ProfitDLL 64 bits - Manual de Uso

# 4.0.0.30

Adicionado o campo AccountType nas seguintes estrututras:

 TConnectorTradingAccountOut

# 4.0.0.28

## Ativos da posição

Adicionado mecânismo para iterar sobre os ativos que compõe a posição de uma conta, além do mecânismo para vincular ordens a um evento de posição.

### Estruturas modificadas

Adicionado o campo EventID nas seguintes estruturas:

 TConnectorTradingAccountPosition  TConnectorOrder

 TConnectorOrderOut

### Tipos novos

 TConnectorAssetPositionListCallback

### Callbacks novas

 SetAssetPositionListCallback

### Funções novas

 EnumerateAllPositionAssets

## Lista de contas e subcontas

Adicionado novos mecânimos para a recuperação das contas e subcontas de roteamento.

### Tipos novos

 TBrokerAccountListCallback

 TBrokerSubAccountListCallback

### Callbacks novas

 SetBrokerAccountListChangedCallback

 SetBrokerSubAccountListChangedCallback

### Funções novas

 GetAccountCountByBroker  GetAccountsByBroker

## Bug fixes

 Ajustada função GetAccounts para quando tamanho era maior que a lista de contas.

# 4.0.0.24

## Nome dos agentes

Novas funções para recuperar o nome dos agentes.

### Funções novas

 GetAgentNameLength  GetAgentName

## Bug fixes

 Ajustado código de erro de GetPositionV2 e SendZeroPositionV2 para quando não tem posição;  Retirada limitação de tamanho dos campos Ticker;

# 4.0.0.21

## Bug fixes

 Corrigida notificação duplicadas de ordem quando não existe nenhuma alteração

# 4.0.0.20

## Callbacks de trade

Adicionado novo mecânismos para recuperação de trades

### Tipos novos

 TConnectorTradeCallback  TConnectorTrade

### Callbacks novas

 SetTradeCallbackV2

 SetHistoryTradeCallbackV2

### Funções novas

 TranslateTrade

## Histórico de ordem

Em virtude do download de histórico de ordens, foi criado um mecânismo aprimorado para a recuperação do histórico de ordens de uma conta.

### Tipos novos

 TConnectorOrder

 TConnectorEnumerateOrdersProc

### Callbacks novas

 SetOrderHistoryCallback

### Funções novas

 HasOrdersInInterval

 EnumerateOrdersByInterval  EnumerateAllOrders

## Bug fixes

 Adicionada flag nas callbacks de livro de oferta (ambas SetOfferBookCallback e

SetOfferBookCallbackV2)

# Descrição do Produto

Os arquivos contidos no arquivo zip estão organizados em diretórios separados para as versões de 64 bits e 32 bits. Cada diretório possui a mesma estrutura de organização de arquivos. No diretório denominado DLL e Executável, é possível encontrar o arquivo ProfitDLL.dll para a versão de 32 bits e ProfitDLL64.dll para a versão de 64 bits. Além disso, há um exemplo compilado em Delphi que pode ser utilizado para validar as funcionalidades do software. Já no diretório denominado Interface, são disponibilizados arquivos contendo as declarações das funções e tipos necessários para realizar a comunicação com a DLL em Delphi.

Existem também exemplos para 4 linguagens de programação diferentes nas pastas Exemplo.

 Delphi  C#

 C++

 Python

Elas contém o código fonte para utilizar as principais funcionalidades do produto.

# Descrição da Biblioteca

A biblioteca possui funções básicas de comunicação com os servidores de Roteamento e Market Data para o desenvolvimento de aplicações 32 ou 64 bits. A DLL responde eventos dos servidores e os envia processados em tempo real para a aplicação cliente, principalmente por meio de callbacks que serão descritos na seção 3.2.

As seções a seguir descrevem, em mais detalhes, como a comunicação entre a biblioteca e a aplicação cliente é realizada, bem como apresentam os detalhes técnicos de cada função ou callback.

# Interface da Biblioteca

A biblioteca expõe diversas funções chamadas diretamente pela aplicação cliente que realizam requisições para os servidores ou diretamente para os serviços e estruturas internas da DLL. Os tipos especificados nesta

documentação estão codificados em Delphi, com exemplos específicos para outras linguagens de programação em seus respectivos arquivos de exemplo.

Todas as estruturas necessárias para definir as funções da biblioteca são definidas a seguir:

Definições:

TAssetIDRec = packed record

pwcTicker : PWideChar; // Representa o nome do ativo ex.: "WDOFUT".

pwcBolsa : PWideChar; // Representa a bolsa que o ativo pertence ex. (para Bovespa): "B".

nFeed : Integer; // Fonte dos dados 0 (Nelogica), 255 (Outro).

end;

PAssetIDRec = ^TAssetIDRec;

TAccountRec = packed record

pwhAccountID : PWideChar; // Identificador da conta

pwhTitular : PWideChar; // Nome do titular da conta pwhNomeCorretora : PWideChar; // Nome da corretora

nCorretoraID : Integer; // Identificador da corretora

end;

PAccountRec = ^TAccountRec;

// 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 |
| ); |  |  |  |

TConnectorAccountIdentifier = record Version : Byte;

end;

// V0

BrokerID : Integer;

AccountID : PWideChar; SubAccountID : PWideChar; Reserved : Int64;

PConnectorAccountIdentifier = ^TConnectorAccountIdentifier;

TConnectorAccountIdentifierOut = record Version : Byte;

end;

// V0

BrokerID : Integer;

AccountID : TString0In; AccountIDLength : Integer;

SubAccountID : TString0In; SubAccountIDLength : Integer;

Reserved : Int64;

PConnectorAccountIdentifierOut = ^TConnectorAccountIdentifierOut;

TConnectorAssetIdentifier = record Version : Byte;

end;

// V0

Ticker : PWideChar; Exchange : PWideChar; FeedType : Byte;

TConnectorAssetIdentifierOut = record Version : Byte;

end;

// V0

Ticker : PWideChar; TickerLength : Integer;

Exchange : PWideChar; ExchangeLength : Integer;

FeedType : Byte;

TConnectorOrderIdentifier = record Version : Byte;

end;

// V0

LocalOrderID : Int64;

ClOrderID : PWideChar;

TConnectorSendOrder = record Version : Byte;

// V0

AccountID : TConnectorAccountIdentifier; AssetID : TConnectorAssetIdentifier;

Password : PWideChar; OrderType : Byte;

OrderSide : Byte;

end;

Price : Double; StopPrice : Double; Quantity : Int64;

PConnectorSendOrder = ^TConnectorSendOrder;

TConnectorChangeOrder = record Version : Byte;

// V0

AccountID : TConnectorAccountIdentifier; OrderID : TConnectorOrderIdentifier;

Password : PWideChar;

end;

Price : Double; StopPrice : Double; Quantity : Int64;

PConnectorChangeOrder = ^TConnectorChangeOrder;

TConnectorCancelOrder = record Version : Byte;

end;

// V0

AccountID : TConnectorAccountIdentifier; OrderID : TConnectorOrderIdentifier;

Password : PWideChar;

PConnectorCancelOrder = ^TConnectorCancelOrder;

TConnectorCancelOrders = record Version : Byte;

// V0

AccountID : TConnectorAccountIdentifier; AssetID : TConnectorAssetIdentifier;

Password : PWideChar;

end;

PConnectorCancelOrders = ^TConnectorCancelOrders;

TConnectorCancelAllOrders = record Version : Byte;

end;

// V0

AccountID : TConnectorAccountIdentifier; Password : PWideChar;

PConnectorCancelAllOrders = ^TConnectorCancelAllOrders;

TConnectorZeroPosition = record Version : Byte;

// V0

AccountID : TConnectorAccountIdentifier; AssetID : TConnectorAssetIdentifier;

Password : PWideChar; Price : Double;

end;

// V1

PositionType : Byte;

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;

end;

// V2

EventID : Int64;

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;

// Delegates

TConnectorEnumerateOrdersProc = function( const a\_Order : PConnectorOrder;

const a\_Param : LPARAM

) : BOOL; stdcall;

// TConnectorTradingAccountOut.Flags CA\_IS\_SUB\_ACCOUNT : TFlags = 1;

CA\_IS\_ENABLED : TFlags = 2;

// TConnectorMarketDataLibrary.Flags CM\_IS\_SHORT\_NAME : TFlags = 1;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| // Bolsas |  | | | |
| 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; // Conexão com servidor de login

CONNECTION\_STATE\_ROTEAMENTO = 1; // Conexão com servidor de roteamento CONNECTION\_STATE\_MARKET\_DATA = 2; // Conexão com servidor de market data CONNECTION\_STATE\_MARKET\_LOGIN = 3; // Login com servidor market data

LOGIN\_CONNECTED = 0; // Servidor de login conectado LOGIN\_INVALID = 1; // Login é inválido

LOGIN\_INVALID\_PASS = 2; // Senha inválida LOGIN\_BLOCKED\_PASS = 3; // Senha bloqueada LOGIN\_EXPIRED\_PASS = 4; // Senha expirada

LOGIN\_UNKNOWN\_ERR = 200; // Erro interno de login

ROTEAMENTO\_DISCONNECTED = 0;

ROTEAMENTO\_CONNECTING = 1;

ROTEAMENTO\_CONNECTED = 2;

ROTEAMENTO\_BROKER\_DISCONNECTED = 3;

ROTEAMENTO\_BROKER\_CONNECTING = 4;

ROTEAMENTO\_BROKER\_CONNECTED = 5;

MARKET\_DISCONNECTED = 0; // Desconectado do servidor de market data MARKET\_CONNECTING = 1; // Conectando ao servidor de market data

MARKET\_WAITING = 2; // Esperando conexão

MARKET\_NOT\_LOGGED = 3; // Não logado ao servidor de market data MARKET\_CONNECTED = 4; // Conectado ao market data

CONNECTION\_ACTIVATE\_VALID = 0; // Ativação válida

CONNECTION\_ACTIVATE\_INVALID = 1; // Ativação inválida

// Versões Antigas

TConnectorOrderTypeV0 = ( cotLimit = 0,

cotStop = 1,

cotMarket = 2

);

TConnectorOrderSideV0 = ( cosBuy = 0,

cosSell = 1

);

// TConnectorTradeCallback.Flags TC\_IS\_EDIT : Cardinal = 1;

TC\_LAST\_PACKET : Cardinal = 2;

