNECTED    = 0;   // Login server connected
LOGIN_INVALID      = 1;   // Login is invalid
LOGIN_INVALID_PASS = 2;   // Invalid password
LOGIN_BLOCKED_PASS = 3;   // Password locked
LOGIN_EXPIRED_PASS = 4;   // Password expired
LOGIN_UNKNOWN_ERR  = 200; // Internal login error

ROTEAMENTO_DISCONNECTED        = 0;
ROTEAMENTO_CONNECTING          = 1;
ROTEAMENTO_CONNECTED           = 2;
ROTEAMENTO_BROKER_DISCONNECTED = 3;
ROTEAMENTO_BROKER_CONNECTING   = 4;
ROTEAMENTO_BROKER_CONNECTED    = 5;

MARKET_DISCONNECTED = 0;  // Disconnected from market data server
MARKET_CONNECTING   = 1;  // Connecting to market data server
MARKET_WAITING      = 2;  // Waiting for connection
MARKET_NOT_LOGGED   = 3;  // Not logged in to market data server
MARKET_CONNECTED    = 4;  // Connected to market data

CONNECTION_ACTIVATE_VALID   = 0;  // Valid activation
CONNECTION_ACTIVATE_INVALID = 1;  // Invalid activation

// Old Versions

TConnectorOrderTypeV0 = (
    cotLimit  = 0,
    cotStop   = 1,
    cotMarket = 2
);

TConnectorOrderSideV0 = (
    cosBuy  = 0,
    cosSell = 1
);

// TConnectorTradeCallback.Flags
TC_IS_EDIT     : Cardinal = 1;
TC_LAST_PACKET : Cardinal = 2;
```

Error Codes:

Name	Value (Hex)	Value (Dec)	Meaning
NL_OK	0	0	Success
NL_INTERNAL_ERROR	0x80000001	-2147483647	Internal error
NL_NOT_INITIALIZED	0x80000002	-2147483646	Not initialized
NL_INVALID_ARGS	0x80000003	-2147483645	Invalid arguments
NL_WAITING_SERVER	0x80000004	-2147483644	Waiting server data
NL_NO_LOGIN	0x80000005	-2147483643	No login found
NL_NO_LICENSE	0x80000006	-2147483642	No license found
NL_OUT_OF_RANGE	0x80000009	-2147483639	Value is out of the possible range
NL_MARKET_ONLY	0x8000000A	-2147483638	License does not allow trading
NL_NO_POSITION	0x8000000B	-2147483637	No trading position has been found
NL_NOT_FOUND	0x8000000C	-2147483636	Resource has not been found
NL_VERSION_NOT_SUPPORTED	0x8000000D	-2147483635	Resource version is not supported
NL_OCO_NO_RULES	0x8000000E	-2147483634	OCO has no rules
NL_EXCHANGE_UNKNOWN	0x8000000F	-2147483633	Exchange is unknown
NL_NO_OCO_DEFINED	0x80000010	-2147483632	No OCO has been found
NL_INVALID_SERIE	0x80000011	-2147483631	Dataserie is not allowed/is invalid
NL_LICENSE_NOT_ALLOWED	0x80000012	-2147483630	Resource is not allowed in this license
NL_NOT_HARD_LOGOUT	0x80000013	-2147483629	Not in a HardLogout state
NL_SERIE_NO_HISTORY	0x80000014	-2147483628	Dataserie has no historu
NL_ASSET_NO_DATA	0x80000015	-2147483627	Asset has no data
NL_SERIE_NO_DATA	0x80000016	-2147483626	Serie has no data
NL_SERIE_NO_MORE_HISTORY	0x80000018	-2147483624	Dataserie does not have any more history to be requested
NL_SERIE_MAX_COUNT	0x80000019	-2147483623	Série esta no limite de dados possíveis
NL_DUPLICATE_RESOURCE	0x8000001A	-2147483622	Resource is duplicated
NL_UNSIGNED_CONTRACT	0x8000001B	-2147483621	There is an unsiged contract
NL_NO_PASSWORD	0x8000001C	-2147483620	No password has been supplied
NL_NO_USER	0x8000001D	-2147483619	No user has been supplied
NL_FILE_ALREADY_EXISTS	0x8000001E	-2147483618	File already exists
NL_INVALID_TICKER	0x8000001F	-2147483617	Asset is invalid

Name	Value (Hex)	Value (Dec)	Meaning
NL_NOT_MASTER_ACCOUNT	0x80000020	-2147483616	Account is not a master account

3.1. Exposed Functions

The declarations of all exposed functions are found in this section. Some functions take types containing "callback" in their name, which will be described in the next subsection.

```
function DLLInitializeLogin(
    const pwcActivationKey : PWideChar;
    const pwcUser          : PWideChar;
    const pwcPassword      : PWideChar;
    StateCallback          : TStateCallback;
    HistoryCallback        : THistoryCallback;
    OrderChangeCallback    : TOrderChangeCallback;
    AccountCallback        : TAccountCallback;
    NewTradeCallback       : TNewTradeCallback;
    NewDailyCallback       : TNewDailyCallback;
    PriceBookCallback      : TPriceBookCallback;
    OfferBookCallback      : TOfferBookCallback;
    HistoryTradeCallback   : THistoryTradeCallback;
    ProgressCallback       : TProgressCallback;
    TinyBookCallback       : TTinyBookCallback) : Integer; stdcall;

function DLLInitializeMarketLogin(
    const pwcActivationKey : PWideChar;
    const pwcUser          : PWideChar;
    const pwcPassword      : PWideChar;
    StateCallback          : TStateCallback;
    NewTradeCallback       : TNewTradeCallback;
    NewDailyCallback       : TnewDailyCallback
    PriceBookCallback      : TPriceBookCallback;
    OfferBookCallback      : TOfferBookCallback;
    HistoryTradeCallback   : THistoryTradeCallback;
    ProgressCallback       : TProgressCallback;
    TinyBookCallback       : TTinyBookCallback) : Integer; stdcall;

function DLLFinalize: Integer; stdcall;

function SubscribeTicker(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;

function UnsubscribeTicker(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;

function SubscribePriceBook(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;

function UnsubscribePriceBook(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;

function SubscribeOfferBook(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;
```

```
function UnsubscribeOfferBook(pwcTicker : PWideChar; pwcBolsa : PWideChar) : Integer;
stdcall;

function GetAgentNameByID(nID : Integer) : PWideChar; stdcall;

function GetAgentShortNameByID(nID : Integer) : PWideChar; stdcall;

function GetAgentNameLength(nAgentID : Integer; nShortName : Cardinal): Integer;
stdcall;

function GetAgentName(
    nCount : Integer;
    nAgentID : Integer;
    pwcAgent : PWideChar;
    nShortName : Cardinal) : Integer; stdcall;


function GetAccount : Integer; stdcall;

function SendBuyOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    dPrice         : Double;
    nAmount        : Integer) : Int64; stdcall;

function SendSellOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    dPrice         : Double;
    nAmount        : Integer) : Int64; stdcall;

function SendMarketBuyOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    nAmount        : Integer) : Int64; stdcall;

function SendMarketSellOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    nAmount        : Integer) : Int64; stdcall;

function SendStopBuyOrder(
    pwcIDAccount   : PWideChar;
```

```
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    dPrice         : Double;
    dStopPrice     : Double;
    nAmount        : Integer) : Int64; stdcall;

function SendStopSellOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    dPrice         : Double;
    dStopPrice     : Double;
    nAmount        : Integer) : Int64; stdcall;

function SendChangeOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcstrClOrdID  : PWideChar;
    dPrice         : Double;
    nAmount        : Integer) : Integer; stdcall;

function SendCancelOrder(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcClOrdId     : PWideChar;
    pwcSenha       : PWideChar) : Integer; stdcall;

function SendCancelOrders(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar) : Integer; stdcall;

function SendCancelAllOrders(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcSenha       : PWideChar) : Integer; stdcall;

function SendZeroPosition(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar;
    pwcSenha       : PWideChar;
    dPrice         : Double) : Int64; stdcall;

function SendZeroPositionAtMarket(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcTicker      : PWideChar;
```

```
    pwcBolsa       : PWideChar;
    pwcSenha       : PWideChar) : Int64; stdcall;

function GetOrders(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    dtStart        : PWideChar;
    dtEnd          : PWideChar) : Integer; stdcall;

function GetOrder(pwcClOrdId : PWideChar) : Integer; stdcall;

function GetOrderProfitID(nProfitID : Int64): Integer; stdcall;

function GetPosition(
    pwcIDAccount   : PWideChar;
    pwcIDCorretora : PWideChar;
    pwcTicker      : PWideChar;
    pwcBolsa       : PWideChar) : Pointer; stdcall;

function GetHistoryTrades(
    const pwcTicker : PWideChar;
    const pwcBolsa  : PWideChar;
    dtDateStart     : PWideChar;
    dtDateEnd       : PWideChar) : Integer; stdcall;

function SendOrder             (const a_SendOrder    : PConnectorSendOrder)       :
Int64;   stdcall;
function SendChangeOrderV2     (const a_ChangeOrder  : PConnectorChangeOrder)     :
Integer; stdcall;
function SendCancelOrderV2     (const a_CancelOrder  : PConnectorCancelOrder)     :
Integer; stdcall;
function SendCancelOrdersV2    (const a_CancelOrder  : PConnectorCancelOrders)    :
Integer; stdcall;
function SendCancelAllOrdersV2 (const a_CancelOrder  : PConnectorCancelAllOrders) :
Integer; stdcall;
function SendZeroPositionV2    (const a_ZeroPosition : PConnectorZeroPosition)    :
Int64;   stdcall;

function GetAccountCount : Integer; stdcall;

function GetAccounts(
  const a_nStartSource : Integer;
  const a_nStartDest   : Integer;
  const a_nCount       : Integer;
  const a_arAccounts   : PConnectorAccountIdentifierArrayOut
) : Integer; stdcall;

function GetAccountDetails(var a_Account : TConnectorTradingAccountOut) : Integer;
stdcall;

function GetSubAccountCount(const a_MasterAccountID : PConnectorAccountIdentifier) :
Integer; stdcall;

function GetSubAccounts(
  const a_MasterAccountID : PConnectorAccountIdentifier;
  const a_nStartSource    : Integer;
```

```
  const a_nStartDest      : Integer;
  const a_nCount          : Integer;
  const a_arAccounts      : PConnectorAccountIdentifierArrayOut
) : Integer; stdcall;

function GetPositionV2(var a_Position : TConnectorTradingAccountPosition) : Integer;
stdcall;

function GetOrderDetails(var a_Order : TConnectorOrderOut) : Integer; stdcall;

function HasOrdersInInterval(const a_AccountID : PConnectorAccountIdentifier; const
a_dtStart : TSystemTime; const a_dtEnd : TSystemTime) : NResult; stdcall;

function EnumerateOrdersByInterval(
  const a_AccountID    : PConnectorAccountIdentifier;
  const a_OrderVersion : Byte;
  const a_dtStart      : TSystemTime;
  const a_dtEnd        : TSystemTime;
  const a_Param        : LPARAM;
  const a_Callback     : TConnectorEnumerateOrdersProc
) : NResult; stdcall;

function EnumerateAllOrders(
  const a_AccountID    : PConnectorAccountIdentifier;
  const a_OrderVersion : Byte;
  const a_Param        : LPARAM;
  const a_Callback     : TConnectorEnumerateOrdersProc
) : NResult; stdcall;

function TranslateTrade(const a_pTrade : Pointer; var a_Trade : TConnectorTrade) :
NResult; stdcall;

function  SubscribePriceDepth(const a_pAssetID : PConnectorAssetIdentifier) :
Integer; stdcall;
function  UnsubscribePriceDepth(const a_pAssetID : PConnectorAssetIdentifier) :
Integer; stdcall;

function  GetPriceDepthSideCount(const a_pAssetID : PConnectorAssetIdentifier; const
a_nSide : Byte) : Integer; stdcall;
function  GetPriceGroup(const a_pAssetID : PConnectorAssetIdentifier; const a_nSide :
Byte; const a_nPosition : Integer; const a_pPrice : PConnectorPriceGroup) : Integer;
stdcall;

function  GetTheoreticalValues(const a_pAssetID : PConnectorAssetIdentifier; out
a_dPrice : Double; out a_nQuantity : Int64) : Integer; stdcall;

function SetServerAndPort(const strServer, strPort : PWideChar) : Integer; stdcall;

function GetServerClock (var dtDate : Double; var nYear, nMonth, nDay, nHour, nMin,
nSec, nMilisec: Integer) : Integer; stdcall;

function SetDayTrade(bUseDayTrade : Integer): Integer; stdcall; forward;

function SetEnabledHistOrder(bEnabled : Integer) : Integer; stdcall; forward;

function SetEnabledLogToDebug(bEnabled : Integer) : Integer; stdcall; forward;
```

