**Manual de Uso**

**Descrição do Produto**

Dentro do arquivo *zip*, se apresentam quatro diretórios ‘*DLL e Executavel’* , ‘Exemplo C++’, ‘*Exemplo Delphi’, ‘Interface’ e ‘Manual’*. No diretório *‘DLL e Executavel’*, se encontra a biblioteca, ou seja, os arquivos binários (*DLL*) necessários para o uso do produto e exemplo executável para validação das funcionalidades. No diretório *‘Interface’* se encontra as variáveis/tipos/callbacks de interface para comunicação com a DLL. Nos diretórios *‘Exemplo C++’ e ‘Exemplo Delphi’,* , há o código fonte como exemplos de uso do produto.

**Descrição da Biblioteca**

No diretório *‘DLL e Executavel’*, há um arquivo *DLL* (*ProfitDLL.dll)* sendo este a Biblioteca.

A Biblioteca exporta funções básicas de comunicação com o servidor de Market Data e Roteamento, permitindo conexão e recebimento de dados referentes aos eventos de negociação em tempo real. A biblioteca está compilada para plataforma Windows 32 bits. As funções para comunicação com a mesma se apresentam a seguir.

**Interface da Biblioteca**

A Biblioteca se comunica através de funções expostas que podem ser invocadas e funções *callback* que são atribuídas a ela. Também são usados estruturas de dados específicas. Estes são especificados a seguir, com sua codificação original Delphi e um equivalente para C++ [**TDM-GCC** (version 4.9.2, 32 bit, SJLJ)].

**Estruturas de dados**

|  |
| --- |
| Delphi  PAssetIDRec = ^TAssetIDRec;  TAssetIDRec = **packed record**  pwcTicker : PWideChar;  pwcBolsa : PWideChar;  nFeed : Integer;  **end**; |

Aqui, pwcTicker apresenta o nome do ativo, pwcBolsa o nome da bolsa. O feed, representado por nFeed, pode apresentar valores **0**, para valores provenientes da *Nelogica*, ou **255**, para fontes desconhecidas.

**Funções expostas**

As funções expostas apresentam tipos que expressam *callbacks*, apresentados adiante. Todas funções com retorno **Short** (em C: **signed char**) seguem os seguintes códigos de erro:

NL\_OK = 000; // OK  
NL\_ERR\_INIT = 080; // Not initialized  
NL\_ERR\_INVALID\_ARGS = 090; // Invalid arguments  
NL\_ERR\_INTERNAL\_ERROR = 100; // Internal error