Códigos de erro:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | **Valor (Hex)** | **Valor (Dec)** | **Significado** |
| NL\_OK | 0 | 0 | Sucesso |
| NL\_INTERNAL\_ERROR | 0x80000001 | -2147483647 | Erro interno |
| NL\_NOT\_INITIALIZED | 0x80000002 | -2147483646 | Não inicializado |
| NL\_INVALID\_ARGS | 0x80000003 | -2147483645 | Agumentos inválidos |
| NL\_WAITING\_SERVER | 0x80000004 | -2147483644 | Aguardando dados do servidor |
| NL\_NO\_LOGIN | 0x80000005 | -2147483643 | Nenhum login encontrado |
| NL\_NO\_LICENSE | 0x80000006 | -2147483642 | Nenhuma licença encontrada |
| NL\_OUT\_OF\_RANGE | 0x80000009 | -2147483639 | Count do parâmetro maior que o tamanho do array |
| NL\_MARKET\_ONLY | 0x8000000A | -2147483638 | Não possui roteamento |
| NL\_NO\_POSITION | 0x8000000B | -2147483637 | Não possui posição |
| NL\_NOT\_FOUND | 0x8000000C | -2147483636 | Recurso não encontrado |
| NL\_VERSION\_NOT\_SUPPORTED | 0x8000000D | -2147483635 | Versão do recurso não suportada |
| NL\_OCO\_NO\_RULES | 0x8000000E | -2147483634 | OCO sem nenhuma regra |
| NL\_EXCHANGE\_UNKNOWN | 0x8000000F | -2147483633 | Bolsa desconhecida |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | **Valor (Hex)** | **Valor (Dec)** | **Significado** |
| NL\_NO\_OCO\_DEFINED | 0x80000010 | -2147483632 | Nenhuma OCO encontrada para a ordem |
| NL\_INVALID\_SERIE | 0x80000011 | -2147483631 | (Level + Offset + Factor) inválido |
| NL\_LICENSE\_NOT\_ALLOWED | 0x80000012 | -2147483630 | Recurso não liberado na licença |
| NL\_NOT\_HARD\_LOGOUT | 0x80000013 | -2147483629 | Retorna que não esta em HardLogout |
| NL\_SERIE\_NO\_HISTORY | 0x80000014 | -2147483628 | Série não tem histórico no servidor |
| NL\_ASSET\_NO\_DATA | 0x80000015 | -2147483627 | Asset não tem o TData carregado |
| NL\_SERIE\_NO\_DATA | 0x80000016 | -2147483626 | Série não tem dados (count = 0) |
| NL\_HAS\_STRATEGY\_RUNNING | 0x80000017 | -2147483625 | Existe uma estratégia rodando |
| NL\_SERIE\_NO\_MORE\_HISTORY | 0x80000018 | -2147483624 | Não tem mais dados disponiveis para a serie |
| NL\_SERIE\_MAX\_COUNT | 0x80000019 | -2147483623 | Série esta no limite de dados possíveis |
| NL\_DUPLICATE\_RESOURCE | 0x8000001A | -2147483622 | Recurso duplicado |
| NL\_UNSIGNED\_CONTRACT | 0x8000001B | -2147483621 | Contrato não assinado |
| NL\_NO\_PASSWORD | 0x8000001C | -2147483620 | Nenhuma senha informada |
| NL\_NO\_USER | 0x8000001D | -2147483619 | Nenhum usuário informado no login |
| NL\_FILE\_ALREADY\_EXISTS | 0x8000001E | -2147483618 | Arquivo já existe |
| NL\_INVALID\_TICKER | 0x8000001F | -2147483617 | Ativo é inválido |
| NL\_NOT\_MASTER\_ACCOUNT | 0x80000020 | -2147483616 | Conta não é master |

## Funções expostas

As declarações de todas as funções expostas se encontram nesta seção. Algumas funções recebem tipos contendo callback no nome, estas serão descritas na próxima subseção.

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 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

### Nome Tipo Descrição

const pwcActivationKey

PWideChar Chave de ativação fornecida para login

|  |  |  |
| --- | --- | --- |
| const pwcUser | PWideChar | Usuário para login da conta correspondente à chave de ativação |
| const pwcPassword | PWideChar | Senha de login |
| StateCallback | TStateCallback | Callback de estado da conexão |
| HistoryCallback | THistoryCallback | Callback de histórico de ordens |
| OrderChangeCallback | TOrderChangeCallback | Callback de mudança no estado de uma ordem |
| AccountCallback | TAccountCallback | Callback de informações da conta de roteamento |
| NewTradeCallback | TNewTradeCallback | Callback de trades em tempo real |
| NewDailyCallback | TNewDailyCallback | Callback de dados diários agregados |
| PriceBookCallback | TPriceBookCallback | Callback de informações do livro de preços |
| OfferBookCallback | TOfferBookCallback | Callback de informações do livro de ofertas |
| HistoryTradeCallback | THistoryTradeCallback | Callback de dados de histórico de trades |

ProgressCallback TProgressCallback Callback de progresso de alguma requisição de

histórico

TinyBookCallback TTinyBookCallback Callback de topo de livro

Função de inicialização dos serviços de Market Data e Roteamento da DLL. Ela irá inicializar conexão com todos servidores e criar os serviços necessários para comunicação. Outras funções podem retornar o status de erro

NL\_ERR\_INIT caso DLLInitializeLogin não seja bem sucedida.

DLLInitializeMarketLogin

### Nome Tipo Descrição

const pwcActivationKey

PWideChar Chave de ativação fornecida para login

|  |  |  |
| --- | --- | --- |
| const pwcUser | PWideChar | Usuário para login da conta correspondente à chave de ativação |
| const pwcPassword | PWideChar | Senha de login |
| StateCallback | TStateCallback | Callback de estado da conexão |
| NewTradeCallback | TNewTradeCallback | Callback de trades em tempo real |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| NewDailyCallback | TNewDailyCallback | Callback de dados diários agregados |
| PriceBookCallback | TPriceBookCallback | Callback de informações do livro de preços |
| OfferBookCallback | TOfferBookCallback | Callback de informações do livro de ofertas |
| HistoryTradeCallback | THistoryTradeCallback | Callback de dados de histórico de trades |

ProgressCallback TProgressCallback Callback de progresso de alguma requisição de

histórico

TinyBookCallback TTinyBookCallback Callback de topo de livro

Equivalente à função DLLInitializeLogin, porém inicializa somente serviços de Market Data.

 DLLFinalize

Função utilizada para finalização dos serviços da DLL.

 SetServerAndPort

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const strServer | Double | Endereço do servidor de Market Data |
| const strPort | Integer | Porta do servidor de Market Data |

É usado para conectar em servidores específicos do Market Data, precisa ser chamado antes da inicialização (DLLInitialize ou InitializeMarket).

Importante: apenas utilizar essa função com orientação da equipe de desenvolvimento, a DLL funciona da melhor maneira escolhendo os servidores internamente

 GetServerClock

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| var dtDate | Double | Data codificada como Double |
| var nYear | Integer | Ano |
| var nMonth | Integer | Mês |
| var nDay | Integer | Dia |
| var nHour | Integer | Hora |
| var nMin | Integer | Minuto |
| var nSec | Integer | Segundo |
| var nMilisec | Integer | Milissegundo |

Retorna o horário do servidor de Market Data, pode ser chamado somente após inicialização. O parâmetro dtDate corresponde a uma referência para Double que segue o padrão TDateTime do Delphi, descrito em [http://docwiki.embarcadero.com/Libraries/Sydney/en/System.TDateTime.](http://docwiki.embarcadero.com/Libraries/Sydney/en/System.TDateTime) Os outros parâmetros também são passados por referência ao caller e somente representam os valores de data calendário do valor codificado no parâmetro dtDate.

GetLastDailyClose

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| var dClose | Double | Valor retornado de fechamento da última sessão |
| bAdjusted | Integer | Indica se deve ajustar o preço |

A função retorna o valor do fechamento (dClose) do candle anterior ao dia atual, de acordo com o parâmetro bAdjusted. Se bAdjusted for 0, o valor não ajustado é retornado; caso contrário, o valor ajustado é retornado. Para que a função retorne NL\_OK com dados, é necessário que SubscribeTicker tenha sido previamente chamada para o mesmo ativo. Na primeira chamada da função, os dados são requisitados ao servidor e a função retorna NL\_WAITING\_SERVER. Todas as chamadas subsequentes para o mesmo ativo retornam os dados já carregados.

Ativos inválidos retornam NL\_ERR\_INVALID\_ARGS. Se os dados da série diária ou ajustes não estiverem previamente carregados, essa chamada irá carregá-los e, consequentemente, disparar os callbacks progressCallback e adjustHistoryCallback.

 SubscribeTicker

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

É usado para receber as cotações em tempo real de determinado ativo. As informações são recebidas posteriormente à inscrição assim que disponíveis pelo callback especificado no parâmetro NewTradeCallback da função de inicialização. Em caso de requisção de ticker inválido, um evento vai ser disparado na callback definida SetInvalidTickerCallback. UnsubscribeTicker desativa este serviço.

 UnsubscribeTicker

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

Solicita ao serviço de Market Data que interrompa o envio de cotações em tempo real de um determinado ativo.

 SubscribeOfferBook

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

É usado para receber informações do livro de ofertas em tempo real. As informações são recebidas posteriormente à inscrição assim que disponíveis pelo callback especificado no parâmetro OfferBookCallback

da função de inicialização. Em caso de requisção de ticker inválido, um evento vai ser disparado na callback definida SetInvalidTickerCallback. UnsubscribeOfferBook desativa esse serviço.

 UnsubscribeOfferBook

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

Solicita ao serviço de Market Data que interrompa o envio em tempo real do livro de ofertas de um determinado ativo.

 SubscribePriceBook

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

É usado para receber informações do livro de preços em tempo real. As informações são recebidas posteriormente à inscrição assim que disponíveis pelo callback especificado no parâmetro PriceBookCallback da função de inicialização. Em caso de requisção de ticker inválido, um evento vai ser disparado na callback definida SetInvalidTickerCallback. UnsubscribePriceBook desativa esse serviço.

 UnsubscribePriceBook

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

Solicita ao serviço de Market Data que interrompa o envio do livro de preços em tempo real de um determinado ativo.

As chamadas de Subscribe e Unsubscribe SubscribeTicker, UnsubscribeTicker, SubscribePriceBook, UnsubscribePriceBook, SubscribeOfferBook, UnsubscribeOfferBook recebe os seus parâmetros no seguinte padrão:

Ticker: PETR4, Bolsa: B Ticker: WINFUT, Bolsa: F

Mais exemplos de bolsas podem ser encontradas na seção de declarações.

 SubscribeAdjustHistory

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

É utilizado para receber histórico de ajustes do ativo determinado ticker. É necessário fornecer a função de callback SetAdjustHistoryCallback ou SetAdjustHistoryCallbackV2 para utilizar esse subscribe. Em caso de requisção de ticker inválido, um evento vai ser disparado na callback definida SetInvalidTickerCallback.

 UnsubscribeAdjustHistory

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

Solicita ao serviço de Market Data que interrompa o envio de informações de ajustes de um determinado ativo.

 GetAgentNameByID e GetAgentShortNameByID

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nID | Integer | Identificador do agente negociante |

O valor retornado apresenta o nome completo e abreviado, respectivamente, deste agente.

**Depreciada**: Utilizada a função GetAgentName junto a GetAgentNameLength para buscar o nome do agente.

 GetAgentNameLength

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nID | Integer | Identificador do agente negociante |
| nShortFlag | Cardinal | Seta a busca pelo nome completo ou abreviado |

O valor retornado representa o tamanho do nome do agente, sendo o nome completo ou abreviado.