```
function RequestTickerInfo(const pwcTicker : PWideChar; const pwcBolsa : PWideChar) :
Integer; stdcall; forward;

function SubscribeAdjustHistory(pwcTicker : PWideChar; pwcBolsa : PWideChar) :
Integer; stdcall;

function UnsubscribeAdjustHistory(pwcTicker : PWideChar; pwcBolsa : PWideChar) :
Integer; stdcall;

function GetLastDailyClose(const pwcTicker, pwcBolsa: var dClose : Double; bAdjusted
: Integer) : Integer; stdcall;

function SetStateCallback(const a_StateCallback : TStateCallback) : Integer; stdcall;

function SetAssetListCallback(const a_AssetListCallback : TAssetListCallback) :
Integer; stdcall;

function SetAssetListInfoCallback(const a_AssetListInfoCallback :
TAssetListInfoCallback) : Integer; stdcall;

function SetAssetListInfoCallbackV2(const a_AssetListInfoCallbackV2 :
TAssetListInfoCallbackV2) : Integer; stdcall;

function SetInvalidTickerCallback(const a_InvalidTickerCallback :
TInvalidTickerCallback) : Integer; stdcall;

function SetTradeCallback(const a_TradeCallback : TTradeCallback) : Integer; stdcall;

function SetHistoryTradeCallback(const a_HistoryTradeCallback :
THistoryTradeCallback) : Integer; stdcall;

function SetDailyCallback(const a_DailyCallback : TDailyCallback) : Integer; stdcall;

function SetTheoreticalPriceCallback(const a_TheoreticalPriceCallback :
TTheoreticalPriceCallback) : Integer; stdcall;

function SetTinyBookCallback(const a_TinyBookCallback : TTinyBookCallback) : Integer;
stdcall;

function SetChangeCotationCallback(const a_ChangeCotation : TChangeCotation) :
Integer; stdcall;

function SetChangeStateTickerCallback(const a_ChangeStateTicker : TChangeStateTicker)
: Integer; stdcall;

function SetSerieProgressCallback(const a_SerieProgressCallback : TProgressCallback)
: Integer; stdcall;

function SetOfferBookCallback(const a_OfferBookCallback : TOfferBookCallback) :
Integer; stdcall;

function SetOfferBookCallbackV2(const a_OfferBookCallbackV2 : TOfferBookCallbackV2) :
Integer; stdcall;

function SetPriceBookCallback(const a_PriceBookCallback : TPriceBookCallback) :
```

```
Integer; stdcall;

function SetPriceBookCallbackV2(const a_PriceBookCallbackV2 : TPriceBookCallbackV2) :
Integer; stdcall;

function SetAdjustHistoryCallback(const a_AdjustHistoryCallback :
TAdjustHistoryCallback) : Integer; stdcall;

function SetAdjustHistoryCallbackV2(const a_AdjustHistoryCallbackV2 :
TAdjustHistoryCallbackV2) : Integer; stdcall;

function SetAssetPositionListCallback(const a_AssetPositionListCallback :
TConnectorAssetPositionListCallback) : Integer; stdcall;

function SetAccountCallback(const a_AccountCallback : TAccountCallback) : Integer;
stdcall;

function SetHistoryCallback(const a_HistoryCallback : THistoryCallback) : Integer;
stdcall;

function SetHistoryCallbackV2(const a_HistoryCallbackV2 : THistoryCallbackV2) :
Integer; stdcall;

function SetOrderChangeCallback(const a_OrderChangeCallback : TOrderChangeCallback) :
Integer; stdcall;

function SetOrderChangeCallbackV2(const a_OrderChangeCallbackV2 :
TOrderChangeCallbackV2) : Integer; stdcall;

function SetOrderCallback(const a_OrderCallback : TConnectorOrderCallback) : Integer;
stdcall;

function SetOrderHistoryCallback(const a_OrderHistoryCallback :
TConnectorAccountCallback) : NResult; stdcall;

function SetTradeCallbackV2(const a_TradeCallbackV2 : TConnectorTradeCallback) :
NResult; stdcall;