|  |
| --- |
| Delphi   * **function** **DLLInitializeLogin**(   const pwcActivationKey : **PWideChar**;  const pwcUser : **PWideChar**;  const pwcPassword : **PWideChar**;  StateCallback : **TStateCallBack**;  HistoryCallBack : **THistoryCallBack**;  AccountCallback : **TAccountCallback**;  NewDailyCallback : **TNewDailyCallback**;  PriceBookCallback : **TPriceBookCallback**;  OfferBookCallback : **TOfferBookCallback;**  HistoryTradeCallBack **: THistoryTradeCallBack;**  ProgressCallBack : **TProgressCallBack**;  TinyBookCallBack **: TTinyBookCallBack**)  : **Short**; stdcall;   * **function** **DLLInitializeMarketLogin**(   const pwcActivationKey : **PWideChar**;  const pwcUser : **PWideChar**;  const pwcPassword : **PWideChar**;  StateCallback : **TStateCallBack**;  NewDailyCallback : **TnewDailyCallback**  PriceBookCallback : **TPriceBookCallback**;  OfferBookCallback **: TOfferBookCallback;**  HistoryTradeCallBack **: THistoryTradeCallBack;**  ProgressCallBack : **TProgressCallBack**;  TinyBookCallBack **: TTinyBookCallBack**)  : **Short**; stdcall;   * **function** **DLLInitialize**(const pwcActivationKey : PWideChar;   StateCallback : **TStateCallBack**;  HistoryCallBack : **THistoryCallBack**;  AccountCallback : **TAccountCallback**;  NewDailyCallback : **TNewDailyCallback**;  PriceBookCallback : **TPriceBookCallback**;  OfferBookCallback : **TOfferBookCallback;**  HistoryTradeCallBack **: THistoryTradeCallBack;**  ProgressCallBack : **TProgressCallBack**;  TinyBookCallBack **: TTinyBookCallBack**) : **Short**; stdcall; **Deprecated**   * **function** **InitializeMarket**(const pwcActivationKey : PWideChar;   StateCallback : **TStateCallBack**;  NewDailyCallback : **TnewDailyCallback**  PriceBookCallback : **TPriceBookCallback**;  OfferBookCallback **: TOfferBookCallback;**  HistoryTradeCallBack **: THistoryTradeCallBack;**  ProgressCallBack : **TProgressCallBack**;  TinyBookCallBack **: TTinyBookCallBack**) : **Short**; stdcall; **Deprecated**   * **function** **DLLFinalize** : **ShortInt**; stdcall; * **function** **SubscribeTicker** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **UnsubscribeTicker** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **SubscribePriceBook** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **UnsubscribePriceBook** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **SubscribeOfferBook** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **UnsubscribeOfferBook** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **GetAccount** : ShortInt; stdcall; * **function** **GetHistoryTradesInInterval** (const pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**; dtDateStart, dtTimeStart : **PWideChar**) : **ShortInt**; stdcall; forward; [deprecated] * **function** **GetHistoryTrades**(const pwcTicker : **PWideChar**; const pwcBolsa : **PWideChar**; dtDateStart, dtDateEnd : **PWideChar**) : **ShortInt**; stdcall; * **function GetSerieHistory** (const pwcTicker : **PWideChar**; const pwcBolsa : **PWideChar**; dtDateStart, dtDateEnd : **PWideChar**; const nQuoteNumberStart, nQuoteNumberEnd : **Cardinal**) : **ShortInt**; stdcall; * **function** **SetChangeCotationCallback**(ChangeCotation : TChangeCotation) : **ShortInt**; stdcall; forward; * **function** **SetAssetListCallback**(AssetListCallback : TAssetListCallback) : **ShortInt**; stdcall; forward; * **function** **SetAssetListInfoCallback**(AssetListInfoCallback : TAssetListInfoCallback) : **ShortInt**; stdcall; forward; * **function** **SetAssetListInfoCallbackV2**(AssetListInfoCallbackV2 : TAssetListInfoCallbackV2) : **ShortInt**; stdcall; forward; * **function** **SetEnabledLogToDebug**(bEnabled : Integer) : **ShortInt**; stdcall; forward; * **function RequestTickerInfo**(const pwcTicker : PWideChar; const pwcBolsa : PWideChar) : **ShortInt**; stdcall; forward; * **function** **GetAllTicker**(pwcBolsa : PWideChar) : **ShortInt**; stdcall; forward; * **function** **SetChangeStateTickerCallback** (ChangeState : **TChangeStateTicker**) : **ShortInt**; stdcall; forward; * **function** **SetEnabledHistOrder** (bEnabled : Integer) : **ShortInt**; stdcall; forward; * **function** **SubscribeAdjustHistory** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **UnsubscribeAdjustHistory** (pwcTicker : **PWideChar;** pwcBolsa : **PWideChar**) : **ShortInt**; stdcall; * **function** **SetAdjustHistoryCallback** (AdjustHistory : **TAdjustHistoryCallback**) : **ShortInt**; stdcall; forward; * **function** **SetAdjustHistoryCallbackV2** (AdjustHistory : **TAdjustHistoryCallbackV2**) : **ShortInt**; stdcall; forward; * **function** **SetTheoreticalPriceCallback** (TheoreticalPrice : TTheoreticalPriceCallback) : **ShortInt**; stdcall; * **function** **SetServerAndPort** (const strServer, strPort : **PWideChar**) : **ShortInt**; stdcall; * **function** **GetServerClock** (var dtDate : **Double**;   var nYear, nMonth, nDay, nHour, nMin, nSec, nMilisec: **Integer**) : **ShortInt**; stdcall;   * **function** **GetLastDailyClose**(const pwcTicker, pwcBolsa: var dClose : **Double**; bAdjusted : **Integer**): **ShortInt**; stdcall |