GetAgentName

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nAgentLength | Integer | Tamanho da string do nome |
| nID | Integer | Identificador do agente negociante |
| pwcAgent | PWideChar | Ponteiro que receberá o nome do agente. |
| nShortFlag | Cardinal | Seta a busca pelo nome completo ou abreviado |

O valor retornado apresenta o nome completo ou abreviado do agente de acordo com a Flag definida. É necessário enviar o tamanho da string, que pode ser adquirido pela função GetAgentNameLength

GetHistoryTrades

### Nome Tipo Descrição

const pwcTicker

const pwcBolsa

PWideChar Ticker do ativo

PWideChar Bolsa do ativo

dtDateStart PWideChar Data de início da requisição no formato DD/MM/YYYY HH:mm:SS (mm

minuto MM mês)

dtDateEnd PWideChar Data do fim da requisição no formato DD/MM/YYYY HH:mm:SS (mm

minuto MM mês)

É utilizado para solicitar as informações do histórico de um ativo a partir de uma data (pwcTicker = ‘PETR4’; dtDateStart = ‘06/08/2018 09:00:00’; dtDateEnd= ‘06/08/2018 18:00:00’). Retorno será dado na função de callback THistoryTradeCallback especificada por parâmetro para a função de inicialização. Em TProgressCallback será retornado o progresso de Download (1 até 100).

 SetDayTrade

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| bUseDayTrade | Integer | Indica se deve usar a flag de day trade (1 true, 0 false) |

Função disponível para clientes cujas corretoras tenham controle de risco DayTrade. Desta forma, as ordens são enviadas com a tag DayTrade. O parâmetro é um boleano (0 = False, 1 = True). Ao definí-lo como true, todas ordens serão enviadas com o modo DayTrade ativado. Para desativar, basta setar para falso.

 SetEnabledLogToDebug

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
|  |  |  |
| bEnabled | Integer | Indica se deve salvar logs de debug |

Função para definir uma se a DLL deve salvar logs para debug (1 = salvar / 0 = Não salvar).

RequestTickerInfo

### Nome Tipo Descrição

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| const pwcTicker | PWideChar | Ticker do ativo |
| const pwcBolsa | PWideChar | Bolsa do ativo |

É utilizado para buscar novas informações do ativo (ex. ISIN). A resposta é retornada nos callbacks

TAssetListInfoCallback, TAssetListInfoCallbackV2 e TAssetListCallback, caso os mesmos tenham sido enviados à DLL por meio das funções SetAssetListInfoCallback, SetAssetListInfoCallbackV2 e

SetAssetListCallback. Em caso de requisção de ticker inválido, um evento vai ser disparado na callback definida SetInvalidTickerCallback.

As funções abaixo fornecem um endereço de callback para a DLL retornar informações. Elas são opcionais para utilização da biblioteca. Caso elas não sejam especificadas, as informações correspondentes não serão fornecidas ao serem requisitadas.

 SetChangeCotationCallback

Utilizado para definir uma função de callback do tipo TChangeCotation, esta função notifica sempre que o ativo sofrer modificação no preço.

 SetAssetListCallback

Utilizado para definir uma função de callback do tipo TAssetListCallback, responsável pelo retorno da informações de ativos.

 SetAssetListInfoCallback

Utilizado para definir uma função de callback do tipo TAssetListInfoCallback, responsável pelo retorno da informações de ativos, retorna informações adicionais comparada a AssetListCallback.

 SetAssetListInfoCallbackV2

Semelhante a SetAssetListInfoCallback, porém retorna informações de setor, subsetor e segmento.

 SetInvalidTickerCallback

Utilizado para definir uma função de callback do tipo TInvalidTickerCallback, responsável pelo retorno de requisição de ticker inválido.

 SetChangeStateTickerCallback

Utilizado para definir o callback TChangeStateTicker que informa as modificações do estado do ticker, tais como, se o ativo está em leilão, suspenso, em pré-fechamento, after market ou fechado.

 SetAdjustHistoryCallback

Utilizado para definir o callback TAdjustHistoryCallback que informa o histórico de ajustes do ticker.

 SetAdjustHistoryCallbackV2

Utilizado para definir o callback TAdjustHistoryCallbackV2 que informa o histórico de ajustes do ticker.

 SetTheoreticalPriceCallback

Utilizado para definir a função de callback do tipo TTheoreticalPriceCallback, que recebe o preço e quantidades teóricas durante o leilão.

 SetHistoryCallbackV2

Utilizado para definir função de callback do tipo THistoryCallbackV2, similar a THistoryCallback, que recebe o histórico de ordens.

**Depreciada**: Utilize a callback SetOrderHistoryCallback.

 SetOrderChangeCallbackV2

Utilizado para definir função de callback do tipo TOrderChangeCallbackV2, similar a TOrderChangeCallback, que recebe atualizações de ordens.

**Depreciada**: Utilize a callback SetOrderCallback.

 SetOfferBookCallbackV2

Utilizado para definir função de callback do tipo TOfferBookCallbackV2, similar a TOfferBookCallback, recebe o livro de ofertas em um formato novo.

 SetPriceBookCallbackV2

Utilizado para definir função de callback do tipo TPriceBookCallbackV2, similar a TPriceBookCallback, recebe o livro de preço em um formato novo.

 SetStateCallback

Utilizado para definar a função de callback do tipo TStateCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

 SetTradeCallback

Utilizado para definar a função de callback do tipo TTradeCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

**Depreciada**: Utilize a callback SetTradeCallbackV2

 SetHistoryTradeCallback

Utilizado para definar a função de callback do tipo THistoryTradeCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

**Depreciada**: Utilize a callback SetHistoryTradeCallbackV2

 SetDailyCallback

Utilizado para definar a função de callback do tipo TDailyCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

 SetSerieProgressCallback

Utilizado para definar a função de callback do tipo TProgressCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

 SetOfferBookCallback

Utilizado para definar a função de callback do tipo TOfferBookCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

 SetPriceBookCallback

Utilizado para definar a função de callback do tipo TPriceBookCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin ou DLLInitializeMarketLogin.

 SetAssetPositionListCallback

Utilizado para definar a função de callback do tipo TAssetPositionListCallback. Disparado quando a uma alteração nas posições da conta. Envia o TConnectorAssetIdentifier que foi alterado, ou caso a alteração tenha sido na lista completa, um TConnectorAssetIdentifier com o campo Ticker em -1. Sobrepõe a callback do tipo TAccountCallback.

 SetAccountCallback

Utilizado para definar a função de callback do tipo TAccountCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin.

 SetHistoryCallback

Utilizado para definar a função de callback do tipo THistoryCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin.

**Depreciada**: Utilize a callback SetOrderHistoryCallback.

 SetOrderChangeCallback

Utilizado para definar a função de callback do tipo TOrderChangeCallback. Sobrepõe a callback definida pelo

DLLInitializeLogin.

**Depreciada**: Utilize a callback SetOrderCallback.

 SetOrderCallback

Utilizado para definar a função de callback do tipo TConnectorOrderCallback. Se for definida uma callback para SetOrderHistoryCallback, essa callback somente é dispara quando houver alterações/criações de uma única ordem. Caso contrário, essa callback é disparada para todo evento de ordem.

 SetOrderHistoryCallback

Utilizado para definar a função de callback do tipo TConnectorAccountCallback. Disparado quando o histórico de ordem de uma conta termina de carregar.

Ao assinar essa callback, as callbacks definidas em SetHistoryCallback, SetHistoryCallbackV2 e

SetOrderCallback **não** disparam mais quando, *e somente quando*, o histórico de ordem for carregado. Outros casos dessas callback continuam operando normalmente.

 SetTradeCallbackV2

Utilizado para definiar uma função de callback do tipo TConnectorTradeCallback. Disparado quando um ativo inscrito recebe um novo trade. Utilizar a função TranslateTrade para traduzir o ponteiro de trade recebido nessa callback.

O parâmetro a\_nFlags pode vir com a flag TC\_IS\_EDIT, indicando que esse trade é uma edição.

 SetHistoryTradeCallbackV2

Utilizado para definiar uma função de callback do tipo TConnectorTradeCallback. Disparado para receber histórico de trades para um ativo. Utilizar a função TranslateTrade para traduzir o ponteiro de trade recebido nessa callback.

O parâmetro a\_nFlags pode vir com a flag TC\_LAST\_PACKET, indicando que esse trade é o último do histórico.

As funções descritas abaixo estão disponíveis somente para inicialização com roteamento, após a utilização da função DLLInitializeLogin na inicialização.

 GetAccount

Função que retorna informações das contas vinculadas através do callback TAccountCallback passado como parâmetro para a função de inicialização.

 SendBuyOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| dPrice | Double | Preço alvo |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de compra limite. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

 SendSellOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| dPrice | Double | Preço alvo |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de venda limite. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

 SendMarketBuyOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de compra a mercado. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

 SendMarketSellOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de venda a mercado. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

SendStopBuyOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| dPrice | Double | Preço alvo de compra |
| dStopPrice | Double | Preço de stop |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de compra stop. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

 SendStopSellOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| dPrice | Double | Preço alvo de venda |
| dStopPrice | Double | Preço de stop |
| nAmount | Integer | Quantidade a ser negociada |

Envia ordem de venda stop. Retorna o ID interno (por sessão) da ordem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendOrder.

SendChangeOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |

pwcstrClOrdID PWideChar ClOrdID da ordem a ser modificada (Fornecido em

OrderChangeCallback)

dPrice PWideChar Preço alvo após edição

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nAmount | Integer | Quantidade após edição |

Envia uma ordem de modificação. Quando a modificação for de uma ordem stop, o preço stop deve ser informado como preço alvo e o preço limite será calculado com base no mesmo offset.

Função obsoleta em favor da nova função SendChangeOrderV2.

 SendCancelOrder

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |

pwcClOrdId PWideChar ClOrdID da ordem a ser cancelada (Fornecido em

OrderChangeCallback)

pwcSenha PWideChar Senha de roteamento

Envia uma ordem de cancelamento. O resultado da requisição de cancelamento pode ser acompanhado em

TOrderChangeCallback.

Função obsoleta em favor da nova função SendCancelOrderV2.

 SendCancelOrders

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |

Envia uma ordem para cancelar todas ordens de um ativo. O resultado da requisição de cancelamento pode ser acompanhado em TOrderChangeCallback para cada ordem cancelada.

Função obsoleta em favor da nova função SendCancelOrdersV2.

SendCancelAllOrders

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcSenha | PWideChar | Senha de roteamento |

Envia uma ordem para cancelar todas ordens em aberto de todos ativos. O resultado da requisição de cancelamento pode ser acompanhado em TOrderChangeCallback para cada ordem cancelada.

Função obsoleta em favor da nova função SendCancelAllOrdersV2.

 SendZeroPosition

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| pwcSenha | PWideChar | Senha de roteamento |
| dPrice | Double | Preço da ordem |

Envia uma ordem para zerar a posição de um determinado ativo. Retorna o ID interno (por sessão) da ordem de zeragem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendZeroPositionV2.

 SendZeroPositionAtMarket

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| pwcTicker | PWideChar | Ticker do ativo a ser negociado |
| pwcBolsa | PWideChar | Bolsa do ativo a ser negociado |
| pwcSenha | PWideChar | Senha de roteamento |

Envia uma ordem para zerar a posição de um determinado ativo com o valor de mercado. Retorna o ID interno (por sessão) da ordem de zeragem que pode ser comparado com o retorno do THistoryCallback.