function SetHistoryTradeCallbackV2(const a_HistoryTradeCallbackV2 :
TConnectorTradeCallback) : NResult; stdcall;
```

- DLLInitializeLogin

Name	Type	Description
const pwcActivationKey	PWideChar	Activation key provided for login
const pwcUser	PWideChar	User for login of the account corresponding to the activation key
const pwcPassword	PWideChar	Login password
StateCallback	TStateCallback	Connection state callback

Name	Type	Description
HistoryCallback	THistoryCallback	Order history callback
OrderChangeCallback	TOrderChangeCallback	Order state change callback
AccountCallback	TAccountCallback	Routing account information callback
NewTradeCallback	TNewTradeCallback	Real-time trades callback
NewDailyCallback	TNewDailyCallback	Aggregated daily data callback
PriceBookCallback	TPriceBookCallback	Price market depth information callback
OfferBookCallback	TOfferBookCallback	Offer book information callback
HistoryTradeCallback	THistoryTradeCallback	Historical trade data callback
ProgressCallback	TProgressCallback	Progress callback for some historical request
TinyBookCallback	TTinyBookCallback	Market depth top-level callback

Function to initialize the Market Data and Routing services of the DLL. It will initialize the connection with all servers and create the necessary services for communication. Other functions may return the error status `NL_ERR_INIT` if DLLInitializeLogin is not successful.

- `DLLInitializeMarketLogin`

Name	Type	Description
const pwcActivationKey	PWideChar	Activation key provided for login
const pwcUser	PWideChar	User for login of the account corresponding to the activation key
const pwcPassword	PWideChar	Login password
StateCallback	TStateCallback	Connection state callback
NewTradeCallback	TNewTradeCallback	Real-time trades callback
NewDailyCallback	TNewDailyCallback	Aggregated daily data callback
PriceBookCallback	TPriceBookCallback	Price market depth information callback
OfferBookCallback	TOfferBookCallback	Offer book information callback
HistoryTradeCallback	THistoryTradeCallback	Historical trade data callback
ProgressCallback	TProgressCallback	Progress callback for some historical request
TinyBookCallback	TTinyBookCallback	Market depth top-level callback

Equivalent to the `DLLInitializeLogin` function, but initializes only Market Data services.

- `DLLFinalize`

Function used to terminate the services of the DLL.

- `SetServerAndPort`

Name	Type	Description
const strServer	Double	Address of the Market Data server
const strPort	Integer	Port of the Market Data server

This is used to connect to specific Market Data servers and needs to be called before initialization (DLLInitialize or InitializeMarket).

`Important`: Only use this function with guidance from the development team; the DLL operates best by internally selecting the servers.

- `GetServerClock`

Name	Type	Description
var dtDate	Double	Date encoded as Double
var nYear	Integer	Year
var nMonth	Integer	Month
var nDay	Integer	Day
var nHour	Integer	Hour
var nMin	Integer	Minute
var nSec	Integer	Second
var nMilisec	Integer	Millisecond

Returns the time of the Market Data server; it can only be called after initialization. The parameter dtDate corresponds to a reference for Double that follows the TDateTime standard of Delphi, as described in http://docwiki.embarcadero.com/Libraries/Sydney/en/System.TDateTime. The other parameters are also passed by reference to the caller and represent the calendar date values of the value encoded in the dtDate parameter.

- `GetLastDailyClose`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset
var dClose	Double	Returned closing value of the last session

Name	Type	Description
bAdjusted	Integer	Indicates whether to adjust the price

The function returns the closing value (dClose) of the candle prior to the current day, according to the bAdjusted parameter. If bAdjusted is 0, the unadjusted value is returned; otherwise, the adjusted value is returned.

For the function to return NL_OK with data, SubscribeTicker must have been called previously for the same asset. On the first call to the function, data is requested from the server, and the function returns NL_WAITING_SERVER.

All subsequent calls for the same asset return the data already loaded. Invalid assets return NL_ERR_INVALID_ARGS. If the daily series data or adjustments are not previously loaded, this call will load them and consequently trigger the progressCallback and adjustHistoryCallback callbacks.

- `SubscribeTicker`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

This is used to receive real-time quotes for a specific asset. The information is received after subscription as soon as it becomes available through the callback specified in the `NewTradeCallback` parameter of the initialization function. In case of an invalid ticker request, an event will be triggered in the callback defined by `SetInvalidTickerCallback`. UnsubscribeTicker disables this service.

- `UnsubscribeTicker`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

Requests the Market Data service to stop sending real-time quotes for a specific asset.

- `SubscribeOfferBook`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

This is used to receive real-time information from the order book. The information is received after subscription as soon as it becomes available through the callback specified in the `OfferBookCallback` parameter of the initialization function. In case of an invalid ticker request, an event will be triggered in the callback defined by `SetInvalidTickerCallback`. UnsubscribeOfferBook disables this service.

- `UnsubscribeOfferBook`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

Requests the Market Data service to stop sending real-time order book updates for a specific asset.

- `SubscribePriceBook`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

It is used to receive real-time price market depth information. The information is received after subscription as soon as it becomes available through the callback specified in the `PriceBookCallback` parameter of the initialization function. In case of an invalid ticker request, an event will be triggered in the callback defined by `SetInvalidTickerCallback`. UnsubscribePriceBook disables this service.

Deprecated: This function has been replaced by `SubscribePriceDepth`.

- `UnsubscribePriceBook`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

Requests the Market Data service to stop sending real-time price market depth updates for a specific asset.

Deprecated: This function has been replaced by `UnsubscribePriceDepth`.

The calls for Subscribe and Unsubscribe `SubscribeTicker`, `UnsubscribeTicker`, `SubscribePriceBook`, `UnsubscribePriceBook`, `SubscribeOfferBook`, `UnsubscribeOfferBook` receive their parameters in the following pattern:

- Ticker: PETR4, Exchange: B
- Ticker: WINFUT, Exchange: F

More examples of exchanges can be found in the declarations section.

- `SubscribeAdjustHistory`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

This is used to receive the adjustment history for the specified ticker asset. It is necessary to provide the callback function `SetAdjustHistoryCallback` or `SetAdjustHistoryCallbackV2` to use this subscribe. In case of an invalid ticker request, an event will be triggered in the callback defined by `SetInvalidTickerCallback`.

- `UnsubscribeAdjustHistory`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

Requests the Market Data service to stop sending adjustment information for a specific asset.

- `GetAgentNameByID` e `GetAgentShortNameByID`

Name	Type	Description
nID	Integer	Identifier of the trading agent

The returned value provides the full name and abbreviated name of this agent, respectively.

Deprecated: Use `GetAgentName` with `GetAgentNameLength` instead to get the agent name.

- `GetAgentNameLength`

Name	Type	Description
nID	Integer	Identifier of the trading agent
nShortFlag	Cardinal	Set the search to the full name or abbreviation.

The returned value represents the size of the agent's name, whether it's the full name or the abbreviation.

- `GetAgentName`

Name	Type	Description
nAgentLength	Integer	Size of the string name
nID	Integer	Identifier of the trading agent
pwcAgent	PWideChar	Pointer that will receive the agent's name
nShortFlag	Cardinal	Set the search to the full name or abbreviation.

The returned value provides the full or abbreviated name of the agent according to the defined Flag. It is necessary to send the string length, which can be obtained using the `GetAgentNameLength` function.

- `GetHistoryTrades`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset
dtDateStart	PWideChar	Start date of the request in the format DD/MM/YYYY HH:mm:SS (mm = minute, MM = month)
dtDateEnd	PWideChar	End date of the request in the format DD/MM/YYYY HH:mm:SS (mm = minute, MM = month)

This is used to request historical information for an asset starting from a specific date (pwcTicker = 'PETR4'; dtDateStart = '06/08/2018 09:00:00'; dtDateEnd = '06/08/2018 18:00:00'). The return will be given in the callback function `THistoryTradeCallback` specified as a parameter in the initialization function. The `TProgressCallback` will return the download progress (from 1 to 100).

- `SetDayTrade`

Name	Type	Description
bUseDayTrade	Integer	Indicates whether to use the day trade flag (1 true, 0 false)

This function is available for clients whose brokers have day trade risk control. Thus, orders are sent with the DayTrade tag. The parameter is a boolean (0 = False, 1 = True). By setting it to true, all orders will be sent with day trade mode activated. To deactivate, simply set it to false.

- `SetEnabledLogToDebug`

Name	Type	Description
bEnabled	Integer	Indicates whether to save debug logs

Function to determine if the DLL should save logs for debugging (1 = save / 0 = do not save).

- `RequestTickerInfo`

Name	Type	Description
const pwcTicker	PWideChar	Ticker of the asset
const pwcBolsa	PWideChar	Exchange of the asset

This is used to fetch new information about the asset (ex., ISIN). The response is returned in the callbacks `TAssetListInfoCallback`, `TAssetListInfoCallbackV2`, and `TAssetListCallback`, provided they have been

set in the DLL via the functions `SetAssetListInfoCallback`, `SetAssetListInfoCallbackV2`, and `SetAssetListCallback`. In case of an invalid ticker request, an event will be triggered in the callback defined by `SetInvalidTickerCallback`.

The functions below provide a callback address for the DLL to return information. They are optional for using the library. If they are not specified, the corresponding information will not be provided when requested.

- `SetChangeCotationCallback`

Used to set a callback function of type `TChangeCotation`, this function notifies whenever the asset undergoes a price change.

- `SetAssetListCallback`

Used to set a callback function of type `TAssetListCallback`, which is responsible for returning asset information.

- `SetAssetListInfoCallback`

Used to set a callback function of type `TAssetListInfoCallback`, which is responsible for returning asset information and provides additional details compared to `AssetListCallback`.

- `SetAssetListInfoCallbackV2`

Similar to `SetAssetListInfoCallback`, but returns information about the sector, subsector, and segment.

- `SetInvalidTickerCallback`

Used to define a callback function of type `TInvalidTickerCallback`, responsible for returning response to invalid ticker requests.

- `SetChangeStateTickerCallback`

Used to set the callback `TChangeStateTicker`, which informs about changes in the ticker state, such as whether the asset is in auction, suspended, in pre-closing, after market, or closed.

- `SetAdjustHistoryCallback`

Used to set the callback `TAdjustHistoryCallback`, which informs about the adjustment history of the ticker.

- `SetAdjustHistoryCallbackV2`

Used to set the callback `TAdjustHistoryCallbackV2`, which provides information about the adjustment history of the ticker.

- `SetTheoreticalPriceCallback`

Used to set the callback function of type `TTheoreticalPriceCallback`, which receives theoretical prices and quantities during the auction.

- `SetHistoryCallbackV2`

Used to set the callback function of type `THistoryCallbackV2`, which is similar to `THistoryCallback` and receives order history.

Deprecated: Use `SerOrderHistoryCallback` instead.

- `SetOrderChangeCallbackV2`

Used to set the callback function of type `TOrderChangeCallbackV2`, which is similar to `TOrderChangeCallback` and receives updates about orders.

Deprecated: Use `SetOrderCallback` instead.

- `SetOfferBookCallbackV2`

Used to set the callback function of type `TOfferBookCallbackV2`, which is similar to `TOfferBookCallback` and receives the order book in a new format.

- `SetPriceBookCallbackV2`

Used to set the callback function of type `TPriceBookCallbackV2`, which is similar to `TPriceBookCallback` and receives the price market depth in a new format.

Deprecated: This function has been replaced by `SetPriceDepthCallback`.

- `SetStateCallback`

Used to define the callback function of type `TStateCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

- `SetTradeCallback`

Used to define the callback function of type `TTradeCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

Deprecated: Use `SetTradeCallbackV2` instead.

- `SetHistoryTradeCallback`

Used to define the callback function of type `THistoryTradeCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

Deprecated: Use `SetHistoryTradeCallbackV2` instead.

- `SetDailyCallback`

Used to define the callback function of type `TDailyCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

- `SetSerieProgressCallback`

Used to define the callback function of type `TProgressCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

- `SetOfferBookCallback`

Used to define the callback function of type `TOfferBookCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

- `SetPriceBookCallback`

Used to define the callback function of type `TPriceBookCallback`. This overrides the callback defined by `DLLInitializeLogin` or `DLLInitializeMarketLogin`.

Deprecated: This function has been replaced by `SetPriceDepthCallback`.

- `SetAssetPositionListCallback`

Used to define the callback function of type `TConnectorAssetPositionListCallback`. Fired when there is a change in the account positions. Send the altered `TConnectorAssetIdentifier`, or in the case of a full list change, a `TConnectorAssetIdentifier` with the Ticker value on -1. This overrides the callback defined on `TAccountCallback`.

- `SetAccountCallback`

Used to define the callback function of type `TAccountCallback`. This overrides the callback defined by `DLLInitializeLogin`.

- `SetHistoryCallback`

Used to define the callback function of type `THistoryCallback`. This overrides the callback defined by `DLLInitializeLogin`.

Deprecated: Use `SerOrderHistoryCallback` instead.

- `SetOrderChangeCallback`

Used to define the callback function of type `TOrderChangeCallback`. This overrides the callback defined by `DLLInitializeLogin`.

Deprecated: Use `SetOrderCallback` instead.

- `SetOrderCallback`

Used to define the callback function of type `TConnectorOrderCallback`. If a callback is defined for `SetOrderHistoryCallback`, this callback is only fired when there are changes (or creation of) in a single order. Otherwise, it will be fired for all order events.

- `SetOrderHistoryCallback`

Used to define a callback function of type `TConnectorAccountCallback`. Fired when the order history for a particular account has completed loading.

When subscribing to this callback, the callbacks defined in `SetHistoryCallback`, `SetHistoryCallbackV2` and `SetOrderCallback` will **not** fire when, *and only when*, the order history is loaded. Other uses for those callbacks retain their behavior.

- `SetTradeCallbackV2`

Used to define a callback function of type `TConnectorTradeCallback`. Fired when an instrument receives a new trade. Use the function `TranslateTrade` to translate the pointer received in this callback.

The `a_nFlags` parameter can have the flag `TC_IS_EDIT`, indicating an edit.

- `SetHistoryTradeCallbackV2`

Used to define a callback function of type `TConnectorTradeCallback`. Fired while receiving trade history. Use the function `TranslateTrade` to translate the pointer received in this callback.

The `a_nFlags` parameter can have the flag `TC_LAST_PACKET`, indicating the last trade of that particular history.

- `SetPriceDepthCallback`

Used to define a funtion of type `TConnectorPriceDepthCallback`. Fired for any and all price depth update. If more than one notification is going to happen in rapid succession, there will be a `Prepare` and `Flush` notification to specify the begining and end of the notifications.

The functions described below are only available for initialization with routing after using the `DLLInitializeLogin` function during initialization.

- `GetAccount`

Function that returns information about linked accounts through the `TAccountCallback` passed as a parameter to the initialization function.

- `SendBuyOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
dPrice	Double	Target price
nAmount	Integer	Quantity to be traded

Sends a limit buy order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendSellOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
dPrice	Double	Target price
nAmount	Integer	Quantity to be traded

Sends a limit sell order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendMarketBuyOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
nAmount	Integer	Quantity to be traded

Sends a market buy order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendMarketSellOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
nAmount	Integer	Quantity to be traded

Sends a market sell order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendStopBuyOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded

Name	Type	Description
pwcBolsa	PWideChar	Exchange of the asset to be traded
dPrice	Double	Target purchase price
dStopPrice	Double	Stop price
nAmount	Integer	Quantity to be traded

Sends a stop buy order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendStopSellOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
dPrice	Double	Target selling price
dStopPrice	Double	Stop price
nAmount	Integer	Quantity to be traded

Sends a stop sell order. Returns the internal ID (per session) of the order that can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendOrder`.

- `SendChangeOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcstrClOrdID	PWideChar	ClOrdID of the order to be modified (provided in OrderChangeCallback)
dPrice	PWideChar	Target price after edit

Name	Type	Description
nAmount	Integer	Quantity after edit

Sends a modification order. When modifying a stop order, the stop price must be provided as the target price, and the limit price will be calculated based on the same offset.

Function deprecated in favor of function `SendChangeOrderV2`.

- `SendCancelOrder`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcClOrdId	PWideChar	ClOrdID of the order to be canceled (provided in OrderChangeCallback)
pwcSenha	PWideChar	Routing password

Sends a cancellation order. The result of the cancellation request can be monitored in `TOrderChangeCallback`.

Function deprecated in favor of function `SendCancelOrderV2`.

- `SendCancelOrders`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcSenha	PWideChar	Routing password
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded

Sends an order to cancel all orders for an asset. The result of the cancellation request can be monitored in `TOrderChangeCallback` for each canceled order.

Function deprecated in favor of function `SendCancelOrdersV2`.