Descrição das funções:

* **DLLInitialize / InitializeMarket**, pacActivationKey funções foram depreciadas por motivos de segurança. Quando utilizadas o valor de retorno é NL\_ERR\_INIT. Utilizar as funções **DLLInitializeLogin** e **DLLInitializeMarketLogin**.
* **DLLInitializeLogin** função de inicialização dos serviços de Market Data e Roteamento da DLL, pwcActivationKey é a chave de ativação fornecida para login. Os parâmetros pwcUser e pwcPassword representam o login e senha da conta em que a chave de ativação foi criada. Outros parâmetros correspondem à funções de callback utilizados pela DLL para transmitir os dados à aplicação cliente.
* **DLLInitializeMarketLogin** função de inicialização do serviço de Market Data, possui mesmo comportamento da função **DLLInitializeLogin**, porém não inicializa o serviço de roteamento.
* **DLLFinalize** é utilizada para finalização do serviço.
* **SetServerAndPort** é usado para conectar em servidores específicos do Market Data, precisa ser chamado antes da inicialização (DLLInitialize ou InitializeMarket). Os parâmetros strServer e strPort são o endereço do servidor e porta, respectivamente. **Importante:** apenas utilizar essa função com orientação da equipe de desenvolvimento, a DLL funciona da melhor maneira escolhendo os servidores internamente.
* **GetServerClock** 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>.

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** retorna o valor do fechamento (dClose) da sessão anterior ao dia atual. O parâmetro bAdjusted se for 0 retorna o valor não ajustado, caso contrário retorna o valor ajustado. Essa função retornará NL\_OK com os dados somente caso SubscribeTicker tenha sido chamada para o mesmo ativo. Ao chamar a função pela primeira vez, a função requisita dados ao servidor e retornar NL\_WAITING\_SERVER. Todas as chamadas subsequentes para o mesmo ativo retornam diretamente os dados já carregados. Ativos inválidos retornam NL\_ERR\_INVALID\_ARGS. Caso os dados da série diária ou ajustes não estejam previamente carregados, essa chamada causa seu carregamento e por consequência dispara os callbacks progressCallback e adjustHistoryCallback.
* **SubscribeTicker** é usado para receber as cotações em tempo real de determinado ativo. **UnsubscribeTicker** desativa este serviço
* **SubscribePriceBook** é utilizado para receber atualização do livro de preço. **UnsubscribePriceBook** desativa este serviço.
* **SubscribeOfferBook** é utilizado para receber atualização do livro de ofertas. **UnsubscribeOfferBook** desativa este serviço.
* Nos subscribers e unsubs das chamadas **SubscribeTicker,** **UnsubscribeTicker, SubscribePriceBook, UnsubscribePriceBook, SubscribeOfferBook, UnsubscribeOfferBook** 
  + pwcTicker é o nome do ativo para inscrição e
  + pwcBolsa é a bolsa referente ao ativo. Este deve seguir o padrão:
    - Ex: (**nome abreviado do ativo**), (**bolsa**)
    - Abreviação do ativo: **PETR4** ;pwcTicker = **PETR4**

Exemplo de bolsas pode ser encontrado no arquivo ‘**ConnectorInterfaceU’** , a seguir alguns exemplos:

* Bovespa = B
* BMF = F
* **GetHistoryTradesInInterval** é utilizado para solicitar as informações do histórico de um ativo a partir de uma data (
  + pwcTicker = ‘PETR4’;
  + dtDateStart = ‘06/08/2018’;
  + dtTimeStart= ‘10:00:00’) até o final do dia ou último trade realizado.
  + Retorno será dado na função de callback **ThistoryTradeCallBack**.
* **GetHistoryTrades** 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**.
  + Em **TProgressCallBack** o será retornado o progresso de Download (1 até 100), quando o progresso for igual a 1000 significa que todos trades foram enviadas para a aplicação.
* **GetSerieHistory** é utilizado para solicitar as informações do histórico de um ativo a partir de uma data e
  + QuoteNumber (
    - pwcTicker = ‘PETR4’;
    - dtDateStart = ‘06/08/2018 10:01:54’;
    - dtDateEnd= ‘06/08/2018 11:37:42’,
    - nQuoteNumberStart = 10,
    - nQuoteNumberEnd = 15420) .
    - O QuoteNumber é resetado a cada, começando em 10, e aumenta em incrementos de 10.
    - Retorno será dado na função de callback **ThistoryTradeCallBack**.
    - Em **TProgressCallBack** o será retornado o progresso de Download (1 até 100), quando o progresso for igual a 1000 significa que todos trades foram enviadas para a aplicação.
* **SetChangeCotationCallback** pode ser usado para definir uma função de callback do tipo TChangeCotation, esta função notifica sempre que o ativo sofrer modificação no preço.
* **SetAssetListCallback** pode ser usado para definir uma função de callback do tipo TAssetListCallback, responsável pelo retorno da informações de ativos.
* **SetAssetListInfoCallback** pode ser usado 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.
* **SetEnabledLogToDebug** pode ser usado para definir uma se a DLL deve salvar logs para debug.
  + 1 = salvar
  + 0 = Não salvar.
* **RequestTickerInfo** é utilizado para buscar novas informações do ativo (eg. ISIN). A reposta acontece pelo **TassetListInfoCallback, TAssetListInfoCallbackV2** e **TAssetListCallback**.
* **GetAllTicker** é utilizado para solicitação de informações de ativos de uma bolsa
  + {B=Bovespa,
  + F=BMF,
  + ‘’=TODAS}.
* **SetChangeStateTickerCallback** é utilizado para definir o callback **TchangeStateTicker** que informa as modificações do estado do ticker.
* **SubscribeAdjustHistory** é utilizado para receber histórico de ajustes do ativo determinado ticker.
* **UnsubscribeAdjustHistory**  é utilizado para desativar o recebimento de histórico de ajustes de determinado ticker.
* **SetAdjustHistoryCallback** é utilizado para definir o callback **TAdjustHistoryCallback** que informa o histórico de ajustes do ticker**.**
* **SetAdjustHistoryCallbackV2** é utilizado para definir o callback **TAdjustHistoryCallbackV2** que informa o histórico de ajustes do ticker.

Abaixo estão as funções presentes apenas ao utilizara inicialização da DLL com roteamento, a **DLLInitialize:**

* **GetAccount** função que retorna contas, retorno por callback(**AccountCallback**).
* **SetEnabledHistOrder** pode ser usado para desativar o histórico e update automático de ordens ao iniciar a aplicação.
  + 1 = Ativar /
  + 0 = Desativar.
  + Ao desativar o histórico, a aplicação não recebe os dados de ordens automaticamente ao inicializar, e por isso chamadas como GetPosition, onde é preciso montar a posição utilizando as operações não retornarão resultados válidos.
  + Para desativar o update automático utilizar esta função logo após a chamada de **DLLInitialize.**
  + (Usuário deve estar ciente que desativando o histórico seu controle de posição não será calculado corretamente pela plataforma. Além disso, as funcionalidades de zeragem e status da ordem podem ficar comprometidas).
* **SetTheoreticalPriceCallback** pode ser usado para definir a função de callback do tipo TTheoreticalPriceCallback, usado para receber o preço e quantidades teóricas durante o leilão.