Função obsoleta em favor da nova função SendZeroPositionV2.

GetOrders

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| pwcIDAccount | PWideChar | Identificador de conta (fornecido em GetAccount) |
| pwcIDCorretora | PWideChar | Identificador da corretora (fornecido em GetAccount) |
| dtStart | PWideChar | Data inicial no formato DD/MM/YYYY |
| dtEnd | PWideChar | Data final no formato DD/MM/YYYY |

Função que retorna as ordens em determinado período. Retorno feito pelo callback THistoryCallback, passado como parâmetro para a função de inicialização.

Função obsoleta em favor das novas funções HasOrdersInInterval, EnumerateOrdersByInterval e

EnumerateAllOrders.

|  |  |  |
| --- | --- | --- |
| GetOrder  **Nome** | **Tipo** | **Descrição** |
| pwcClOrdId | PWideChar | ClOrdID da ordem a ser retornada |

Função que retorna dados de uma ordem a partir de um ClOrdID. Retorno feito pelo callback

TOrderChangeCallback, passado como parâmetro para a função de inicialização. Função obsoleta em favor da nova função GetOrderDetails.

 GetOrderProfitID

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nProfitID | Int64 | ProfitID da ordem a ser retornada |

Função que retorna dados de uma ordem a partir de um ProfitID (ID interno por sessão). Retorno feito pelo callback TOrderChangeCallback, passado como parâmetro para a função de inicialização. O ProfitID é válido apenas durante a execução da aplicação, ao contrário do ClOrdID. Esse ID é o retorno das funções de envio de ordem.

Função obsoleta em favor da nova função GetOrderDetails.

 GetPosition

Função que retorna a posição para determinado ticker. Retorna uma estrutura de dados especificada abaixo. Com tamanho total (91 + N + T + K) bytes:

|  |  |  |
| --- | --- | --- |
| **Campo/descrição** | **Tipo** | **Tamanho** |
| Quantidade de contas | Integer | 4 bytes |
| Tamanho do buffer | Integer | 4 bytes |

|  |  |  |
| --- | --- | --- |
| **Campo/descrição** | **Tipo** | **Tamanho** |
| ID da corretora | Integer | 4 bytes |
| N tamanho string Conta | Short | 2 bytes |
| String conta | (Array de caracteres) | N Bytes |
| T tamanho string Titular | Short | 2 bytes |
| String titular | (Array de caracteres) | T Bytes |
| K tamanho string Ticker | Short | 2 bytes |
| String ticker | (Array de caracteres) | K Bytes |
| Intraday nQtd | Integer | 4 bytes |
| Intraday dPrice | Double | 8 bytes |
| Day SellAvgPriceToday | Double | 8 bytes |
| Day SellQtdToday | Integer | 4 bytes |
| Day BuyAvgPriceToday | Double | 8 bytes |
| Day BuyQtdToday | Integer | 4 bytes |
| Custodia Quantidade em D+1 | Integer | 4 bytes |
| Custodia Quantidade em D+2 | Integer | 4 bytes |
| Custodia Quantidade em D+3 | Integer | 4 bytes |
| Custodia Quantidade bloqueada | Integer | 4 bytes |
| Custodia Quantidade Pending | Integer | 4 bytes |
| Custodia Quantidade alocada | Integer | 4 bytes |
| Custodia Quantidade provisionada | Integer | 4 bytes |
| Custodia Quantidade da posição | Integer | 4 bytes |
| Custodia Quantidade Disponível | Integer | 4 bytes |
| Lado da Posição | Byte | 1 byte |

O campo lado da posição equivale a um tipo enumerado descrito abaixo:

Comprada = 1

Vendida = 2

Desconhecida = 0

Função obsoleta em favor da nova função GetPositionV2.

SetEnabledHistOrder

Esta função é utilizada para desativar/ativar o histórico e o update automático de ordens ao iniciar a aplicação (1

= Ativar / 0 = Desativar). Quando o histórico é desativado, a aplicação não recebe automaticamente os dados de ordens no início e, por isso, chamadas como GetPosition, que exigem a montagem da posição utilizando as operações, não retornarão resultados válidos. Para desativar o update automático, esta função deve ser utilizada logo após a chamada das funções de inicialização. É importante ressaltar que, ao desativar o histórico, o controle de posição não será calculado corretamente pela plataforma e as funcionalidades de zeragem e status da ordem podem ser comprometidas. O usuário deve estar ciente desses riscos antes de desativar o histórico..

|  |  |  |
| --- | --- | --- |
| SendOrder  **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0, 1 |
| AccountID | TConnectorAccountIdentifier | Estrutura para o identificador da conta |
| AssetID | TConnectorAssetIdentifier | Estrturua para o identificador do ativo |
| Password | PWideChar | Senha de roteamento em texto plano |
| OrderType | Byte | Indica se é ordem Limite, Mercado ou Stop |
| OrderSide | Byte | Indica se é compra ou venda |
| Price | Double | Preço da ordem, ordens a mercado deve ser -1 |
| StopPrice | Double | Preço stop, ordens não stop deve ser -1 |
| Quantity | Int64 | Quantidade |

Função para envio de ordens. Aceita como parâmetro um ponteiro para uma estratura do tipo

TConnectorSendOrder. É possível enviar todos os tipos de ordem em uma única função, além de aceitar tanto contas, sub-contas. Em caso de sucesso, retorna o ID local da ordem, caso aconteça algum erro, retorna uma código de erro. Estado da ordem pode ser acompanhado pela callback definida em SetOrderCallback.

A partir da versão 4.0.0.18, é possível informar a versão 1 da estrutura. Nessa versão os campos OrderType e

OrderSide passam a ser iguais ao valores de GetOrderDetails, e estão definidos em TConnectorOrderType e

TConnectorOrderSide.

Anterior a versão 4.0.0.18, apenas a versão 0 é suportada, e os campos OrderType e OrderSide tem os valores definidos em TConnectorOrderTypeV0 e TConnectorOrderSideV0.

SendChangeOrderV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0 |
| AccountID | TConnectorAccountIdentifier | Estrutura para o identificador da conta |

OrderID TConnectorOrderIdentifier Estrutura para o identificador da ordem. É possível

informar apenas um dos IDs.

Password PWideChar Senha de roteamento em texto plano

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Price | Double | Novo preço da ordem |
| StopPrice | Double | Novo preço stop |
| Quantity | Int64 | Nova quantidade |

Envia uma ordem de modificação. Aceita como parâmetro um ponteiro para uma estratura do tipo

TConnectorChangeOrder. Aceita tanto contas, como subcontas. Estado da ordem pode ser acompanhado pela callback definida em SetOrderCallback.

 SendCancelOrderV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0 |
| AccountID | TConnectorAccountIdentifier | Estrutura para o identificador da conta |

OrderID TConnectorOrderIdentifier Estrutura para o identificador da ordem. É possível

informar apenas um dos IDs.

Password PWideChar Senha de roteamento em texto plano

Envia uma ordem de cancelamento. Aceita como parâmetro um ponteiro para uma estrutura do tipo

TConnectorCancelOrder. Aceita tanto contas, como subcontas. O resultado da requisição de cancelamento pode ser acompanhado pela callback definida em SetOrderCallback.

 SendCancelOrdersV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0 |
| AccountID | TConnectorAccountIdentifier | Estrutura para o identificador da conta |
| AssetID | TConnectorAssetIdentifier | Estrutura para o identificador do ativo |
| Password | PWideChar | Senha de roteamento em texto plano |

Envia uma ordem para cancelar todas ordens de um ativo. Aceita como parâmetro um pontieor para uma estrutura do tipo TConnectorCancelOrders. Aceita tanto contas, como subcontas. O resultado da requisição de cancelamento pode ser acompanhado pela callback definida em SetOrderCallback para cada ordem cancelada.

SendCancelAllOrdersV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0 |
| AccountID | TConnectorAccountIdentifier | Estrutura para o identificador da conta |
| Password | PWideChar | Senha de roteamento em texto plano |

Envia uma ordem para cancelar todas ordens em aberto de todos ativos. Aceita como parâmetro um pontieor para uma estrutura do tipo TConnectorCancelAllOrders. Aceita tanto contas, como subcontas. O resultado da requisição de cancelamento pode ser acompanhado pela callback definida em SetOrderCallback para cada ordem cancelada.

 SendZeroPositionV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| Version | Byte | Versão da estrutura. Suportadas: 0 .. 1 |
| AccountID | TConnectorAccountIdentifier | Identificador da conta |
| AssetID | TConnectorAssetIdentifier | Identificador do ativo |
| Password | PWideChar | Senha de roteamento |
| Price | Double | Preço da zeragem |

PositionType Byte [V1] Tipo da posição, um dos valores de

TConnectorPositionType

Envia uma ordem para zerar a posição de um determinado ativo. Para zeragem a mercado, preço deve ser -1. Aceita tanto contas, sub-contas Aceita como parâmetro um ponteiro para uma estratura do tipo

TConnectorZeroPosition. Em caso de sucesso, retorna o ID local da ordem. Estado da ordem pode ser acompanhado pela callback definida em SetOrderCallback.

A partir da versão 1 do ponteiro, é necessário informar o tipo da posição.

 GetAccountCount

Returna o número total de contas carregadas. Não inclui subcontas.

GetAccounts

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_nStartSource | Integer | Índice para começar as buscas de contas |

a\_nStartDest Integer Índice do array a\_arAccounts indicando onde começar a gravação dos IDs

a\_nCount Integer Contagem de contas a serem buscadas Ponteiro para o primeiro índice de um array

a\_arAccounts PConnectorAccountIdentifierArrayOut

do tipo

TConnectorAccountIdentifierArrayOut

Função para buscar os identificadores das contas. Em caso de sucesso, retorna o número de contas encontradas. Não busca sub-contas.

GetAccountDetails

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_Account | TConnectorTradingAccountOut | Ponteiro para os detalhes da conta |

Função para retornar os detalhes de uma conta ou sub-conta. É preciso informar o identificador da conta no ponteiro.

 GetAccountCountByBroker

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_nBrokerID | Integer | ID da corretora |

Returna o número total de contas de uma corretora.

GetAccountsByBroker

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_nBrokerID | Integer | ID da corretora |
| a\_nStartSource | Integer | Índice para começar as buscas de contas |

a\_nStartDest Integer Índice do array a\_arAccounts indicando onde começar a gravação dos IDs

a\_nCount Integer Contagem de contas a serem buscadas Ponteiro para o primeiro índice de um array

a\_arAccounts PConnectorAccountIdentifierArrayOut

do tipo

TConnectorAccountIdentifierArrayOut

Função para buscar os identificadores das contas de uma corretora. Em caso de sucesso, retorna o número de contas encontradas. Não busca sub-contas.

 GetSubAccountCount

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_MasterAccountID | PConnectorAccountIdentifier | Conta master contendo as subcontas |

Returna o número total de sub-contas de uma conta.