- `SendCancelAllOrders`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)

Name	Type	Description
pwcSenha	PWideChar	Routing password

Sends an order to cancel all open orders for all assets. The result of the cancellation request can be monitored in `TOrderChangeCallback` for each canceled order.

Function deprecated in favor of function `SendCancelAllOrdersV2`.

- `SendZeroPosition`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
pwcSenha	PWideChar	Routing password
dPrice	Double	Order price

Sends an order to close the position of a specific asset. Returns the internal ID (per session) of the closing order, which can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendZeroPositionV2`.

- `SendZeroPositionAtMarket`

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
pwcTicker	PWideChar	Ticker of the asset to be traded
pwcBolsa	PWideChar	Exchange of the asset to be traded
pwcSenha	PWideChar	Routing password

Sends an order to close the position of a specific asset at market value. Returns the internal ID (per session) of the closing order, which can be compared with the return of `THistoryCallback`.

Function deprecated in favor of function `SendZeroPositionV2`.

- `GetOrders`

Name	Type	Description

Name	Type	Description
pwcIDAccount	PWideChar	Account identifier (provided in GetAccount)
pwcIDCorretora	PWideChar	Broker identifier (provided in GetAccount)
dtStart	PWideChar	Start date in the format DD/MM/YYYY
dtEnd	PWideChar	End date in the format DD/MM/YYYY

Function that returns orders within a specified period. The return is made through the callback `THistoryCallback`, passed as a parameter to the initialization function.

Function deprecated in favor of functions `HasOrdersInInterval`, `EnumerateOrdersByInterval` and `EnumerateAllOrders`.

- `GetOrder`

Name	Type	Description
pwcClOrdId	PWideChar	ClOrdID of the order to be returned

Function that returns order data based on a ClOrdID. The return is made through the callback `TOrderChangeCallback`, passed as a parameter to the initialization function.

Function deprecated in favor of function `GetOrderDetails`.

- `GetOrderProfitID`

Name	Type	Description
nProfitID	Int64	ProfitID of the order to be returned

Function that returns order data based on a ProfitID (internal ID per session). The return is made through the callback `TOrderChangeCallback`, passed as a parameter to the initialization function. The ProfitID is valid only during the application's execution, unlike the ClOrdID. This ID is the return from the order sending functions.

Function deprecated in favor of function `GetOrderDetails`.

- `GetPosition`

Function that returns the position for a given ticker. It returns a data structure specified below, with a total size of (91 + N + S + K) bytes:

Field/Description	Type	Size
Number of accounts	Integer	4 bytes
Buffer size	Integer	4 bytes

Field/Description	Type	Size
Broker ID	Integer	4 bytes
N size of Account string	Short	2 bytes
Account string	(Array of characters)	N bytes
S size of Holder string	Short	2 bytes
Holder string	(Array of characters)	S bytes
K size of Ticker string	Short	2 bytes
Ticker string	(Array of characters)	K bytes
Intraday nQty	Integer	4 bytes
Intraday dPrice	Double	8 bytes
Day SellAvgPriceToday	Double	8 bytes
Day SellQtyToday	Integer	4 bytes
Day BuyAvgPriceToday	Double	8 bytes
Day BuyQtyToday	Integer	4 bytes
Custody Quantity in D+1	Integer	4 bytes
Custody Quantity in D+2	Integer	4 bytes
Custody Quantity in D+3	Integer	4 bytes
Custody Quantity blocked	Integer	4 bytes
Custody Pending Quantity	Integer	4 bytes
Custody Allocated Quantity	Integer	4 bytes
Custody Provisioned Quantity	Integer	4 bytes
Custody Position Quantity	Integer	4 bytes
Custody Available Quantity	Integer	4 bytes
Position Side	Byte	1 byte

The field "Position Side" corresponds to an enumerated type described below:

- Purchased = 1
- Sold = 2
- Unknown = 0

Function deprecated in favor of function `GetPositionV2`.

- `SendOrder`

Name	Type	Description

Name	Type	Description
Version	Byte	Structure version. Supported: 0, 1
AccountID	TConnectorAccountIdentifier	Structure for the account identifier
AssetID	TConnectorAssetIdentifier	Structure for the asset identifier
Password	PWideChar	Plain text routing password
OrderType	Byte	Indicates if it is a Limit, Market, or Stop order
OrderSide	Byte	Indicates if it is a buy or sell order
Price	Double	Order price, market orders should be -1
StopPrice	Double	Stop price, non-stop orders should be -1
Quantity	Int64	Quantity

Function to send orders. Accepts a pointer to a structure of type `TConnectorSendOrder` as a parameter. It is possible to send all types of orders in a single function, and it accepts both accounts and sub-accounts. In case of success, it returns the local order ID, in the case of a error, returns a error code. The order status can be tracked by the callback defined in `SetOrderCallback`.

As of version 4.0.0.18, it is possible to use version 1 of the structure. From this version onwards, fields `OrderType` and `OrderSide` are sychronized with the values defined in `GetOrderDetails` and are defined by `TConnectorOrderType` and `TConnectorOrderSide`.

Prior to version 4.0.0.18, only version 0 is supported, and the fields `OrderType` and `OrderSide` are defined by the enumerations `TConnectorOrderTypeV0` and `TConnectorOrderSideV0`.

- `SendChangeOrderV2`

Name	Type	Description
Version	Byte	Structure version. Supported: 0
AccountID	TConnectorAccountIdentifier	Structure for the account identifier
OrderID	TConnectorOrderIdentifier	Structure for the order identifier. It is possible to inform only one of the IDs.
Password	PWideChar	Plain text routing password
Price	Double	New order price
StopPrice	Double	New stop price
Quantity	Int64	New quantity

Sends a modification order. Accepts a pointer to a structure of type `TConnectorChangeOrder` as a parameter. It accepts both accounts and sub-accounts. The order status can be tracked by the callback defined in `SetOrderCallback`.

- `SendCancelOrderV2`

Name	Type	Description
Version	Byte	Structure version. Supported: 0
AccountID	TConnectorAccountIdentifier	Structure for the account identifier
OrderID	TConnectorOrderIdentifier	Structure for the order identifier. It is possible to inform only one of the IDs.
Password	PWideChar	Plain text routing password

Sends a cancellation order. Accepts a pointer to a structure of type `TConnectorCancelOrder` as a parameter. It accepts both accounts and sub-accounts. The result of the cancellation request can be tracked by the callback defined in `SetOrderCallback`.

- `SendCancelOrdersV2`

Name	Type	Description
Version	Byte	Structure version. Supported: 0
AccountID	TConnectorAccountIdentifier	Structure for the account identifier
AssetID	TConnectorAssetIdentifier	Structure for the asset identifier
Password	PWideChar	Plain text routing password

Sends an order to cancel all orders for an asset. Accepts a pointer to a structure of type `TConnectorCancelOrders` as a parameter. It accepts both accounts and sub-accounts. The result of the cancellation request can be tracked by the callback defined in `SetOrderCallback` for each canceled order.

- `SendCancelAllOrdersV2`

Name	Type	Description
Version	Byte	Structure version. Supported: 0
AccountID	TConnectorAccountIdentifier	Structure for the account identifier
Password	PWideChar	Plain text routing password

Sends an order to cancel all open orders for all assets. Accepts a pointer to a structure of type `TConnectorCancelAllOrders` as a parameter. It accepts both accounts and sub-accounts. The result of the cancellation request can be tracked by the callback defined in `SetOrderCallback` for each canceled order.

- `SendZeroPositionV2`

Name	Type	Description
Version	Byte	Structure version. Supported: 0 .. 1
AccountID	TConnectorAccountIdentifier	Identifier to the account
AssetID	TConnectorAssetIdentifier	Identifier to the asset

Name	Type	Description
Password	PWideChar	Trading password
Price	Double	Zero Price
PositionType	Byte (`TConnectorPositionType`)	Indicates the position type

Sends an order to zero the position of a specific asset. For market zeroing, the price should be -1. Accepts both accounts and sub-accounts. Accepts a pointer to a structure of type `TConnectorZeroPosition` as a parameter. In case of success, it returns the local order ID. The order status can be tracked by the callback defined in `SetOrderCallback`.

Starting from version 1, `PositionType` is required;

- `GetAccountCount`

Returns the total number of loaded accounts. Does not include sub-accounts.

- `GetAccounts`

Name	Type	Description
a_nStartSource	Integer	Index to start account searches
a_nStartDest	Integer	Index of the `a_arAccounts` array indicating where to start writing the IDs
a_nCount	Integer	Number of accounts to be searched
a_arAccounts	PConnectorAccountIdentifierArrayOut	Pointer to the first index of an array of type `TConnectorAccountIdentifierArrayOut`

Function to fetch account identifiers. In case of success, it returns the number of accounts found. Does not fetch sub-accounts.

- `GetAccountDetails`

Name	Type	Description
a_Account	TConnectorTradingAccountOut	Pointer to account details

Function to return the details of an account or sub-account. The account identifier must be provided in the pointer.

- `GetAccountCountByBroker`

Name	Type	Description
a_nBrokerID	Integer	Broker identifier

Returns the total number of loaded accounts filtered by the broker. Does not include sub-accounts.

- `GetAccountsByBroker`

Name	Type	Description
a_nBrokerID	Integer	Broker identifier
a_nStartSource	Integer	Index to start account searches
a_nStartDest	Integer	Index of the `a_arAccounts` array indicating where to start writing the IDs
a_nCount	Integer	Number of accounts to be searched
a_arAccounts	PConnectorAccountIdentifierArrayOut	Pointer to the first index of an array of type `TConnectorAccountIdentifierArrayOut`

Function to fetch account identifiers filtered by the broker. In case of success, it returns the number of accounts found. Does not fetch sub-accounts.

- `GetSubAccountCount`

Name	Type	Description
a_MasterAccountID	PConnectorAccountIdentifier	Master account containing the sub-accounts

Returns the total number of sub-accounts for an account.

- `GetSubAccounts`

Name	Type	Description
a_MasterAccountID	PConnectorAccountIdentifier	Master account containing the sub-accounts
a_nStartSource	Integer	Index to start account searches
a_nStartDest	Integer	Index of the `a_arAccounts` array indicating where to start writing the IDs
a_nCount	Integer	Number of accounts to be searched
a_arAccounts	PConnectorAccountIdentifierArrayOut	Pointer to the first index of an array of type `TConnectorAccountIdentifierArrayOut`

Function to fetch the identifiers of the sub-accounts of an account. In case of success, it returns the number of sub-accounts found.

- `GetPositionV2`

Name	Type	Description
a_Position	TConnectorTradingAccountPosition	Position data

Function that returns the position for a specific account/sub-account and asset. The account and asset identifiers must be provided in the pointer.

Starting from `TConnectorTradingAccountPosition`'s version 1, `PositionType` is required (`TConnectorPositionType`).

The field EventID is related to the EventID received on the `TConnectorAssetPositionListCallback` callback.

- `GetOrderDetails`

Name	Type	Description
a_Order	TConnectorOrderOut	Pointer to order details

Function to return the details of an order. The order identifier must be provided in the pointer. `TSystemTime` is defined in SystemTime.

- `HasOrdersInInterval`

Name	Type	Description
a_AccountID	PConnectorAccountIdentifier	Identifier for the account
a_dtStart	TSystemTime	Start date
a_dtEnd	TSystemTime	Final date

This functions returns whether the order history for an account has been loaded at the specified interval. This function ignores the time part for the interval. Subaccount use their master account for their order history.

If the order history has been loaded, this function returns `NL_OK`. Otherwise, a order history request will be sent to the server, and this function `NL_WAITING_SERVER`. Multiple calls to this function do not generate multiple requests to the server. If the date paremeters are invalid, this function returns `NL_OUT_OF_RANGE`. Neither date can be greater then the date returned by `GetServerClock`.

If there is a server request, the callback defined by `SetOrderHistoryCallback` will notify when the history has been loaded.

Once the orders are loaded, they can be iterate with the functionas `EnumerateOrdersByInterval` or `EnumerateAllOrders`.

- `EnumerateOrdersByInterval`

Name	Type	Description
a_AccountID	PConnectorAccountIdentifier	Identifier for the account or subaccount
a_OrderVersion	Byte	Structure version for the Order pointer in `a_Callback`
a_dtStart	TSystemTime	Start datetime
a_dtEnd	TSystemTime	Final datetime

Name	Type	Description
a_Param	LPARAM	An application-defined value passed along to `a_Callback`
a_Callback	TConnectorEnumerateOrdersProc	An applcation-defined function to be callbed

This function iterates over the orders of an account, filtering for subaccount (if supplied), and filtering for the specified datetime interval. For each order which fits the criteria, `a_Callback` is called.

The `PConnectorOrder` pointer in `a_Callback` will have the version specified by `a_OrderVersion`. The data in the pointer is destroyed after each iteraction. No action is performed on `a_Param`, this paremeter is only passed along to `a_Callback`.

If there is not a order history for the specified interval, this function will request it, as if calling the function `HasOrdersInInterval`, and this function will return `NL_WAITING_SERVER`. If the datetime paremeters are invalid, this function returns `NL_OUT_OF_RANGE`. Neither datetime can be greater then the datetime returned by `GetServerClock`.

In case of success, this function will only return once the iration ends, returning `NL_OK`. It is possible to end the iteration earlier by returning `FALSE` on `a_Callback`.

- `EnumerateAllOrders`

Name	Type	Description
a_AccountID	PConnectorAccountIdentifier	Identificador da conta/subconta
a_OrderVersion	Byte	Versão do ponteiro da ordem para retornar em `a_Callback`
a_Param	LPARAM	Parâmetro definido pelo implementador, retornado em `a_Callback`
a_Callback	TConnectorEnumerateOrdersProc	Função para receber os ordens

This function iterates over all the orders of an account, filtering for subaccount (if supplied). For each order which fits the criteria, `a_Callback` is called.

The `PConnectorOrder` pointer in `a_Callback` will have the version specified by `a_OrderVersion`. The data in the pointer is destroyed after each iteraction. No action is performed on `a_Param`, this paremeter is only passed along to `a_Callback`.

Unlike the function `EnumerateOrdersByInterval`, this function does not perform server requests, and returns a success even if there are no orders loaded in the order history.

In case of success, this function will only return once the iration ends, returning `NL_OK`. It is possible to end the iteration earlier by returning `FALSE` on `a_Callback`.

- `EnumerateAllPositionAssets`

Name	Type	Description

Name	Type	Description
a_AccountID	PConnectorAccountIdentifier	Identifier for the account or subaccount
a_AssetVersion	Byte	Structure version for the Asset identifier in `a_Callback`
a_Param	LPARAM	An application-defined value passed along to `a_Callback`
a_Callback	TConnectorEnumerateAssetProc	An application-defined function to be called

This function iterates over all the open positions of an account, filtering for subaccount (if supplied), For each asset found, `a_Callback` is called.

The `TConnectorAssetIdentifier` identifier in `a_Callback` will have the version specified by `a_AssetVersion`. The data in the identifier is destroyed after each iteraction. No action is performed on `a_Param`, this paremeter is only passed along to `a_Callback`.

In case of success, this function will only return once the iration ends, returning `NL_OK`. It is possible to end the iteration earlier by returning `FALSE` on `a_Callback`.

- `TranslateTrade`

Name	Type	Description
a_pTrade	Pointer	Pointer with trade data in need of translation
a_Trade	TConnectorTrade	Versioned structure which will receive the translated data

This function translates trades from callbacks of type `TConnectorTradeCallback`.

- `SubscribePriceDepth`

Name	Type	Description
a_pAssetID	PConnectorAssetIdentifier	Asset identifier to subscribe to

Performs a price depth subscription for an asset. Notifications for price depth updates are notified in the function defined by the callback `SetPriceDepthCallback`.

- `UnsubscribePriceDepth`

Name	Type	Description
a_pAssetID	PConnectorAssetIdentifier	Asset identifier to unsubscribe from

Removes a price depth subscription for an asset.

- `GetPriceDepthSideCount`

Name	Type	Description

Name	Type	Description
a_pAssetID	PConnectorAssetIdentifier	Asset identifier that is subscribed to Price Depth
a_nSide	Byte	Book Side, 0 foy buy or 1 for sell

If successful, this function returns the size of the Price Depth for one its sides. It is necessary to be subscribed to Price Depth in order for this function to return properly. To subscribe to Price Depth, call `SubscribePriceDepth`.

- `GetPriceGroup`

Name	Type	Description
a_pAssetID	PConnectorAssetIdentifier	Asset identifier that is subscribed to Price Depth
a_nSide	Byte	Book Side, 0 foy buy or 1 for sell
a_nPosition	Integer	Group price position
a_pPrice	PConnectorPriceGroup	Group price

If successful, this function returns a Price Depth Entry (aka. price group), with position 0 being the top of the book, during auction, this is the position the theoric price will be located. It is necessary to be subscribed to Price Depth in order for this function to return properly. To subscribe to Price Depth, call `SubscribePriceDepth`.

If the entry is a theoretical price entry, the price will be set to `-INF`, to obtain the actual therotical price, call `GetTheoreticalValues`.

- `GetTheoreticalValues`

Name	Type	Description
a_pAssetID	PConnectorAssetIdentifier	Asset identifier
a_dPrice	Double	Theoric Price
a_nQuantity	Int64	Theoric Quantity

During auction, this function returns the current theoric price and quantity. This function only works on assets that have been subscribed one way or another. Changes to the theoric values are notified in the function defined in the callback `SetTheoreticalPriceCallback`.

- `SetEnabledHistOrder`

This function is used to enable/disable the history and automatic order updates when starting the application (1 = Enable / 0 = Disable). When the history is disabled, the application will not automatically receive order data at startup, and calls such as GetPosition, which require the position to be built using operations, will not return valid results. To disable automatic updates, this function should be called immediately after the initialization functions. It is important to note that by disabling the history, position control will not be calculated correctly by the platform, and the functionalities for zeroing positions and order status may be compromised. The user should be aware of these risks before disabling the history..

3.2 Callbacks

This section describes how each callback function in the library should be declared and its purpose.

`Important:` Other DLL functions should not be used within a callback.

Callbacks are invoked from the ConnectorThread, meaning they run on a different thread than the main thread of the client program.

All callback functions must be declared using the `stdcall` calling convention (https://en.wikipedia.org/wiki/X86_calling_conventions). This applies to both 32-bit and 64-bit versions.