Funções callback

WARNING: Não utilizar funções da dll dentro das funções de CALLBACK

|  |
| --- |
| Delphi   * **TStateCallBack** = **procedure**(nConnStateType, nResult : **Integer**) stdcall; * **TNewTradeCallback** = **procedure**( rAssetID : **TAssetIDRec**;  pwcDate : **PWideChar**;   nTradeNumber : **Cardinal;**  sPrice, sVol : **Double**;  nQtd, nBuyAgent, nSellAgent, nTradeType : **Integer**;  bEdit : **Char**) stdcall;   * **TNewDailyCallback** = **procedure**( rAssetID : **TAssetIDRec**;   pwcDate : **PWideChar**;  sOpen, sHigh, sLow, sClose, sVol, sAjuste,  sMaxLimit, sMinLimit, sVolBuyer, sVolSeller : **Double**;  nQtd, nNegocios, nContratosOpen,  nQtdBuyer, nQtdSeller, nNegBuyer, nNegSeller : **Integer**) stdcall;   * **TPriceBookCallback = procedure** ( rAssetID : **TAssetIDRec** ;   nAction , nPosition, Side, nQtds, nCount : **Integer**;  sPrice : **Double**;  pArraySell, pArrayBuy : **Pointer**) stdcall;   * **TOfferBookCallback** **= procedure** ( rAssetID : **TAssetIDRec** ;   nAction, nPosition, Side, nQtd, nAgent : **Integer**;  nOfferID : **Int64**;  sPrice : **Double**;  bHasPrice, bHasQtd,  bHasDate, bHasOfferID, bHasAgent : **Char**;  pwcDate : **PWideChar**;  pArraySell, pArrayBuy : **Pointer**) stdcall;   * **TAccountCallback = procedure** ( nCorretora : **Integer**;   CorretoraNomeCompleto, AccountID, NomeTitular : **PWideChar**) stdcall; forward;   * **THistoryCallBack** **= procedure** ( rAssetID : **TAssetIDRec**;   nCorretora , nQtd, nTradedQtd, nLeavesQtd, nSide : **Integer**;  sPrice, sStopPrice, sAvgPrice : **Double**;  nProfitID : **Int64**;  TipoOrdem, Conta, Titular, ClOrdID, Status, Date : **PWideChar**) stdcall;   * **THistoryTradeCallBack = procedure** ( rAssetID : **TAssetIDRec**;   pwcDate : **PWideChar**;  nTradeNumber : **Cardinal;**  sPrice, sVol : **Double**;  nQtd, nBuyAgent, nSellAgent, nTradeType : **Integer**) stdcall;   * **TProgressCallBack** **= procedure** (rAssetID : **TAssetIDRec**;   nProgress : **Integer**) stdcall;   * **TTinyBookCallBack = procedure** (rAssetID : **TAssetIDRec**;   sPrice : **Double**;  nQtd, nSide : **Integer**) stdcall;   * **TAssetListCallback** = **procedure** (rAssetID : **TAssetIDRec**;   pwcName : **PWideChar**) stdcall;   * **TAssetListInfoCallback** = **procedure** (rAssetID : **TAssetIDRec**;   pwcName, pwcDescription : **PwideChar;**  nMinOrderQtd, nMaxOrderQtd, nLote, stSecurityType, ssSecuritySubType : **Integer**;  sMinPriceIncrement, sContractMultiplier : **Double**;  strValidDate, strISIN : **PwideChar**;  ) stdcall;   * **TAssetListInfoCallbackV2** = **procedure** (rAssetID : **TAssetIDRec**;   pwcName, pwcDescription : **PwideChar;**  nMinOrderQtd, nMaxOrderQtd, nLote, stSecurityType, ssSecuritySubType : **Integer**;  sMinPriceIncrement, sContractMultiplier : **Double**;  strValidDate, strISIN, strSetor, strSubSetor, strSegmento : **PwideChar**;  ) stdcall;   * **TChangeStateTicker** = **procedure**(rAssetID : **TAssetIDRec**;   pwcDate : **PWideChar**;  nState : **Integer**) stdcall;   * **TAdjustHistoryCallback** = **procedure**( rAssetID : **TAssetIDRec**;   sValue : **Double**;  strAdjustType, strObserv, dtAjuste, dtDeliber, dtPagamento : **PwideChar;**  nAffectPrice : **Integer**) stdcall;   * **TAdjustHistoryCallbackV2** = **procedure**( rAssetID : **TAssetIDRec**;   dValue : **Double**;  strAdjustType, strObserv, dtAjuste, dtDeliber, dtPagamento : **PwideChar;**  nFlags : **Cardinal;**  dMult : **Double**) stdcall;   * **TTheoreticalPriceCallback** = **procedure** ( rAssetID : TAssetIDRec;   sTheoreticalPrice : Double;  nTheoreticalQtd : Int64) 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: 0 : connStLogin (Notify Login Change)
  + 1 : connStBroker (Notify Broker Change)
  + 2 : connStMarket (Notify Mercury Change)
  + 3 : connStActv (Notify Atctivation do Profit)