GetSubAccounts

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_MasterAccountID | PConnectorAccountIdentifier | Conta master contendo as subcontas |
| a\_nStartSource | Integer | Índice para começar as buscas de contas |

### Nome Tipo Descrição

a\_nStartDest Integer Índice do array a\_arAccounts indicando onde começar a gravação dos IDs

a\_nCount Integer Contagem de contas a serem buscadas Ponteiro para o primeiro índice de um array

a\_arAccounts PConnectorAccountIdentifierArrayOut

do tipo

TConnectorAccountIdentifierArrayOut

Função para buscar os identificadores das sub-contas de uma conta. Em caso de sucesso, retorna o número de sub-contas encontradas.

 GetPositionV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_Position | TConnectorTradingAccountPosition | Dados da posição |

Função que retorna a posição para determinado conta/subconta e ativo. É preciso informar o identificador da conta e ativo no ponteiro.

A partir da versão 1 do ponteiro, é necessário informar o tipo da posição (TConnectorPositionType). O atributo EventID está relacionado ao EventID recebido pela callback TAssetPositionListCallback.

 GetOrderDetails

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_Order | TConnectorOrderOut | Ponteiro para detalhes da ordem |

Função para retornar os detalhes de uma ordem. É preciso informar o identificador da ordem no ponteiro.

TSystemTime é definido em [SystemTime](https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-systemtime).

 HasOrdersInInterval

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_AccountID | PConnectorAccountIdentifier | Identificador da conta |
| a\_dtStart | TSystemTime | Data de início |
| a\_dtEnd | TSystemTime | Data final |

Essa função retorna se o histórico de ordens de uma conta já foi carregado para um intervalo de datas, horários são ignorados aqui. Subcontas usam a conta master para buscar ordens.

Caso o histórico exista, a função retorna NL\_OK. Se o histórico não foi carregado, será realizada uma requisição ao servidor, e a função retornará NL\_WAITING\_SERVER. Chamar essa função enquanto a requisição é processada apenas retorna NL\_WAITING\_SERVER, não gerando requisição adicionais para o servidor. Se as datas estiveram

fora de ordem, a função retorna NL\_OUT\_OF\_RANGE. Nenhuma data não pode ser maior que a data retornada em

GetServerClock.

Em caso de requisição para o servidor, o resultado será notificado pela callback definida em

SetOrderHistoryCallback.

Uma vez que a ordens estão disponíveis, podem ser iteradas com as funções EnumerateOrdersByInterval ou

EnumerateAllOrders.

 EnumerateOrdersByInterval

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_AccountID | PConnectorAccountIdentifier | Identificador da conta/subconta |
| a\_OrderVersion | Byte | Versão do ponteiro da ordem para retornar em  a\_Callback |
| a\_dtStart | TSystemTime | Data/hora de início |
| a\_dtEnd | TSystemTime | Data/hora final |

a\_Param LPARAM Parâmetro definido pelo implementador, retornado em a\_Callback

a\_Callback TConnectorEnumerateOrdersProc Função para receber os ordens

Essa função itera sobre as ordens de uma conta/subconta, que estão em no intervalo definido (horários são respeitados). Para cada ordem que se enquadra do filtro, a função definida em a\_Callback é invocada.

O ponteiro de PConnectorOrder terá a versão definida em a\_OrderVersion. Os dados do ponteiro não tem garantia de integridade após cada iteração. O parâmetro a\_Param é somente repassado da função para a callback.

Caso não haja ordens para o intervalo, essa função irá requisitar o histórico de ordens, como se fosse a função

HasOrdersInInterval, retornando NL\_WAITING\_SERVER. Se as datas estiveram fora de ordem, a função retorna

NL\_OUT\_OF\_RANGE. A data final não pode ser maior que a data retornada em GetServerClock.

Em caso de sucesso, essa função somente retorna assim que iteração terminar, retornando NL\_OK. É possível parar a iteração retornando FALSE na callback.

EnumerateAllOrders

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| 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

Essa função itera sobre todas as ordens de uma conta/subconta. Para cada ordem que se enquadra do filtro, a função definida em a\_Callback é invocada.

O ponteiro de PConnectorOrder terá a versão definida em a\_OrderVersion. Os dados do ponteiro são destruídos após cada iteração. O parâmetro a\_Param é somente repassado da função para a callback.

Diferente da função EnumerateOrdersByInterval, essa função não realiza operações com o servidor, e retorna sucesso mesmo se não há contas carregadas para a conta/subconta.

Em caso de sucesso, essa função somente retorna assim que iteração terminar, retornando NL\_OK. É possível parar a iteração retornando FALSE na callback.

 EnumerateAllPositionAssets

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_AccountID | PConnectorAccountIdentifier | Identificador da conta/subconta |

a\_AssetVersion Byte Versão do identificador do ativo para retornar em

a\_Callback

a\_Param LPARAM Parâmetro definido pelo implementador, retornado em a\_Callback

a\_Callback TConnectorEnumerateAssetProc Função para receber os ativos

Essa função itera sobro os ativos de todas posições abertas da conta/subconta. Para cada ativo encontrado, a função definida em a\_Callback é invocada.

O identificador de TConnectorAssetIdentifier terá a versão definida em a\_OrderVersion. Os dados do identificador não tem garantia de integridade após cada iteração. O parâmetro a\_Param é somente repassado da função para a callback.

Em caso de sucesso, essa função somente retorna assim que iteração terminar, retornando NL\_OK. É possível parar a iteração retornando FALSE na callback.

 TranslateTrade

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| a\_pTrade | Pointer | Ponteiro com os dados de trade a serem traduzidos |
| a\_Trade | TConnectorTrade | Estrutura versionada para receber os dados traduzidos |

Essa função traduz um trade recebido pelas callbacks do tipo TConnectorTradeCallback.

## 3.2 Callbacks

Essa seção descreve como devem ser declaradas e o objetivo de cada função de callback da biblioteca.

Importante: Outras funções da DLL não devem ser utilizadas dentro de um callback.

Callbacks são chamados a partir da thread ConnectorThread e portanto estão em uma thread diferente da thread principal do programa do cliente.

As funções de callbacks devem ser todas declaradas com a convenção de chamadas stdcall

(https://en.wikipedia.org/wiki/X86\_calling\_conventions). Isso é válido para ambas versões, 32 e 64 bits.

TStateCallback = procedure(nConnStateType : Integer; Integer : 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

Corresponde ao callback para informar o estado de login, de conexão, de roteamento e de ativação do produto. De acordo com o tipo de nConnStateType informado, sendo eles:

CONNECTION\_STATE\_LOGIN = 0;

CONNECTION\_STATE\_ROTEAMENTO = 1;

CONNECTION\_STATE\_MARKET\_DATA = 2;

CONNECTION\_STATE\_MARKET\_LOGIN = 3;

// Conexão com servidor de login

// Conexão com servidor de roteamento

// Conexão com servidor de market data

// Login com servidor market data

LOGIN\_CONNECTED

LOGIN\_INVALID

= 0;

= 1;

LOGIN\_INVALID\_PASS = 2;

LOGIN\_BLOCKED\_PASS = 3;

LOGIN\_EXPIRED\_PASS = 4;

// Servidor de login conectado

// Login é inválido

// Senha inválida

// Senha bloqueada

// Senha expirada

LOGIN\_UNKNOWN\_ERR = 200; // Erro interno de login

ROTEAMENTO\_DISCONNECTED

ROTEAMENTO\_CONNECTING ROTEAMENTO\_CONNECTED

= 0;

= 1;

= 2;

ROTEAMENTO\_BROKER\_DISCONNECTED = 3;

ROTEAMENTO\_BROKER\_CONNECTING = 4;

ROTEAMENTO\_BROKER\_CONNECTED = 5;

MARKET\_DISCONNECTED = 0;

MARKET\_CONNECTING = 1;

MARKET\_WAITING = 2;

MARKET\_NOT\_LOGGED = 3;

MARKET\_CONNECTED = 4;

// Desconectado do servidor de market data

// Conectando ao servidor de market data

// Esperando conexão

// Não logado ao servidor de market data

// Conectado ao market data

CONNECTION\_ACTIVATE\_VALID = 0; // Ativação válida

CONNECTION\_ACTIVATE\_INVALID = 1; // Ativação inválida

Sendo o tipo nConnStateType recebido um dos valores de CONNECTION\_STATE, e Integer o estado de login do serviço específico. Os valores corretos para uma conexão válida são:

 nConnStateType = CONNECTION\_STATE\_LOGIN  Integer = LOGIN\_CONNECTED

 nConnStateType = CONNECTION\_STATE\_ROTEAMENTO  Integer = ROTEAMENTO\_CONNECTED

 nConnStateType = CONNECTION\_STATE\_MARKET\_DATA  Integer = MARKET\_CONNECTED

 nConnStateType = CONNECTION\_STATE\_MARKET\_LOGIN  Integer = CONNECTION\_ACTIVATE\_VALID

 TNewTradeCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o trade pertence |
| pwcDate | PWideChar | Data do trade no formato DD/MM/YYYY HH:mm:SS.ZZZ (mm minuto, MM mês e ZZZ milissegundo) |
| nTradeNumber | Cardinal | Número de série de um trade |
| dPrice | Double | Preço de execução |
| dVol | Double | Volume financeiro |
| nQtd | Integer | Quantidade |
| nBuyAgent | Integer | Agente comprador |
| nSellAgent | Integer | Agente vendedor |
| nTradeType | Integer | Tipo do trade |
| bEdit | Char | Indica se é uma edição |

Corresponde ao callback para informar um novo trade, recebido após se inscrever para este mesmo ativo (segundo função SubscribeTicker já especificada). O nTradeNumber é o identificador único do trade por pregão. bEdit informa se o trade recebido é uma edição (informação da bolsa) ou uma adição. O ID para identificar um trade editado é o pwcDate. tradeType indica o tipo de trade segundo a tabela abaixo:

1. Cross trade
2. Compra agressão
3. Venda agressão
4. Leilão
5. Surveillance
6. Expit
7. Options Exercise
8. Over the counter
9. Derivative Term
10. Index
11. BTC
12. On Behalf
13. RLP

32. Desconhecido

 TNewDailyCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o trade pertence |
| pwcDate | PWideChar | Data do trade no formato DD/MM/YYYY HH:mm:SS.ZZZ (mm minuto, MM mês e ZZZ milissegundo) |
| dOpen | Double | Preço do trade na abertura do mercado |
| dHigh | Double | Maior preço atingido |
| dLow | Double | Menor preço atingido |
| dClose | Double | Preço do último trade ocorrido |
| dVol | Double | Volume financeiro |
| dAjuste | Double | Ajuste do preço |
| dMaxLimit | Double | Limite superior de preço para ordem |
| dMinLimit | Double | Limite inferior de preço para ordem |
| dVolBuyer | Double | Volume de compradores |
| dVolSeller | Double | Volume de vendedores |
| nQtd | Integer | Quantidade |
| nNegocios | Integer | Número total de negócios ocorridos |
| nContratosOpen | Integer | Número de contratos abertos |
| nQtdBuyer | Integer | Número de compradores |
| nQtdSeller | Integer | Número de vendedores |
| nNegBuyer | Integer | Número de negócios compradores |
| nNegSeller | Integer | Número de negócios vendedores |

Corresponde ao callback para informar uma nova cotação com informações agregadas do dia de pregão.

TPriceBookCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nAction | Integer | Ação a ser feita no livro |
| nPosition | Integer | Posição a ser inserida a oferta |
| nSide | Integer | Compra ou venda (Compra=0, Venda=1) |
| nQtds | Integer | Quantidade vendida/comprada |
| nCount | Integer | Quantidade de oferta vendida/comprada |
| dPrice | Double | Preço ofertado |
| pArraySell | Pointer | Livro completo de venda |
| pArrayBuy | Pointer | Livro completo de compra |

Corresponde ao callback para informar uma atualização no livro de preços. Os parâmetros são válidos ou não de acordo com o valor de nAction, descrito abaixo discriminadamente:

 rAssetID: Ticker;

 nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);  nPosition: Posição no grid; (Válido em atAdd, atEdit, atDelete e atDeleteFrom).

 Side: Compra ou venda; (Sempre válido).

 nQtds: Quantidade vendida/Comprada; (Válido em atAdd e atEdit).

 nCount: Quantidade de oferta Vendida/Comprada; (Válido em atAdd e atEdit).  dPrice: Preço; (Válido em atAdd).

pArraySell, pArrayBuy: Lista com as ofertas de compra/venda; (Válidos em atFullBook).

Esse callback foi feito de modo a manter uma lista de ofertas de venda e compra separadas. Portanto, cada nAction recebido deve ser tratado de forma a alterar essas listas, dependendo do lado recebido em nSide, como descrito a seguir. Todos os ajustes que dependem de nPosition se referem à posição a partir do final da lista (em listas com início em 0, size - nPosition - 1).

 atAdd: Inserir uma nova oferta após posição dada por nPosition.  atDelete: Deletar uma oferta na posição dada por nPosition.

 atDeleteFrom: Remover todas as ofertas a partir da posição dada por nPosition.

 atEdit: Atualizar as informações da oferta que se encontra na posição dada por nPosition.  atFullBook: Criação do book com todas as ofertas existentes.

Essas informações são recebidas através dos parâmetros pArrayBuy e pArraySell. Para criação da lista, ao receber atFullBook, ambos arrays pArrayBuy e pArraySell possuem o seguinte layout em memória:

|  |  |  |  |
| --- | --- | --- | --- |
| Cabeçalho  **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Quantidade de ofertas (Q) | Integer | 4 bytes | 0 |
| Tamanho do array (deve ser usado em FreePointer) | Integer | 4 bytes | 4 |

Q entradas a serem inseridas no livro, contendo

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Preço | Double | 8 bytes | 8 |
| Quantidade | Integer | 4 bytes | 16 |
| Count | Integer | 4 bytes | 20 |

Para mais detalhes de como montar o livro corretamente, consultar exemplos em C++ e Delphi.

 TPriceBookCallbackV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nAction | Integer | Ação a ser feita no livro |
| nPosition | Integer | Posição a ser inserida a oferta |
| nSide | Integer | Compra ou venda (Compra=0, Venda=1) |
| nQtds | Int64 | Quantidade vendida/comprada |
| nCount | Integer | Quantidade de oferta vendida/comprada |
| dPrice | Double | Preço ofertado |
| pArraySell | Pointer | Livro completo de venda |
| pArrayBuy | Pointer | Livro completo de compra |

Corresponde ao callback para informar uma atualização no livro de preços. Os parâmetros são válidos ou não de acordo com o valor de nAction, descrito abaixo discriminadamente:

 rAssetID: Ticker;

 nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);  nPosition: Posição no grid; (Válido em atAdd, atEdit, atDelete e atDeleteFrom).

 Side: Compra ou venda; (Sempre válido).

 nQtds: Quantidade vendida/Comprada; (Válido em atAdd e atEdit).

 nCount: Quantidade de oferta Vendida/Comprada; (Válido em atAdd e atEdit).  dPrice: Preço; (Válido em atAdd).

pArraySell, pArrayBuy: Lista com as ofertas de compra/venda; (Válidos em atFullBook).

Esse callback foi feito de modo a manter uma lista de ofertas de venda e compra separadas. Portanto, cada nAction recebido deve ser tratado de forma a alterar essas listas, dependendo do lado recebido em nSide, como descrito a seguir. Todos os ajustes que dependem de nPosition se referem à posição a partir do final da lista (em listas com início em 0, size - nPosition - 1).

 atAdd: Inserir uma nova oferta após posição dada por nPosition.  atDelete: Deletar uma oferta na posição dada por nPosition.

 atDeleteFrom: Remover todas as ofertas a partir da posição dada por nPosition.

 atEdit: Atualizar as informações da oferta que se encontra na posição dada por nPosition.  atFullBook: Criação do book com todas as ofertas existentes.

Essas informações são recebidas através dos parâmetros pArrayBuy e pArraySell. Para criação da lista, ao receber atFullBook, ambos arrays pArrayBuy e pArraySell possuem o seguinte layout em memória:

|  |  |  |  |
| --- | --- | --- | --- |
| Cabeçalho  **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Quantidade de ofertas (Q) | Integer | 4 bytes | 0 |
| Tamanho do array (deve ser usado em FreePointer)  Q entradas a serem inseridas no livro, contendo | Integer | 4 bytes | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Preço | Double | 8 bytes | 8 |
| Quantidade | Int64 | 8 bytes | 16 |
| Count | Cardinal | 4 bytes | 24 |

Para mais detalhes de como montar o livro corretamente, consultar exemplos em C++ e Delphi.

TOfferBookCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nAction | Integer | Ação a ser feita no livro |
| nPosition | Integer | Posição a ser inserida a oferta |
| nSide | Integer | Compra ou venda (Compra=0, Venda=1) |
| nQtd | Integer | Quantidade vendida/comprada |
| nAgent | Integer | Identificador do agente |
| nOfferID | Integer | Identificador da oferta |
| dPrice | Double | Preço ofertado |
| bHasPrice | Char | 1 byte para especificar se existe preço |
| bHasQtd | Char | 1 byte para especificar se existe quantidade |
| bHasDate | Char | 1 byte para especificar se existe data |
| bHasOfferID | Char | 1 byte para especificar se existe id de oferta |
| bHasAgent | Char | 1 byte para especificar se existe agente |
| pwcDate | PWideChar | Data da oferta no formato DD/MM/YYYY HH:mm:SS.ZZZ (mm minuto, MM mês e ZZZ milissegundo) |
| pArraySell | Pointer | Livro completo de venda |
| pArrayBuy | Pointer | Livro completo de compra |

Corresponde ao callback para informar uma atualização no livro de ofertas:

 rAssetID: Ticker; nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);  nPosition: Posição no array; nSide: Lado da ordem (Compra=0, Venda=1);

 nQtd: Quantidade vendida/Comprada;

 nAgent: indicam os IDs dos agentes de compra e venda, respectivamente; Pode-se se obter o nome destes através das funções GetAgentName já especificada;

O callback é tratado seguindo a mesma especificação do TPriceBookCallback, com exceção do layout dos arrays pArrayBuy e pArraySell:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cabeçalho**  **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Q Quantidade de ofertas | Integer | 4 bytes | 0 |
| Tamanho do array (deve ser usado em FreePointer) | Integer | 4 bytes | 4 |

### Array

Q entradas a serem inseridas no livro, contendo

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Preço | Double | 8 bytes | 8 |
| Quantidade | Integer | 4 bytes | 16 |
| Agente | Integer | 4 bytes | 20 |
| OfferID | Int64 | 8 bytes | 24 |
| T tamanho string Data | Short | 2 bytes | 32 |
| Data da oferta | Array of bytes | T bytes | 34 |

### Rodapé

Após as Q entradas no livro

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| OfferBookFlags | Cardinal | 4 bytes | (Q \* 49) + 8 |

O campo OfferBookFlags cotém as seguintes flags

|  |  |  |
| --- | --- | --- |
| **Flag** | **Valor** | **Significado** |
| OB\_LAST\_PACKET | 1 | Indica se é a última página do livro de ofertas |

 TOfferBookCallbackV2

### Nome Tipo Descrição

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nAction | Integer | Ação a ser feita no livro |
| nPosition | Integer | Posição a ser inserida a oferta |
| nSide | Integer | Compra ou venda (Compra=0, Venda=1) |
| nQtd | Int64 | Quantidade vendida/comprada |
| nAgent | Integer | Identificador do agente |
| nOfferID | Integer | Identificador da oferta |
| dPrice | Double | Preço ofertado |
| bHasPrice | Char | 1 byte para especificar se existe preço |
| bHasQtd | Char | 1 byte para especificar se existe quantidade |
| bHasDate | Char | 1 byte para especificar se existe data |
| bHasOfferID | Char | 1 byte para especificar se existe id de oferta |
| bHasAgent | Char | 1 byte para especificar se existe agente |
| pwcDate | PWideChar | Data da oferta no formato DD/MM/YYYY HH:mm:SS.ZZZ (mm minuto, MM mês e ZZZ milissegundo) |
| pArraySell | Pointer | Livro completo de venda |
| pArrayBuy | Pointer | Livro completo de compra |

Corresponde ao callback para informar uma atualização no livro de ofertas:

 rAssetID: Ticker; nAction: (atAdd = 0, atEdit = 1, atDelete = 2, atDeleteFrom = 3, atFullBook = 4);  nPosition: Posição no array; nSide: Lado da ordem (Compra=0, Venda=1);

 nQtd: Quantidade vendida/Comprada;

 nAgent: indicam os IDs dos agentes de compra e venda, respectivamente; Pode-se se obter o nome destes através das funções GetAgentName já especificada;

O callback é tratado seguindo a mesma especificação do TPriceBookCallbackV2, com exceção do layout dos arrays pArrayBuy e pArraySell:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cabeçalho**  **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Q Quantidade de ofertas | Integer | 4 bytes | 0 |
| Tamanho do array (deve ser usado em FreePointer) | Integer | 4 bytes | 4 |

### Array

Q entradas a serem inseridas no livro, contendo

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| Preço | Double | 8 bytes | 8 |
| Quantidade | Int64 | 8 bytes | 16 |
| Agente | Integer | 4 bytes | 24 |
| OfferID | Int64 | 8 bytes | 28 |
| T tamanho string Data | Short | 2 bytes | 36 |
| Data da oferta | Array of bytes | T bytes | 38 |

### Rodapé

Após as Q entradas no livro

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Offset** |
| OfferBookFlags | Cardinal | 4 bytes | (Q \* 53) + 8 |

O campo OfferBookFlags cotém as seguintes flags

|  |  |  |
| --- | --- | --- |
| **Flag** | **Valor** | **Significado** |
| OB\_LAST\_PACKET | 1 | Indica se é a última página do livro de ofertas |

 THistoryTradeCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o trade pertence |
| pwcDate | PWideChar | Data do trade no formato DD/MM/YYYY HH:mm:SS.ZZZ (mm minuto, MM mês e ZZZ milissegundo) |
| nTradeNumber | Cardinal | Número de série de um trade |
| dPrice | Double | Preço de execução |
| dVol | Double | Volume financeiro |
| nQtd | Integer | Quantidade |
| nBuyAgent | Integer | Agente comprador |
| nSellAgent | Integer | Agente vendedor |
| nTradeType | Integer | Tipo do trade |

Corresponde ao callback de trades que foram solicitados a partir da função GetHistoryTrades.