```
TStateCallback = procedure(nConnStateType : Integer; nResult : Integer) stdcall;

TProgressCallback = procedure(rAssetID : TAssetIDRec; nProgress : Integer) stdcall;

TNewTradeCallback = procedure(
    rAssetID     : TAssetIDRec;
    pwcDate      : PWideChar;
    nTradeNumber : Cardinal;
    dPrice       : Double;
    dVol         : Double;
    nQtd         : Integer;
    nBuyAgent    : Integer;
    nSellAgent   : Integer;
    nTradeType   : Integer;
    bEdit        : Char) stdcall;

TNewDailyCallback = procedure(
    rAssetID       : TAssetIDRec;
    pwcDate        : PWideChar;
    dOpen          : Double;
    dHigh          : Double;
    dLow           : Double;
    dClose         : Double;
    dVol           : Double;
    dAjuste        : Double;
    dMaxLimit      : Double;
    dMinLimit      : Double;
    dVolBuyer      : Double;
    dVolSeller     : Double;
    nQtd           : Integer;
    nNegocios      : Integer;
    nContratosOpen : Integer;
    nQtdBuyer      : Integer;
    nQtdSeller     : Integer;
    nNegBuyer      : Integer;
    nNegSeller     : Integer) stdcall;

TPriceBookCallback = procedure(
    rAssetID   : TAssetIDRec;
    nAction    : Integer;
    nPosition  : Integer;
    nSide      : Integer;
    nQtds      : Integer;
```

```
    nCount     : Integer;
    dPrice     : Double;
    pArraySell : Pointer;
    pArrayBuy  : Pointer) stdcall;

TPriceBookCallbackV2 = procedure(
    rAssetID   : TAssetIDRec;
    nAction    : Integer;
    nPosition  : Integer;
    nSide      : Integer;
    nQtds      : Int64;
    nCount     : Integer;
    dPrice     : Double;
    pArraySell : Pointer;
    pArrayBuy  : Pointer) stdcall;

TOfferBookCallback = procedure(
    rAssetID    : TAssetIDRec ;
    nAction     : Integer;
    nPosition   : Integer;
    Side        : Integer;
    nQtd        : Integer;
    nAgent      : Integer;
    nOfferID    : Int64;
    dPrice      : Double;
    bHasPrice   : Char;
    bHasQtd     : Char;
    bHasDate    : Char;
    bHasOfferID : Char;
    bHasAgent   : Char;
    pwcDate     : PWideChar;
    pArraySell  : Pointer
    pArrayBuy   : Pointer) stdcall;

TOfferBookCallbackV2 = procedure(
    rAssetID    : TAssetIDRec ;
    nAction     : Integer;
    nPosition   : Integer;
    Side        : Integer;
    nQtd        : Int64;
    nAgent      : Integer;
    nOfferID    : Int64;
    dPrice      : Double;
    bHasPrice   : Char;
    bHasQtd     : Char;
    bHasDate    : Char;
    bHasOfferID : Char;
    bHasAgent   : Char;
    pwcDate     : PWideChar;
    pArraySell  : Pointer
    pArrayBuy   : Pointer) stdcall;

TConnectorAssetPositionListCallback = procedure(
    AccountID : TConnectorAccountIdentifier;
    AssetID   : TConnectorAssetIdentifier;
    EventID   : Int64) stdcall; forward;
```

```
TAccountCallback = procedure(
    nCorretora            : Integer;
    CorretoraNomeCompleto : PWideChar;
    AccountID             : PWideChar
    NomeTitular           : PWideChar) stdcall; forward;

TBrokerAccountListCallback = procedure(
    BrokerID : Integer;
    Changed  : Cardinal); stdcall;

TBrokerSubAccountListCallback = procedure(
    BrokerID    : Integer;
    a_AccountID : TConnectorAccountIdentifier
); stdcall;

TOrderChangeCallback = procedure(
    rAssetID    : TAssetIDRec;
    nCorretora  : Integer;
    nQtd        : Integer;
    nTradedQtd  : Integer;
    nLeavesQtd  : Integer;
    nSide       : Integer;
    dPrice      : Double;
    dStopPrice  : Double;
    dAvgPrice   : Double;
    nProfitID   : Int64;
    TipoOrdem   : PWideChar;
    Conta       : PWideChar;
    Titular     : PWideChar;
    ClOrdID     : PWideChar;
    Status      : PWideChar;
    Date        : PWideChar;
    TextMessage : PWideChar) stdcall;

THistoryCallback = procedure(
    rAssetID   : TAssetIDRec;
    nCorretora : Integer;
    nQtd       : Integer;
    nTradedQtd : Integer;
    nLeavesQtd : Integer;
    nSide      : Integer;
    dPrice     : Double;
    dStopPrice : Double;
    dAvgPrice  : Double;
    nProfitID  : Int64;
    TipoOrdem  : PWideChar;
    Conta      : PWideChar;
    Titular    : PWideChar;
    ClOrdID    : PWideChar;
    Status     : PWideChar;
    Date       : PWideChar) stdcall;

THistoryTradeCallback = procedure(
    rAssetID     : TAssetIDRec;
    pwcDate      : PWideChar;
```

```
    nTradeNumber : Cardinal;
    dPrice       : Double;
    dVol         : Double;
    nQtd         : Integer;
    nBuyAgent    : Integer;
    nSellAgent   : Integer;
    nTradeType   : Integer) stdcall;

TTinyBookCallback = procedure(
    rAssetID : TAssetIDRec;
    dPrice   : Double;
    nQtd     : Integer;
    nSide    : Integer) stdcall;

TAssetListCallback = procedure(
    rAssetID : TAssetIDRec;
    pwcName  : PWideChar) stdcall;

TAssetListInfoCallback = procedure(
    rAssetID            : TAssetIDRec;
    pwcName             : PWideChar;
    pwcDescription      : PWideChar;
    nMinOrderQtd        : Integer;
    nMaxOrderQtd        : Integer;
    nLote               : Integer;
    stSecurityType      : Integer;
    ssSecuritySubType   : Integer;
    dMinPriceIncrement  : Double;
    dContractMultiplier : Double;
    strValidDate        : PWideChar;
    strISIN             : PWideChar) stdcall;

TAssetListInfoCallbackV2 = procedure(
    rAssetID            : TAssetIDRec;
    pwcName             : PWideChar;
    pwcDescription      : PWideChar;
    nMinOrderQtd        : Integer;
    nMaxOrderQtd        : Integer;
    nLote               : Integer;
    stSecurityType      : Integer;
    ssSecuritySubType   : Integer;
    dMinPriceIncrement  : Double;
    dContractMultiplier : Double;
    strValidDate        : PWideChar;
    strISIN             : PWideChar;
    strSetor            : PWideChar;
    strSubSetor         : PWideChar;
    strSegmento         : PWideChar) stdcall;

TChangeStateTicker = procedure(
    rAssetID : TAssetIDRec;
    pwcDate  : PWideChar;
    nState   : Integer) stdcall;

TInvalidTickerCallback = procedure(
    const AssetID : TConnectorAssetIdentifier
```

```
) stdcall;

TAdjustHistoryCallback = procedure(
    rAssetID      : TAssetIDRec;
    dValue        : Double;
    strAdjustType : PWideChar;
    strObserv     : PWideChar;
    dtAjuste      : PWideChar;
    dtDeliber     : PWideChar;
    dtPagamento   : PWideChar;
    nAffectPrice  : Integer) stdcall;