Já a entrada nResult recebe diferentes entradas de acordo com o tipo de conexão:

* + **connStMarket :** TConnMarketDataState ( conncsDisconnected = 0, conncsConnecting = 1, conncsConnectedWaiting = 2, conncsConnectedNotLogged = 3, conncsConnectedLogged = 4 );
  + **connStLogin** : TConnAuthenticationResult ( connArSuccess = 0, connArLoginInvalid = 1, connArPasswordInvalid = 2, connArPasswordBlocked = 3, connArPasswordExpired = 4, connArUnknown = 200);
  + **connStBroker :** TConnBrokerConnectionState ( connHcsDisconnected=0, connHcsConnecting=1, connHcsConnected=2, connHcsBrokerDisconnected=3, connHcsBrokerConnecting=4, connHcsBrokerConnected=5 );
  + **connStActv :** TConnActivationResult ( connActivatValid = 0, connActivatInvalid =1 );

* **bEdit** *é a informação se o trade que está sendo recebido é uma edição(informação da bolsa) ou um adição de trade, o ID para identificar um trade editado é o pwcDate .*
* **TNewDailyCallback** corresponde ao *Callback* para informar uma nova cotação.
  + pwcDate informa a data do *trade*, como uma string segundo o padrão ‘dd/mm/yyyy hh:mm:ss.zzz’.
  + sOpen indica o preço do *trade na abertura do mercado*.
  + sHigh indica o maior preço atingido pelo *trade.*
  + sLow indica o menor preço atingido pelo *trade.*
  + sClose indica último preço de fechamento de mercado do *trade.*
  + sVol indica o volume do *trade*.
  + sAjuste indica o ajuste do preço*.*
  + sMaxLimit e sMinLimit são os limites de preço em que há a possibilidade de efetuar um negócio.
  + sVolBuyer e sVolSellersão os volumes de compradores e vendedores, respectivamente, do trade.
  + nQtd indica a quantidade do *trade*.
  + nNegocios indica o número de negócios ocorridos.
  + nContratosOpen indica o número de contratos abertos.
  + nQtdBuyer e nQtdSellersão os números de compradores e vendedores, respectivamente, do trade.
  + nNegBuyer e nNegSellersão os números de negócios de compradores e vendedores, respectivamente, no trade.
* **TPriceBookCallback** 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).
  + sPrice: Preço; (Válido em atAdd).
  + pArraySell, pArrayBuy: Lista com livro 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 as ofertas existentes completo, 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
  + Quantidade de ofertas (**Q**) = 4 Bytes Integer
  + Tamanho do array = 4 Bytes Integer (deve ser usado em FreePointer)
  + **Q** entradas a serem inseridas no book, contendo
  + Preço = 8 Bytes Double
  + Quantidade = 4 Bytes Integer
  + Count = 4 Bytes Integer
* **TOfferBookCallback** 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 **GetAgentNameByID** e **GetAgentShortNameByID** já especificadas;
  + nOfferID: ID da oferta;
  + sPrice: Preço;
  + bHasPrice: 1 byte para especificar se existe(#1) preço;
  + bHasQtd: 1 byte para especificar se existe(#1) quantidade;
  + bHasDate: 1 byte para especificar se existe(#1) data;
  + bHasOfferID: 1 byte para especificar se existe(#1) oferta;
  + bHasAgent: 1 byte para especificar se existe(#1) agente,
  + pwcDate informa a data da oferta, como uma string segundo o padrão ‘dd/mm/yyyy hh:mm:ss.zzz’.
* O callback é tratado seguindo a mesma especificação do **TPriceBookCallback** , com exceção do layout dos arrays pArrayBuy e pArraySell:
  + Cabeçalho
  + Quantidade de ofertas (**Q**) = 4 Bytes Integer
  + Tamanho do array = 4 Bytes Integer (deve ser usado em FreePointer)
  + **Q** entradas a serem inseridas no book, contendo
  + Preço = 8 Bytes Double
  + Quantidade = 4 Bytes Integer
  + Agente = 4 Bytes Integer
  + Offer ID = 8 Bytes Int64
  + **T** tamanho string Data = 2 Bytes Short
  + Data da oferta = **T** Bytes
* **THistoryTradeCallBack** corresponde ao *Callback* de trades que foram solicitados a partir do **GetHistoryTradesInInterval[deprecated]/GetHistoryTrades .**
* **TProgressCallBack** corresponde ao *Callback* do progresso do **THistoryTradeCallBack**. rAssetID é o ativo que foi solicitado histórico;
  + nProgress é a porcentagem concluída. Quando o progresso for igual a 1000 significa que todos trades foram enviadas para a aplicação.
* **TTinyBookCallBack** 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 os Callbacks presentes apenas ao utilizara inicialização da DLL com roteamento, a DLLInitialize:

* **TAccountCallback** corresponde ao *Callback* para informar as contas existentes:
  + nCorretora: ID da corretora;
  + CorretoraNomeCompleto: Nome completo da corretora;
  + AccountID: ID da conta;
  + NomeTitula: Nome do titular da conta.
* **THistoryCallBack** corresponde ao *Callback* da solicitação de histórico de ordens:
  + nCorretora: ID da corretora;
  + nQtd: Quantidade da ordem;
  + nTradedQtd : Qtd já executada;
  + nLeavesQtd : Qtd restante para execução;
  + nSide: Lado da ordem (Compra=1, Venda=2);
  + sPrice: Preço da ordem;
  + sAvgPrice: Media do preço executado;
  + nProfitID: ID local para identificação da ordem;
  + Conta: ID da conta;
  + Titular: Nome do titular da conta;
  + ClOrdID: ID da ordem;
  + Status: Status da ordem;
  + Date: Data de execução da ordem;
  + TextMessage: Mensagem de informação.

Abaixo estão os callbacks inicializados por chamada de função:

* **TAssetListCallback** corresponde ao callback de solicitação de informação de ativos:
  + **rAssetID** informa a qual ativo pertence de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + **pwcName** é a descrição do ativo.
* **TChangeCotation** este callback é usado para informar quando ocorrer uma modificação de preço no ativo, informando qual foi o último preço e hora da negociado.
  + rAssetID informa a qual ativo pertence de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + **pwcDate** informa a data que ocorreu o trade que alterou o preço.
  + **sprice** indica qual é o último preço negociado.
* **TChangeStateTicker** corresponde ao *Callback* de identificação de alteração de estado do ativo.
  + rAssetID informa o ativo de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + PwcDate : é a data que houve modificação do estado, apenas alguns estados mostram a data.
  + nState : informa o estado atual.
  + Exemplo de tipos abaixo:
    - TypeChangeState = (tcsFrozen=2, tcsClosed=6, tcsOpened=0, tcsInhibited=3, tcsAuctioned=4, tcsPreClosing=10, tcsPreOpening=13);
* **TAdjustHistoryCallback** corresponde ao *Callback* de ajustes de um ativo.
  + rAssetID informa o ativo de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + sValue : é o valor do ajuste.
  + strAdjustType : tipo de ajuste.
  + strObserv : Observação.
  + dtAjuste : data do ajuste.
  + dtDeliber : Data de deliberação.
  + DtPagamento : Data do pagamento.
  + nAffectPrice quando 1 indica que o ajuste afeta o preço, quando 0 não.
* **TAdjustHistoryCallbackV2** corresponde ao *Callback* de ajustes de um ativo.
  + rAssetID informa o ativo de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + dValue : é o valor do ajuste.
  + strAdjustType : tipo de ajuste.
  + strObserv : Observação.
  + dtAjuste : data do ajuste.
  + dtDeliber : Data de deliberação.
  + DtPagamento : Data do pagamento.
  + nFlags é um campo de bits b0 a b31, onde o bit **0** 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.
* **TAssetListInfoCallback** corresponde ao *Callback* de informações de ativos.
  + rAssetID informa o ativo de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + strDescription : nome do ativo.
  + nMinOrderQtd: mínimo de ordens permitido.
  + nMaxOrderQtd: Máximo de ordens permitido.
  + nLote: Tamanho do lote.
  + strDate: data de validade.