TProgressCallback

### Nome Tipo Descrição

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o pedido de histórico se refere |
| nProgress | Integer | Valor de progresso (0-100) |

Corresponde ao callback do progresso do THistoryTradeCallback.

 TTinyBookCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual a oferta pertence |
| dPrice | Double | Preço da oferta |
| nQtd | Integer | Quantidade |
| nSide | Integer | Lado comprador ou vendedor (Compra=0, Venda=1) |

Corresponde ao callback do topo do livro de preço. rAssetID informa a qual ativo pertence de acordo com a estrutura TAssetIDRec já especificada. sPrice: Preço; nQtd : Quantidade venda/compra; nSide: Lado da ordem (Compra=0, Venda=1)

Abaixo estão descritas os callbacks apenas disponíveis após a inicialização utilizando a função

DLLInitializeLogin, portanto somente para inicialização com roteamento.

 TAccountCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nCorretora | Integer | Identificador da corretora |
| CorretoraNomeCompleto | PWideChar | Nome completo da corretora |
| AccountID | PWideChar | Identificação da conta de cliente |
| NomeTitular | PWideChar | Nome do titular da conta |

Corresponde ao callback para informar as contas existentes. É possível verificar se a conta é de simulação através do nome ou identificador da corretora.

TOrderChangeCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nCorretora | Integer | Identificador da corretora |
| nQtd | Integer | Quantidade da ordem |
| nTradedQtd | Integer | Quantidade já executada |
| nLeavesQtd | Integer | Quantidade pendente de execução |
| nSide | Integer | Lado da ordem (Compra=1, Venda=2) |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| dPrice | Double | Preço da ordem |
| dStopPrice | Double | Preço de stop em caso de ordem stop |
| dAvgPrice | Double | Média do preço executado |
| nProfitID | Int64 | Identificador interno por sessão da ordem |
| TipoOrdem | PWideChar | Tipo da ordem |
| Conta | PWideChar | Identificador da conta |
| Titular | PWideChar | Titular da conta |
| ClOrdID | PWideChar | Identificador único da ordem (permanente) |
| Status | PWideChar | Status da ordem |
| Date | PWideChar | Data de execução da ordem |
| TextMessage | PWideChar | Mensagem de informações extras |

Corresponde ao callback para informar as modificações de ordens enviadas por uma conta.

THistoryCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nCorretora | Integer | Identificador da corretora |
| nQtd | Integer | Quantidade da ordem |
| nTradedQtd | Integer | Quantidade já executada |
| nLeavesQtd | Integer | Quantidade pendente de execução |
| nSide | Integer | Lado da ordem (Compra=1, Venda=2) |
| dPrice | Double | Preço da ordem |
| dStopPrice | Double | Preço de stop em caso de ordem stop |
| dAvgPrice | Double | Média do preço executado |
| nProfitID | Int64 | Identificador interno por sessão da ordem |
| TipoOrdem | PWideChar | Tipo da ordem |
| Conta | PWideChar | Identificador da conta |
| Titular | PWideChar | Titular da conta |
| ClOrdID | PWideChar | Identificador único da ordem (permanente) |
| Status | PWideChar | Status da ordem |
| Date | PWideChar | Data de execução da ordem |

Corresponde ao callback da solicitação de histórico de ordens. O histórico corresponde apenas às ordens do dia atual.

 TAssetListCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| pwcName | PWideChar | Descrição do ativo |

Corresponde ao callback de solicitação de informação de ativos. É necessário utilizar a função

SetAssetListCallback para que essa função receba dados.

 TAssetListInfoCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual a informação pertence |
| pwcName | PWideChar | Nome do ativo |
| pwcDescription | PWideChar | Descrição do ativo |
| nMinOrderQtd | Integer | Mínima quantidade de ordens permitidas |
| nMaxOrderQtd | Integer | Máxima quantidade de ordens permitidas |
| nLote | Integer | Tamanho de um lote |
| stSecurityType | Integer | Tipo do ativo \* |
| ssSecuritySubType | Integer | Subtipo do ativo \*\* |
| dMinPriceIncrement | Double | Incremento mínimo de preço |
| dContractMultiplier | Double | Multiplicador do contrato |
| strValidDate | PWideChar | Data de validade caso expire |
| strISIN | PWideChar | String ISIN do ativo |

Corresponde ao callback de informações de ativos. O campo stSecurityType representa o tipo do ativo retornado, que pode ser um dos abaixo:

\* Tipo do Ativo

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

255. stNelogicaSyntheticAsset

O campo ssSecuritySubType é uma especificação dentro do tipo e pode ser um dos abaixo:

\*\* Subtipo do ativo

1. ssFXSpot
2. ssGold
3. ssIndex
4. ssInterestRate
5. ssFXRate
6. ssForeignDebt
7. ssAgricultural
8. ssEnergy
9. ssEconomicIndicator
10. ssStrategy
11. ssFutureOption
12. ssVolatility
13. ssSwap
14. ssMiniContract
15. ssFinancialRollOver
16. ssAgriculturalRollOver
17. ssCarbonCredit
18. ssUnknown
19. ssFractionary
20. ssStock
21. ssCurrency
22. ssOTC // OTC MercadoBalcao
23. ssFII // FII Fundo de Investimento Imobiliario

// PUMA 2.0 -Equities

1. ssOrdinaryRights // DO
2. ssPreferredRights // DP
3. ssCommonShares // ON
4. ssPreferredShares // PN
5. ssClassApreferredShares // PNA
6. ssClassBpreferredShares // PNB
7. ssClassCpreferredShares // PNC
8. ssClassDpreferredShares // PND
9. ssOrdinaryReceipts // ON REC
10. ssPreferredReceipts // PN REC
11. ssCommonForward
12. ssFlexibleForward
13. ssDollarForward
14. ssIndexPointsForward
15. ssNonTradeableETFIndex
16. ssPredefinedCoveredSpread
17. ssTraceableETF
18. ssNonTradeableIndex
19. ssUserDefinedSpread
20. ssExchangeDefinedspread // Não usado atualmente
21. ssSecurityLoan
22. ssTradeableIndex
23. ssOthers
24. ssBrazilianDepositaryReceipt // BDR
25. ssFund
26. ssOtherReceipt
27. ssOtherRight
28. ssUNIT
29. ssClassEPreferredShare // PNE
30. ssClassFPreferredShare // PNF
31. ssClassGPreferredShare // PNG
32. ssWarrant
33. ssNonTradableSecurityLending
34. ssForeignIndexETF
35. ssGovernmentETF
36. ssIpoOrFollowOn
37. ssGrossAuction
38. ssNetAuction
39. ssTradableIndexInPartnership
40. ssNontradableIndexInPartnership
41. ssFixedIncomeETF
42. ssNontradableFixedIncomeETF
43. ssOutrightPurchase
44. ssSpecificCollateralRepo
45. ssDebenture
46. ssRealStateReceivableCertificate
47. ssAgribusinessReceivableCertificate
48. ssPromissoryNote
49. ssLetraFinanceira
50. ssAmericanDepositaryReceipt
51. ssUnitInvestmentFund
52. ssReceivableInvestmentFund
53. ssOutrightTPlus1
54. ssRepoTPlus1
55. ssNonTradableGrossSettlement
56. ssNonTradableNetSettlement
57. ssETFPrimaryMarket
58. ssSharesPrimaryMarket
59. ssRightsPrimaryMarket
60. ssUnitPrimaryMarket
61. ssFundPrimaryMarket
62. ssForeignIndexETFPrimaryMarket
63. ssWarrantPrimaryMarket
64. ssReceiptPrimaryMarket
65. ssGermanPublicDebts
66. ssStockRollover
67. ssStrategySpotDollar
68. ssTargetRate
69. ssTradableETFRealState
70. ssNonTradableETFRealEstate

254. ssDefault

 TAssetListInfoCallbackV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual a informação pertence |
| pwcName | PWideChar | Nome do ativo |
| pwcDescription | PWideChar | Descrição do ativo |
| nMinOrderQtd | Integer | Mínima quantidade de ordens permitidas |
| nMaxOrderQtd | Integer | Máxima quantidade de ordens permitidas |
| nLote | Integer | Tamanho de um lote |
| stSecurityType | Integer | Tipo do ativo \* |
| ssSecuritySubType | Integer | Subtipo do ativo \*\* |
| dMinPriceIncrement | Double | Incremento mínimo de preço |
| dContractMultiplier | Double | Multiplicador do contrato |
| strValidDate | PWideChar | Data de validade caso expire |
| strISIN | PWideChar | String ISIN do ativo |
| strSetor | PWideChar | Setor de atuação |
| strSubSetor | PWideChar | Subsetor dentro do setor |
| strSegmento | PWideChar | Segmento de atuação |

Extensão do callback TAssetListInfoCallback, apenas adiciona os campos setor, subsetor e segmento.

 TInvalidTickerCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| AssetID | TConnectorAssetIdentifier | Ativo inválido |

Corresponde ao callback para retorno de requisição do ticker inválido.

TTheoreticalPriceCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual a informação pertence |
| dTheoreticalPrice | Double | Preço teórico |
| nTheoreticalQtd | Int64 | Quantidade teórica |

Corresponde ao callback para retorno do preço e quantidades teóricas durante o leilão de um ativo.

 TAdjustHistoryCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo correspondente ao ajuste |
| dValue | Double | Valor do ajuste |
| strAdjustType | PWideChar | Tipo de ajuste \* |
| strObserv | PWideChar | Observação |
| dtAjuste | PWideChar | Data do ajuste |
| dtDeliber | PWideChar | Data de deliberação |
| dtPagamento | PWideChar | Data do pagamento |
| nAffectPrice | Integer | Indica se afeta ou não o preço |

Corresponde ao callback de ajustes de um ativo. Para utilizar esse callback é necessário enviá-lo à DLL através da função SetAdjustHistoryCallback. Preferir utilizar a função SetAdjustHistoryCallbackV2, nela há uma descrição mais detalhada de como realizar o cálculo do ajuste.