TAdjustHistoryCallbackV2 = procedure(
    rAssetID      : TAssetIDRec;
    dValue        : Double;
    strAdjustType : PwideChar;
    strObserv     : PwideChar;
    dtAjuste      : PwideChar;
    dtDeliber     : PwideChar;
    dtPagamento   : PwideChar;
    nFlags        : Cardinal;
    dMult         : Double) stdcall;

TTheoreticalPriceCallback = procedure(
    rAssetID          : TAssetIDRec;
    dTheoreticalPrice : Double;
    nTheoreticalQtd   : Int64) stdcall;

TChangeCotation = procedure(
    rAssetID     : TAssetIDRec;
    pwcDate      : PWideChar;
    nTradeNumber : Cardinal;
    dPrice       : Double) stdcall;

THistoryCallbackV2 = procedure(
    rAssetID     : TAssetIDRec;
    nCorretora   : Integer;
    nQtd         : Integer;
    nTradedQtd   : Integer;
    nLeavesQtd   : Integer;
    nSide        : Integer;
    nValidity    : Integer;
    dPrice       : Double;
    dStopPrice   : Double;
    dAvgPrice    : Double;
    nProfitID    : Int64;
    TipoOrdem    : PWideChar;
    Conta        : PWideChar;
    Titular      : PWideChar;
    ClOrdID      : PWideChar;
    Status       : PWideChar;
    LastUpdate   : PWideChar;
    CloseDate    : PWideChar;
    ValidityDate : PWideChar) stdcall;

TOrderChangeCallbackV2 = procedure(
```

```
    rAssetID     : TAssetIDRec;
    nCorretora   : Integer;
    nQtd         : Integer;
    nTradedQtd   : Integer;
    nLeavesQtd   : Integer;
    nSide        : Integer;
    nValidity    : Integer;
    dPrice       : Double;
    dStopPrice   : Double;
    dAvgPrice    : Double;
    nProfitID    : Int64;
    TipoOrdem    : PWideChar;
    Conta        : PWideChar;
    Titular      : PWideChar;
    ClOrdID      : PWideChar;
    Status       : PWideChar;
    LastUpdate   : PWideChar;
    CloseDate    : PWideChar;
    ValidityDate : PWideChar;
    TextMessage  : PWideChar) stdcall;

TConnectorOrderCallback = procedure(
    const a_OrderID : TConnectorOrderIdentifier
); stdcall;

TConnectorAccountCallback = procedure(
    const a_AccountID : TConnectorAccountIdentifier
); stdcall;

TConnectorTradeCallback = procedure(
    const a_Asset  : TConnectorAssetIdentifier;
    const a_pTrade : Pointer;
    const a_nFlags : Cardinal
); stdcall;
```

- `TStateCallback`

Corresponds to the callback to inform the login state, connection status, routing status, and product activation. According to the type of nConnStateType provided, which are:

```
CONNECTION_STATE_LOGIN        = 0;  // Connection to login server
CONNECTION_STATE_ROTEAMENTO   = 1;  // Connection to routing server
CONNECTION_STATE_MARKET_DATA  = 2;  // Connection to market data server
CONNECTION_STATE_MARKET_LOGIN = 3;  // Login to server market data

LOGIN_CONNECTED    = 0;   // Login server connected
LOGIN_INVALID      = 1;   // Login is invalid
LOGIN_INVALID_PASS = 2;   // Password is invalid
LOGIN_BLOCKED_PASS = 3;   // Password is blocked
LOGIN_EXPIRED_PASS = 4;   // Password has expired
LOGIN_UNKNOWN_ERR  = 200; // Internal login error
```

```
ROTEAMENTO_DISCONNECTED        = 0;
ROTEAMENTO_CONNECTING          = 1;
ROTEAMENTO_CONNECTED           = 2;
ROTEAMENTO_BROKER_DISCONNECTED = 3;
ROTEAMENTO_BROKER_CONNECTING   = 4;
ROTEAMENTO_BROKER_CONNECTED    = 5;

MARKET_DISCONNECTED = 0;  // Disconnected from the market data server
MARKET_CONNECTING   = 1;  // Connecting to the market data server
MARKET_WAITING      = 2;  // Waiting for connection
MARKET_NOT_LOGGED   = 3;  // Not logged into the market data server
MARKET_CONNECTED    = 4;  // Connected to the market data

CONNECTION_ACTIVATE_VALID   = 0;  // Valid activation
CONNECTION_ACTIVATE_INVALID = 1;  // Invalid activation
```

Given that the type `nConnStateType` received is one of the values of CONNECTION_STATE, and nResult is the login state of the specific service. The correct values for a valid connection are:

- nConnStateType = CONNECTION_STATE_LOGIN
  - nResult = LOGIN_CONNECTED
- nConnStateType = CONNECTION_STATE_ROTEAMENTO
  - nResult = ROTEAMENTO_CONNECTED
- nConnStateType = CONNECTION_STATE_MARKET_DATA
  - nResult = MARKET_CONNECTED
- nConnStateType = CONNECTION_STATE_MARKET_LOGIN
  - nResult = CONNECTION_ACTIVATE_VALID

- `TNewTradeCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the trade belongs
pwcDate	PWideChar	Trade date in the format DD/MM/YYYY HH:mm:SS.ZZZ (mm = minute, MM = month and ZZZ = millisecond)
nTradeNumber	Cardinal	Serial number of a trade
dPrice	Double	Execution price
dVol	Double	Financial volume
nQtd	Integer	Quantity
nBuyAgent	Integer	Buying agent
nSellAgent	Integer	Selling agent
nTradeType	Integer	Trade type
bEdit	Char	Indicates if it is an edit

Corresponds to the callback to inform a new trade, received after subscribing to this asset (according to the previously specified `SubscribeTicker` function). The `nTradeNumber` is the unique identifier of the trade per

trading session. `bEdit` indicates whether the received trade is an edit (information from the exchange) or an addition. The ID to identify an edited trade is `pwcDate`. `tradeType` indicates the type of trade according to the table below:"