TAdjustHistoryCallbackV2

\* Tipo de ajuste 'None'

'Unknown' 'JurosRF'

'Dividendo' 'Rendimento' 'Subscricao'

'Desdobramento' 'ResgateTotalRF' 'ResgateTotalRV' 'AmortizacaoRF'

'JurosCapProprio' 'SubsComRenuncia' 'Bonificacao'

'Grupamento' 'JuncaoSerie' 'Cisao'

'Unknown'

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo correspondente ao ajuste |
| dValue | Double | Valor do ajuste |
| strAdjustType | PWideChar | Tipo de ajuste \* |
| strObserv | PWideChar | Observação |
| dtAjuste | PWideChar | Data do ajuste |
| dtDeliber | PWideChar | Data de deliberação |
| dtPagamento | PWideChar | Data do pagamento |
| nFlags | Cardinal | Flag de soma (descrita abaixo) |
| dMult | Double | Multiplicador |

Corresponde ao callback de ajustes de um ativo. Para utilizar esse callback é necessário enviá-lo à DLL através da função SetAdjustHistoryCallbackV2. nFlags é um campo de bits b0 a b31, onde o bit 0 (menos significativo) indica se o ajuste afeta o preço e o bit 1 indica se é um ajuste de Soma. dMult é o valor pré-computado que deve ser multiplicado pelo preço para realizar o ajuste, somente é utilizado caso o ajuste não seja um ajuste de soma e seja um ajuste que afeta preço, informação fornecida no campo nFlags. O valor -9999 de dMult indica que o mesmo é inválido e não deve ser utilizado. Caso o valor dMult seja inválido, utiliza-se dValue para realizar o cálculo, sendo uma subtração em caso de ajuste de soma e divisão caso contrário.

Para realizar o cálculo do ajuste é possível utilizar os parâmetros da seguinte forma:

 Quando dMult for um valor válido, o ajuste é feito multiplicando o preço por esse valor.  Quando a flag de soma está setada, o valor de ajuste é subtraído do preço

 Quando a flag de soma não está setada, o preço é dividido pelo valor de ajuste.

Pseudocódigo:

TChangeCotation

enquanto Data < DataAjuste se nFlag AND 1 e

(tipo diferente de Grupamento, Junção, Desdobramento e não(Unknown e não(nFlag AND 2))) ou

(tipo é Grupamento, Junção, Desdobramento e não(nFlag AND 2) então

se dMult <> -9999

Resultado := Resultado \* dMult senão

se (nFlag AND 2)

Preço := Preço - ValorAjuste senão

Preço := Preço / ValorAjuste

### Nome Tipo Descrição

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo em que ocorreu a mudança |
| pwcDate | PWideChar | Data da mudança na cotação |
| nTradeNumber | Cardinal | Número sequencial do trade em que ocorreu a mudança |
| dPrice | Double | Preço novo |

Este callback é usado para informar quando ocorrer uma modificação de preço no ativo, informando qual foi o último preço e hora da negociado. Para utilizar esse callback é necessário enviá-lo à DLL através da função

SetChangeCotationCallback.

 TChangeStateTicker

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo em que ocorreu a mudança |
| pwcDate | PWideChar | Data da mudança de estado |
| nState | Integer | Estado do ativo |

Corresponde ao callback de identificação de alteração de estado do ativo. A data informada é a data em que houve modificação do estado, apenas alguns estados mostram a data. Os estados possíveis estão listados abaixo:

0. tcsOpened

1. tcsFrozen
2. tcsInhibited
3. tcsAuctioned

6. tcsClosed

10. tcsPreClosing

13. tcsPreOpening

// Ativo em negociação aberta

// Ativo em leilão

// Ativo com negociação fechada

THistoryCallbackV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nCorretora | Integer | Identificador da corretora |
| nQtd | Integer | Quantidade da ordem |
| nTradedQtd | Integer | Quantidade já executada |
| nLeavesQtd | Integer | Quantidade pendente de execução |
| nSide | Integer | Lado da ordem (Compra=1, Venda=2) |
| nValidity | Integer | Tipo de validade da ordem\* |
| dPrice | Double | Preço da ordem |
| dStopPrice | Double | Preço de stop em caso de ordem stop |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| dAvgPrice | Double | Média do preço executado |
| nProfitID | Int64 | Identificador interno por sessão da ordem |
| TipoOrdem | PWideChar | Tipo da ordem |
| Conta | PWideChar | Identificador da conta |
| Titular | PWideChar | Titular da conta |
| ClOrdID | PWideChar | Identificador único da ordem (permanente) |
| Status | PWideChar | Status da ordem |
| LastUpdate | PWideChar | Data da última atualização da ordem |
| CloseDate | PWideChar | Data do fechamento da ordem, se já estiver fechada |
| ValidityDate | PWideChar | Data de referência para a validade da ordem |

Corresponde ao callback secundário (opcional) da solicitação de histórico de ordens. Para utilizar esse callback é necessário enviá-lo à DLL através da função SetHistoryCallbackV2, sendo, então, chamado nas mesmas ocasiões que THistoryCallback for chamado. O histórico corresponde apenas às ordens do dia atual. O campo nValidity representa o tipo de validade da ordem retornada, que pode ser um dos valores abaixo:

\* Tipo de validade da ordem

1. btfDay
2. btfGoodTillCancel
3. btfAtTheOpening
4. btfImmediateOrCancel
5. btfFillOrKill
6. btfGoodTillCrossing
7. btfGoodTillDate
8. btfAtTheClose

201. btfGoodForAuction

200. btfUnknown

TOrderChangeCallbackV2

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| rAssetID | TAssetIDRec | Ativo ao qual o livro pertence |
| nCorretora | Integer | Identificador da corretora |
| nQtd | Integer | Quantidade da ordem |
| nTradedQtd | Integer | Quantidade já executada |
| nLeavesQtd | Integer | Quantidade pendente de execução |
| nSide | Integer | Lado da ordem (Compra=1, Venda=2) |
| nValidity | Integer | Tipo de validade da ordem |

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| dPrice | Double | Preço da ordem |
| dStopPrice | Double | Preço de stop em caso de ordem stop |
| dAvgPrice | Double | Média do preço executado |
| nProfitID | Int64 | Identificador interno por sessão da ordem |
| TipoOrdem | PWideChar | Tipo da ordem |
| Conta | PWideChar | Identificador da conta |
| Titular | PWideChar | Titular da conta |
| ClOrdID | PWideChar | Identificador único da ordem (permanente) |
| Status | PWideChar | Status da ordem |
| LastUpdate | PWideChar | Data da última atualização da ordem |
| CloseDate | PWideChar | Data do fechamento da ordem, se já estiver fechada |
| ValidityDate | PWideChar | Data de referência para a validade da ordem |
| TextMessage | PWideChar | Mensagem de informações extras |

Corresponde ao callback secundário (opcional) para informar as modificações de ordens enviadas por uma conta. Para utilizar esse callback é necessário enviá-lo à DLL através da função SetOrderChangeCallbackV2, sendo, então, chamado nas mesmas ocasiões que TOrderChangeCallback for chamado. O campo nValidity representa o tipo de validade da ordem retornada e os valores possíveis podem ser checados na documentação de THistoryCallbackV2.

 TBrokerAccountListCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nCorretora | Integer | Identificador da corretora |
| Changed | Cardinal | Status de mudança das contas |

Corresponde ao callback para informar as contas existentes já disponiveis para usar.

 TBrokerSubAccountListCallback

|  |  |  |
| --- | --- | --- |
| **Nome** | **Tipo** | **Descrição** |
| nCorretora | Integer | Identificador da corretora |
| AccountID | TConnectorAccountIdentifier | Identificador da conta |

Corresponde ao callback para informar as sub-contas existentes já disponiveis para usar.

# Uso do Produto

## Inicializando com Roteamento

Para utilizar a biblioteca é fundamental inicializar os serviços através das funções de inicialização. Mais especificamente, caso os serviços de roteamento sejam utilizados, deve-se utilizar a função

DLLInitializeLogin, que fará a conexão com os servidores de roteamento e market data.

Essa função é descrita na seção de funções expostas e requer um código de ativação fornecido no momento da contratação do produto, bem como nome de usuário e senha para efetuar o login no servidor de autenticação. Os outros parâmetros são callbacks obrigatórios que serão chamados pela DLL durante o uso que precisam ser especificados no momento da inicialização.

É importante notar que todos os callbacks ocorrem em uma thread chamada ConnectorThread e, portanto, ocorrem simultaneamente à aplicação cliente. A aplicação cliente deve processar os dados fornecidos através dos callbacks como dados a serem consumidos de outra thread. Sendo assim, caso necessário, devem tratar a escrita desses dados com seções críticas ou mutexes.

Os dados recebidos por meio de callbacks são armazenados em uma única fila de dados, portanto, qualquer processamento demorado dentro das funções de callback pode atrasar a fila de processamento de mensagens interna da DLL e causar atrasos no recebimento de trades ou outras informações. Para evitar isso, os dados devem ser processados e enviados para outras threads da aplicação imediatamente, ou realizar o mínimo de processamento possível. Acessos a banco de dados ou escritas em disco devem ser evitados durante o processamento de um callback.

Por fim, é importante ressaltar que os callbacks são projetados apenas para receber dados. Portanto, as funções de requisições à DLL ou qualquer outra função da interface da DLL não devem ser chamadas dentro de um callback, pois isso pode causar exceções inesperadas e comportamento indefinido.

Mais detalhes de implementação podem ser esclarecidos nos exemplos disponibilizados.

## Inicializando com Market Data

O processo de inicialização do Market Data é análogo à inicialização com Roteamento, com a diferença do nome da função de inicialização DLLInitializeMarketLogin e redução de callbacks enviados por parâmetro, pois os mesmos são relacionados a ordens ou contas de roteamento.

## Tipos de dados

Todos os tipos citados nesse documento são tipos especificados na linguagem Delphi, abaixo estão alguns links para conversão ou mapeamento desses tipos para as linguagens dos exemplos.

 Mapeamento de tipos de Delphi para C

 https://docwiki.embarcadero.com/RADStudio/Tokyo/en/Delphi\_to\_C%2B%2B\_types\_mapping  Conversão de tipos C para Python

 https://docs.python.org/2/library/ctypes.html  Conversão de tipos de Delphi para C#

 <http://www.netcoole.com/delphi2cs/datatype.htm>

## Linkagem em 32 bits

Para utilizar a biblioteca em 32 bits é necessário que a aplicação também seja compilada em 32 bits. Por ser 32 bits, há uma limitação de memória em 4GB, que será compartilhada entre a biblioteca e a aplicação cliente.

Portanto, não é recomendado fazer requisições de muitos dados em apenas uma requisição, pois isso pode exceder o limite de memória do processo.

 C#

 Utilizando Visual Studio, é necessário alterar a plataforma alvo em Configuration Manager de Any CPU para x86.

 Python

 É necessário que o interpretador python.exe também seja 32 bits. Além disso existe um bug em python 32 bits em que um callback contendo um tipo maior que 32 bits falha e causa uma exception. Link para acompanhamento: https://bugs.python.org/issue41021. Por isso solicitamos que o cliente caso queira utilizar python 32 bits utilize a versão 3.6.2 que foi testada pela equipe da Nelogica e não possui esse problema.

Para as demais linguagens é necessário apenas trocar o modo de compilação para 32 bits.

## Linkagem em 64 bits

Para utilizar a biblioteca em 64 bits a aplicação também deve ser compilada em 64 bits. A convenção de chamadas continua sendo stdcall, assim como na versão 32 bits. Não existem problemas conhecidos para as linguagens de exemplo na versão 64 bits, portanto não há uma versão recomendada, é possível o uso das últimas versões de cada uma das linguagens.

A versão 64 bits não possui limitação de memória e portanto pode utilizar o máximo de memória disponível no sistema, podendo requisitar mais dados em uma requisição, limitado pela quantidade de RAM disponível.