```
1. Cross trade
2. Agressive Buy
3. Agressive Sell
4. Auction
5. Surveillance
6. Expit
7. Options Exercise
8. Over the counter
9. Derivative Term
10. Index
11. BTC
12. On Behalf
13. RLP
32. Unknown
```

- `TNewDailyCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the trade belongs
pwcDate	PWideChar	Trade date in the format DD/MM/YYYY HH:mm:SS.ZZZ (mm = minute, MM = month, and ZZZ = millisecond)
dOpen	Double	Trade price at market open
dHigh	Double	Highest price reached
dLow	Double	Lowest price reached
dClose	Double	Price of the last trade
dVol	Double	Financial volume
dAjuste	Double	Price adjustment
dMaxLimit	Double	Upper price limit for order
dMinLimit	Double	Lower price limit for order
dVolBuyer	Double	Volume of buyers
dVolSeller	Double	Volume of sellers
nQtd	Integer	Quantity
nNegocios	Integer	Total number of trades
nContratosOpen	Integer	Number of open contracts
nQtdBuyer	Integer	Number of buyers

Name	Type	Description
nQtdSeller	Integer	Number of sellers
nNegBuyer	Integer	Number of buyer trades
nNegSeller	Integer	Number of seller trades

Corresponds to the callback to provide a new quotation with aggregated information from the trading day.

- `TPriceBookCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nAction	Integer	Action to be performed on the market depth
nPosition	Integer	Position where the offer is to be inserted
nSide	Integer	Buy or sell (Buy=0, Sell=1)
nQtds	Integer	Quantity sold/bought
nCount	Integer	Number of offers sold/bought
dPrice	Double	Offered price
pArraySell	Pointer	Complete sell market depth
pArrayBuy	Pointer	Complete buy market depth

Deprecated: This callback has been replaced by `TConnectorPriceDepthCallback`.

Corresponds to the callback to provide an update in the price market depth. The parameters are valid or not according to the value of nAction, described below in detail:

- rAssetID: Ticker;
- nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);
- nPosition: Position in the grid; (Valid in atAdd, atEdit, atDelete and atDeleteFrom).
- Side: Buy or sell; (Always valid).
- nQtds: Quantity sold/bought; (Valid in atAdd and atEdit).
- nCount: Number of offers sold/bought; (Valid in atAdd and atEdit).
- dPrice: Price; (Valid in atAdd).

pArraySell, pArrayBuy: List of buy/sell offers; (Valid in atFullBook)..

This callback was designed to maintain separate lists of buy and sell offers. Therefore, each nAction received must be handled to modify these lists, depending on the side received in nSide, as described below. All adjustments depending on nPosition refer to the position from the end of the list (in lists starting at 0, size - nPosition - 1).

- atAdd: Insert a new offer after the position given by nPosition.
- atDelete: Delete an offer at the position given by nPosition.
- atDeleteFrom: Remove all offers starting from the position given by nPosition.
- atEdit: Update the offer information at the position given by nPosition.
- atFullBook: Create the market depth with all existing offers.

These details are received through the pArrayBuy and pArraySell parameters. For creating the list, when receiving atFullBook, both pArrayBuy and pArraySell arrays have the following layout in memory:

Header

Field	Type	Size	Offset
Number of offers (Q)	Integer	4 bytes	0
Array size (to be used in FreePointer)	Integer	4 bytes	4

Q entries to be inserted into the market depth, containing

Field	Type	Size	Offset
Price	Double	8 bytes	8
Quantity	Integer	4 bytes	16
Count	Integer	4 bytes	20

For more details on how to correctly assemble the market depth, refer to the examples in C++ and Delphi.

- `TPriceBookCallbackV2`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nAction	Integer	Action to be performed on the market depth
nPosition	Integer	Position where the offer is to be inserted
nSide	Integer	Buy or sell (Buy=0, Sell=1)
nQtds	Int64	Quantity sold/bought
nCount	Integer	Number of offers sold/bought
dPrice	Double	Offered price
pArraySell	Pointer	Complete sell market depth
pArrayBuy	Pointer	Complete buy market depth

Deprecated: This callback has been replaced by `TConnectorPriceDepthCallback`.

Corresponds to the callback to provide an update in the price market depth. The parameters are valid or not according to the value of nAction, described below in detail:

- rAssetID: Ticker;
- nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);
- nPosition: Position in the grid; (Valid in atAdd, atEdit, atDelete, and atDeleteFrom).
- Side: Buy or sell; (Always valid).
- nQtds: Quantity sold/bought; (Valid in atAdd and atEdit).
- nCount: Number of offers sold/bought; (Valid in atAdd and atEdit).
- dPrice: Price; (Valid in atAdd).

pArraySell, pArrayBuy: List of buy/sell offers; (Valid in atFullBook).

This callback was designed to maintain separate lists of buy and sell offers. Therefore, each nAction received must be handled to modify these lists, depending on the side received in nSide, as described below. All adjustments depending on nPosition refer to the position from the end of the list (in lists starting at 0, size - nPosition - 1).

- atAdd: Insert a new offer after the position given by nPosition.
- atDelete: Delete an offer at the position given by nPosition.
- atDeleteFrom: Remove all offers starting from the position given by nPosition.
- atEdit: Update the offer information at the position given by nPosition.
- atFullBook: Create the market depth with all existing offers.

These details are received through the pArrayBuy and pArraySell parameters. For creating the list, when receiving atFullBook, both pArrayBuy and pArraySell arrays have the following layout in memory:

Header

Field	Type	Size	Offset
Number of offers (Q)	Integer	4 bytes	0
Array size (to be used in FreePointer)	Integer	4 bytes	4

Q entries to be inserted into the market depth, containing

Field	Type	Size	Offset
Price	Double	8 bytes	8
Quantity	Int64	8 bytes	16
Count	Cardinal	4 bytes	24

For more details on how to correctly assemble the market depth, refer to the examples in C++ and Delphi.

- `TOfferBookCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nAction	Integer	Action to be performed on the market depth
nPosition	Integer	Position where the offer is to be inserted
nSide	Integer	Buy or sell (Buy=0, Sell=1)
nQtd	Integer	Quantity sold/bought
nAgent	Integer	Agent identifier
nOfferID	Integer	Offer identifier
dPrice	Double	Offered price
bHasPrice	Char	1 byte to specify if price exists
bHasQtd	Char	1 byte to specify if quantity exists

Name	Type	Description
bHasDate	Char	1 byte to specify if date exists
bHasOfferID	Char	1 byte to specify if offer ID exists
bHasAgent	Char	1 byte to specify if agent exists
pwcDate	PWideChar	Offer date in the format DD/MM/YYYY DD/MM/YYYY HH:mm:SS.ZZZ (mm = minute, MM = month e ZZZ = millisecond)
pArraySell	Pointer	Complete sell market depth
pArrayBuy	Pointer	Complete buy market depth

Corresponds to the callback to provide an update in the order book:

- rAssetID: Ticker; nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);
- nPosition: Position in the array; nSide: Order side (Buy=0, Sell=1);
- nQtd: Quantity sold/bought;
- nAgent: Indicates the IDs of the buying and selling agents, respectively; It is possible to obtain their names through the function GetAgentName already specified;

The callback is handled following the same specification as `TPriceBookCallback`, except for the layout of the pArrayBuy and pArraySell arrays:

Header

Field	Type	Size	Offset
`Q` Number of offers	Integer	4 bytes	0
Array size (to be used in FreePointer)	Integer	4 bytes	4

Array

`Q` entries to be inserted into the market depth, containing

Field	Type	Size	Offset
Price	Double	8 bytes	8
Quantity	Integer	4 bytes	16
Agent	Integer	4 bytes	20
OfferID	Int64	8 bytes	24
`T` string size for Date	Short	2 bytes	32
Offer date	Array of bytes	`T` bytes	34

Trailer

After `Q` entries of the market depth

Field	Type	Size	Offset
OfferBookFlags	Cardinal	4 bytes	(Q * 49) + 8

The `OfferBookFlags` field can contain the following flags

Flag	Value	Meaning
OB_LAST_PACKET	1	Indicates whether this is the last page of the market depth

- `TOfferBookCallbackV2`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nAction	Integer	Action to be performed on the market depth
nPosition	Integer	Position where the offer is to be inserted
nSide	Integer	Buy or sell (Buy=0, Sell=1)
nQtd	Int64	Quantity sold/bought
nAgent	Integer	Agent identifier
nOfferID	Integer	Offer identifier
dPrice	Double	Offered price
bHasPrice	Char	1 byte to specify if price exists
bHasQtd	Char	1 byte to specify if quantity exists
bHasDate	Char	1 byte to specify if date exists
bHasOfferID	Char	1 byte to specify if offer ID exists
bHasAgent	Char	1 byte to specify if agent exists
pwcDate	PWideChar	Offer date in the format DD/MM/YYYY HH:mm:SS.ZZZ (mm = minute, MM = month, ZZZ = millisecond)
pArraySell	Pointer	Complete sell market depth
pArrayBuy	Pointer	Complete buy market depth

Corresponds to the callback to provide an update in the market depth:

- rAssetID: Ticker; nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);
- nPosition: Position in the array; nSide: Order side (Buy=0, Sell=1);
- nQtd: Quantity sold/bought;
- nAgent: Indicates the IDs of the buying and selling agents, respectively; It is possible to obtain their names through the function GetAgentName already specified;

The callback is handled following the same specification as `TPriceBookCallbackV2`, except for the layout of the pArrayBuy and pArraySell arrays:

Header

Field	Type	Size	Offset
Q Number of offers	Integer	4 bytes	0
Array size (to be used in FreePointer)	Integer	4 bytes	4

Array

Q entries to be inserted into the market depth, containing

Field	Type	Size	Offset
Price	Double	8 bytes	8
Quantity	Int64	8 bytes	16
Agent	Integer	4 bytes	24
OfferID	Int64	8 bytes	28
T string size for Date	Short	2 bytes	36
Offer date	Array of bytes	T bytes	38

Trailer

After Q entries of the market depth

Field	Type	Size	Offset
OfferBookFlags	Cardinal	4 bytes	(Q * 53) + 8

The `OfferBookFlags` field can contain the following flags

Flag	Value	Meaning
OB_LAST_PACKET	1	Indicates whether this is the last page of the market depth

---

- `THistoryTradeCallback`

Name	Size	Description
rAssetID	TAssetIDRec	Asset to which the trade belongs
pwcDate	PWideChar	Trade date in the format DD/MM/YYYY HH:mm:SS.ZZZ (mm = minute, MM = month e ZZZ = millisecond)
nTradeNumber	Cardinal	Serial number of a trade
dPrice	Double	Execution price
dVol	Double	Financial volume
nQtd	Integer	Quantity

Name	Size	Description
nBuyAgent	Integer	Buying agent
nSellAgent	Integer	Selling agent
nTradeType	Integer	Trade type

Corresponds to the callback for trades that were requested via the `GetHistoryTrades` function.

- `TProgressCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the history request refers
nProgress	Integer	Progress value (0-100)

Corresponds to the progress callback of `THistoryTradeCallback`.

- `TTinyBookCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the offer belongs
dPrice	Double	Offer price
nQtd	Integer	Quantity
nSide	Integer	Buy or sell side (Buy=0, Sell=1)

Corresponds to the top of the market depth callback. rAssetID indicates which asset the offer belongs to according to the TAssetIDRec structure already specified. dPrice: Price; nQtd: Sell/Buy quantity; nSide: Order side (Buy=0, Sell=1).

The callbacks described below are only available after initialization using the `DLLInitializeLogin` function, therefore only for initialization with routing.

- `TAccountCallback`

Name	Type	Description
nCorretora	Integer	Broker identifier
CorretoraNomeCompleto	PWideChar	Full name of the broker
AccountID	PWideChar	Client account identification
NomeTitular	PWideChar	Account holder's name

Corresponds to the callback to provide information on existing accounts. It is possible to verify if the account is a simulation through the broker's name or identifier.

- `TOrderChangeCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nCorretora	Integer	Broker identifier
nQtd	Integer	Order quantity
nTradedQtd	Integer	Quantity already executed
nLeavesQtd	Integer	Quantity pending execution
nSide	Integer	Order side (Buy=1, Sell=2)
dPrice	Double	Order price
dStopPrice	Double	Stop price in case of a stop order
dAvgPrice	Double	Average execution price
nProfitID	Int64	Internal session identifier for the order
TipoOrdem	PWideChar	Order type
Conta	PWideChar	Account identifier
Titular	PWideChar	Account holder
ClOrdID	PWideChar	Unique order identifier (permanent)
Status	PWideChar	Order status
Date	PWideChar	Order execution date
TextMessage	PWideChar	Extra information message

Corresponds to the callback to inform order modifications sent by an account.

- `THistoryCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nCorretora	Integer	Broker identifier
nQtd	Integer	Order quantity
nTradedQtd	Integer	Quantity already executed
nLeavesQtd	Integer	Quantity pending execution
nSide	Integer	Order side (Buy=1, Sell=2)
dPrice	Double	Order price
dStopPrice	Double	Stop price in case of a stop order
dAvgPrice	Double	Average execution price
nProfitID	Int64	Internal session identifier for the order

Name	Type	Description
TipoOrdem	PWideChar	Order type
Conta	PWideChar	Account identifier
Titular	PWideChar	Account holder
ClOrdID	PWideChar	Unique order identifier (permanent)
Status	PWideChar	Order status
Date	PWideChar	Order execution date

Corresponds to the order history request callback. The history includes only orders from the current day.

- `TAssetListCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
pwcName	PWideChar	Asset description

Corresponds to the asset information request callback. It is necessary to use the `SetAssetListCallback` function for this callback to receive data.

- `TAssetListInfoCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the information belongs
pwcName	PWideChar	Asset name
pwcDescription	PWideChar	Asset description
nMinOrderQtd	Integer	Minimum allowed order quantity
nMaxOrderQtd	Integer	Maximum allowed order quantity
nLote	Integer	Lot size
stSecurityType	Integer	Asset Type *
ssSecuritySubType	Integer	Asset subtype **
dMinPriceIncrement	Double	Minimum price increment
dContractMultiplier	Double	Contract multiplier
strValidDate	PWideChar	Expiration date, if applicable
strISIN	PWideChar	Asset ISIN string

Corresponds to the asset information callback. The `stSecurityType` field represents the asset type returned, which can be one of the following:

```
* Asset Type
0. stFuture
1. stSpot
2. stSpotOption
3. stFutureOption
4. stDerivativeTerm
5. stStock
6. stOption
7. stForward
8. stETF
9. stIndex
10. stOptionExercise
11. stUnknown
12. stEconomicIndicator
13. stMultilegInstrument
14. stCommonStock
15. stPreferredStock
16. stSecurityLoan
17. stOptionOnIndex
18. stRights
19. stCorporateFixedIncome
255. stNelogicaSyntheticAsset
```

The `ssSecuritySubType` field is a specification within the type and can be one of the following:

```
** Asset Subtype
0. ssFXSpot
1. ssGold
2. ssIndex
3. ssInterestRate
4. ssFXRate
5. ssForeignDebt
6. ssAgricultural
7. ssEnergy
8. ssEconomicIndicator
9. ssStrategy
10. ssFutureOption
11. ssVolatility
12. ssSwap
13. ssMiniContract
14. ssFinancialRollOver
15. ssAgriculturalRollOver
16. ssCarbonCredit
17. ssUnknown
18. ssFractionary
19. ssStock
20. ssCurrency
21. ssOTC                       // OTC Over-the-Counter Market
22. ssFII                       // FII Real Estate Investment Fund
```

```
// PUMA 2.0 -Equities
23. ssOrdinaryRights          // DO
24. ssPreferredRights         // DP
25. ssCommonShares            // ON
26. ssPreferredShares         // PN
27. ssClassApreferredShares   // PNA
28. ssClassBpreferredShares   // PNB
29. ssClassCpreferredShares   // PNC
30. ssClassDpreferredShares   // PND
31. ssOrdinaryReceipts        // ON REC
32. ssPreferredReceipts       // PN REC
33. ssCommonForward
34. ssFlexibleForward
35. ssDollarForward
36. ssIndexPointsForward
37. ssNonTradeableETFIndex
38. ssPredefinedCoveredSpread
39. ssTraceableETF
40. ssNonTradeableIndex
41. ssUserDefinedSpread
42. ssExchangeDefinedspread   // Not currently used
43. ssSecurityLoan
44. ssTradeableIndex
45. ssOthers

46. ssBrazilianDepositaryReceipt       // BDR
47. ssFund
48. ssOtherReceipt
49. ssOtherRight
50. ssUNIT
51. ssClassEPreferredShare             // PNE
52. ssClassFPreferredShare             // PNF
53. ssClassGPreferredShare             // PNG
54. ssWarrant
55. ssNonTradableSecurityLending
56. ssForeignIndexETF
57. ssGovernmentETF
58. ssIpoOrFollowOn
59. ssGrossAuction
60. ssNetAuction
61. ssTradableIndexInPartnership
62. ssNontradableIndexInPartnership

63. ssFixedIncomeETF
64. ssNontradableFixedIncomeETF
65. ssOutrightPurchase
66. ssSpecificCollateralRepo
67. ssDebenture
68. ssRealStateReceivableCertificate
69. ssAgribusinessReceivableCertificate
70. ssPromissoryNote
71. ssLetraFinanceira
72. ssAmericanDepositaryReceipt
```

```
73. ssUnitInvestmentFund
74. ssReceivableInvestmentFund
75. ssOutrightTPlus1
76. ssRepoTPlus1
77. ssNonTradableGrossSettlement
78. ssNonTradableNetSettlement
79. ssETFPrimaryMarket
80. ssSharesPrimaryMarket
81. ssRightsPrimaryMarket
82. ssUnitPrimaryMarket
83. ssFundPrimaryMarket
84. ssForeignIndexETFPrimaryMarket
85. ssWarrantPrimaryMarket
86. ssReceiptPrimaryMarket
87. ssGermanPublicDebts
88. ssStockRollover

93. ssStrategySpotDollar
94. ssTargetRate
95. ssTradableETFRealState
96. ssNonTradableETFRealEstate
254. ssDefault
```

- `TAssetListInfoCallbackV2`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the information belongs
pwcName	PWideChar	Asset name
pwcDescription	PWideChar	Asset description
nMinOrderQtd	Integer	Minimum allowed order quantity
nMaxOrderQtd	Integer	Maximum allowed order quantity
nLote	Integer	Lot size
stSecurityType	Integer	Asset type *
ssSecuritySubType	Integer	Asset subtype **
dMinPriceIncrement	Double	Minimum price increment
dContractMultiplier	Double	Contract multiplier
strValidDate	PWideChar	Expiration date, if applicable
strISIN	PWideChar	Asset ISIN string
strSetor	PWideChar	activity sector
strSubSetor	PWideChar	Subsector within the sector
strSegmento	PWideChar	operating segment

Extension of the `TAssetListInfoCallback` callback, only adding the fields sector, subsector, and segment.

- `TInvalidTickerCallback`

Name	Type	Meaning
AssetID	TConnectorAssetIdentifier	Invalid asset

Corresponds to the callback for returning response to an invalid ticker request.

- `TTheoreticalPriceCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the information belongs
dTheoreticalPrice	Double	Theoretical price
nTheoreticalQtd	Int64	Theoretical quantity

Corresponds to the callback for returning the theoretical price and quantities during an asset auction.

- `TAdjustHistoryCallback`

Name	Type	Description
rAssetID	TAssetIDRec	Asset corresponding to the adjustment
dValue	Double	Adjustment value
strAdjustType	PWideChar	Adjustment type *
strObserv	PWideChar	Observation
dtAjuste	PWideChar	Adjustment date
dtDeliber	PWideChar	Deliberation date
dtPagamento	PWideChar	Payment date
nAffectPrice	Integer	Indicates whether it affects the price or not

Corresponds to the asset adjustment callback. To use this callback, it must be sent to the DLL through the `SetAdjustHistoryCallback` function. It is preferable to use the `SetAdjustHistoryCallbackV2` function, which provides a more detailed description of how to calculate the adjustment.

```
* Adjustment Type
'None'
'Unknown'
'JurosRF'
'Dividendo'
'Rendimento'
'Subscricao'
'Desdobramento'
'ResgateTotalRF'
'ResgateTotalRV'
```

```
'AmortizacaoRF'
'JurosCapProprio'
'SubsComRenuncia'
'Bonificacao'
'Grupamento'
'JuncaoSerie'
'Cisao'
'Unknown'
```

- `TAdjustHistoryCallbackV2`

Name	Type	Description
rAssetID	TAssetIDRec	Asset corresponding to the adjustment
dValue	Double	Adjustment value
strAdjustType	PWideChar	Adjustment type *
strObserv	PWideChar	Observation
dtAjuste	PWideChar	Adjustment date
dtDeliber	PWideChar	Deliberation date
dtPagamento	PWideChar	Payment date
nFlags	Cardinal	Sum flag (described below)
dMult	Double	Multiplier

Corresponds to the asset adjustment callback. To use this callback, it must be sent to the DLL through the `SetAdjustHistoryCallbackV2` function. `nFlags` is a bit field from b0 to b31, where bit 0 (least significant) indicates whether the adjustment affects the price, and bit 1 indicates if it is a sum adjustment. `dMult` is the pre-computed value that should be multiplied by the price to apply the adjustment. It is only used if the adjustment is not a sum adjustment and affects the price, as indicated in the `nFlags` field. A `dMult` value of -9999 indicates that it is invalid and should not be used. If `dMult` is invalid, dValue is used for the calculation, where subtraction is performed for sum adjustments and division otherwise.

To calculate the adjustment, the parameters are used as follows:

- When `dMult` is a valid value, the adjustment is made by multiplying the price by this value.
- When the sum flag is set, the adjustment value is subtracted from the price
- When the sum flag is not set, the price is divided by the adjustment value.

Pseudocode:

```
while Date < AdjustmentDate if nFlag AND 1 and
(type is different from Reverse Split, Merger, Split and not(Unknown and not(nFlag
AND 2))) or
(type is different from Reverse Split, Merger, Split and not(nFlag AND 2))
then
  if dMult <> -9999
    Result := Result * dMult
```

```
else
  if (nFlag AND 2)
    Price := Price - AdjustmentValue
  else
    Price := Price / AdjustmentValue
```

- TChangeCotation

Name	Type	Description
rAssetID	TAssetIDRec	Asset in which the price change occurred
pwcDate	PWideChar	Date of the price change
nTradeNumber	Cardinal	Sequential number of the trade in which the change occurred
dPrice	Double	New price

This callback is used to report when a price modification occurs for the asset, providing the last price and time of the trade. To use this callback, it must be sent to the DLL through the SetChangeCotationCallback function.

- TChangeStateTicker

Name	Type	Description
rAssetID	TAssetIDRec	Asset in which the state change occurred
pwcDate	PWideChar	Date of the state change
nState	Integer	Asset state

Corresponds to the callback for identifying an asset's state change. The provided date is when the state modification occurred, but only some states show the date. The possible states are listed below:

```
0. tcsOpened       // Asset in open trading
2. tcsFrozen
3. tcsInhibited
4. tcsAuctioned    // Asset in auction
6. tcsClosed       // Asset with closed trading
10. tcsPreClosing
13. tcsPreOpening
```

- THistoryCallbackV2

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nCorretora	Integer	Broker identifier
nQtd	Integer	Order quantity
nTradedQtd	Integer	Quantity already executed

Name	Type	Description
nLeavesQtd	Integer	Quantity pending execution
nSide	Integer	Order side (Buy=1, Sell=2)
nValidity	Integer	Order validity type*
dPrice	Double	Order price
dStopPrice	Double	Stop price in case of stop order
dAvgPrice	Double	Average execution price
nProfitID	Int64	Internal session identifier for the order
TipoOrdem	PWideChar	Order type
Conta	PWideChar	Account identifier
Titular	PWideChar	Account holder
ClOrdID	PWideChar	Unique order identifier (permanent)
Status	PWideChar	Order status
LastUpdate	PWideChar	Date of the last order update
CloseDate	PWideChar	Order close date, if already closed
ValidityDate	PWideChar	Reference date for order validity

Corresponds to the secondary (optional) callback for the order history request. To use this callback, it must be sent to the DLL through the `SetHistoryCallbackV2` function, and it will be called under the same conditions as `THistoryCallback`. The history corresponds only to orders from the current day. The `nValidity` field represents the order validity type, which can be one of the following values:

```
* Order validity type
0. btfDay
1. btfGoodTillCancel
2. btfAtTheOpening
3. btfImmediateOrCancel
4. btfFillOrKill
5. btfGoodTillCrossing
6. btfGoodTillDate
7. btfAtTheClose
201. btfGoodForAuction
200. btfUnknown
```

- `TOrderChangeCallbackV2`

Name	Type	Description
rAssetID	TAssetIDRec	Asset to which the market depth belongs
nCorretora	Integer	Broker identifier

Name	Type	Description
nQtd	Integer	Order quantity
nTradedQtd	Integer	Quantity already executed
nLeavesQtd	Integer	Quantity pending execution
nSide	Integer	Order side (Buy=1, Sell=2)
nValidity	Integer	Order validity type
dPrice	Double	Order price
dStopPrice	Double	Stop price in case of stop order
dAvgPrice	Double	Average execution price
nProfitID	Int64	Internal session identifier for the order
TipoOrdem	PWideChar	Order type
Conta	PWideChar	Account identifier
Titular	PWideChar	Account holder
ClOrdID	PWideChar	Unique order identifier (permanent)
Status	PWideChar	Order status
LastUpdate	PWideChar	Date of the last order update
CloseDate	PWideChar	Order close date, if already closed
ValidityDate	PWideChar	Reference date for order validity
TextMessage	PWideChar	Extra information message

Corresponds to the secondary (optional) callback for reporting order modifications sent by an account. To use this callback, it must be sent to the DLL through the `SetOrderChangeCallbackV2` function, and it will be called under the same conditions as `TOrderChangeCallback`. The `nValidity` field represents the order validity type, and the possible values can be checked in the `THistoryCallbackV2` documentation.

- `TBrokerAccountListCallback`

Name	Type	Description
nCorretora	Integer	Broker identifier
Changed	Cardinal	Account change status

Corresponds to the callback to provide information on existing accounts available to use.

- `TBrokerSubAccountListCallback`

Name	Type	Description
nCorretora	Integer	Broker identifier
AccountID	TConnectorAccountIdentifier	Account identifier

Corresponds to the callback to provide information on existing sub-accounts available to use.

---

4. Product Usage

Initializing with Routing

To use the library, it is essential to initialize the services through the initialization functions. More specifically, if routing services are to be used, the `DLLInitializeLogin` function must be used, which will establish a connection to the routing and market data servers.

This function is described in the exposed functions section and requires an activation code provided at the time of product purchase, as well as a username and password to log into the authentication server. The other parameters are mandatory callbacks that will be called by the DLL during use and need to be specified at the time of initialization.

It is important to note that all callbacks occur in a thread called ConnectorThread and, therefore, happen simultaneously with the client application. The client application should process the data provided through the callbacks as data to be consumed from another thread. If necessary, the handling of writing this data should be done with critical sections or mutexes.

The data received via callbacks is stored in a single data queue, so any lengthy processing within the callback functions may delay the internal message processing queue of the DLL and cause delays in receiving trades or other information. To avoid this, the data should be processed and passed to other application threads immediately, or perform the minimum processing possible. Database accesses or disk writes should be avoided during the processing of a callback.

Finally, it is important to note that callbacks are designed only to receive data. Therefore, request functions to the DLL or any other function from the DLL interface should not be called within a callback, as this may cause unexpected exceptions and undefined behavior.

More implementation details can be clarified in the provided examples.

Initializing with Market Data

The Market Data initialization process is analogous to the Routing initialization, with the difference being the initialization function DLLInitializeMarketLogin and a reduction in the callbacks passed as parameters, as they are related to routing orders or accounts.

Data Types

All types mentioned in this document are types specified in the Delphi language. Below are some links for conversion or mapping of these types to the languages used in the examples.

- Delphi to C Type Mapping
    - https://docwiki.embarcadero.com/RADStudio/Tokyo/en/Delphi_to_C%2B%2B_types_mapping
- C Type to Python Conversion
    - https://docs.python.org/2/library/ctypes.html
- Delphi to C# Type Conversion

    - http://www.netcoole.com/delphi2cs/datatype.htm

32-bit Linkage

To use the library in 32-bit mode, the application must also be compiled in 32 bits. Since it operates in 32 bits, there is a 4GB memory limitation, which will be shared between the library and the client application. Therefore, it is not recommended to request large amounts of data in a single request, as this could exceed the process's memory limit.

- C#
    - Using Visual Studio, it is necessary to change the target platform in the Configuration Manager from `Any CPU` to `x86`.
- Python
    - The `python.exe` interpreter must also be 32-bit. Additionally, there is a bug in 32-bit Python where a callback containing a type larger than 32 bits fails and causes an exception. Follow the issue here: https://bugs.python.org/issue41021. Therefore, we recommend that clients who wish to use 32-bit Python use version 3.6.2, which was tested by the Nelogica team and does not have this issue.

For other languages, it is only necessary to switch the compilation mode to 32 bits.

64-bit Linkage

To use the library in 64-bit mode, the application must also be compiled in 64 bits. The calling convention remains stdcall, just like in the 32-bit version. There are no known issues with the example languages in the 64-bit version, so there is no recommended version; the latest versions of each language can be used.

The 64-bit version does not have a memory limitation and can therefore use the maximum available memory in the system, allowing larger data requests in a single request, limited by the amount of available RAM.