  + strSetor: Setor de atuação.
  + strSubSetor: Subsetor de atuação.
  + strSegmento: Segmento de atuação.
  + stSecurityType: Tipo de ativo conforme lista abaixo:
    - stFuture = 0, stSpot = 1, stSpotOption = 2, stFutureOption = 3,
    - stDerivativeTerm = 4,stStock = 5, stOption = 6, stForward = 7,
    - stETF = 8, stIndex = 9, stOptionExercise = 10, stUnknown=11,
    - stEconomicIndicator = 12, stMultilegInstrument = 13, stCommonStock = 14,
    - stPreferredStock = 15, stSecurityLoan = 16, stOptionOnIndex = 17,
    - stRights = 18, stCorporateFixedIncome = 19
* ssSecuritySubType: subtipo do ativo conforme lista abaixo:
  + ssFXSpot = 0, ssGold = 1, ssIndex = 2, ssInterestRate = 3,
  + ssFXRate = 4, ssForeignDebt = 5, ssAgricultural = 6, ssEnergy=7,
  + ssEconomicIndicator = 8, ssStrategy = 9, ssFutureOption = 10,
  + ssVolatility = 11, ssSwap = 12, ssMiniContract = 13,
  + ssFinancialRollOver = 14, ssAgriculturalRollOver=15,
  + ssCarbonCredit = 16, ssUnknown = 17, ssFractionary = 18,
  + ssStock = 19, ssCurrency = 20, ssOTC = 21,//OTC=MercadoBalcao
  + ssFII = 22,// FII=Fundo de Investimento Imobiliario
  + ssOrdinaryRights = 23,//(DO)
  + ssPreferredRights = 24,//(DP)
  + ssCommonShares = 25,//(ON)
  + ssPreferredShares = 26,//(PN)
  + ssClassApreferredShares = 27,//(PNA)
  + ssClassBpreferredShares = 28,//(PNB)
  + ssClassCpreferredShares = 29,//(PNC)
  + ssClassDpreferredShares = 30,//(PND)
  + ssOrdinaryReceipts = 31,//(ON REC)
  + ssPreferredReceipts = 32,//(PN REC)
  + ssCommonForward = 33,
  + ssFlexibleForward = 34,
  + ssDollarForward = 35,
  + ssIndexPointsForward = 36,
  + ssNonTradeableETFIndex = 37,
  + ssPredefinedCoveredSpread = 38,
  + ssTraceableETF = 39,
  + ssNonTradeableIndex = 40,
  + ssUserDefinedSpread = 41,
  + ssExchangeDefinedspread = 42,
  + ssSecurityLoan = 43,
  + ssTradeableIndex = 44,
  + ssOthers = 45
* **TAssetListInfoCallbackV2** extensão do callback anterior, adiciona os campos
  + strSetor: Setor de atuação.
  + strSubSetor: Subsetor de atuação.
  + strSegmento: Segmento de atuação.
* **TTheoreticalPriceCallback** corresponde ao callback para retorno do preço e quantidades teóricas durante o leilão de um ativo.
  + rAssetID informa a qual ativo pertence de acordo com a estrutura
  + **TAssetIDRec** já especificada.
  + dTheoreticalPrice corresponde ao preço teórico.
  + nTheoreticalQtd corresponde à quantidade teórica.

Uso do Produto

**Inicializando DLL com roteamento**

Para uso da Biblioteca, é fundamental invocar a função **InitializeDLL**, informando sua chave de ativação e as funções para *callback*. Outras funções podem ser invocadas apenas após um **InitializeDLL** sucedido. Para melhor compreensão, consulte os exemplos fornecidos em *‘DLL e Executavel’ , ‘Exemplo C++’ e ‘Exemplo Delphi’*. O ‘*DLL\_ROTEAMENTO\_MAIN*’ *e*xplora todas as funcionalidades oferecidas através de uma aplicação Delphi gráfica. Estes exemplos também se encontram como executáveis em *‘Exemplo C++’ e ‘Exemplo Delphi’*, lembrando que necessitam da *DLL* para executarem (*ProfitDLL.dll*).

**Inicializando DLL apenas com Market Data**

Para uso da Biblioteca, é fundamental invocar a função **InitializeMarket**, informando sua chave de ativação e as funções para callback. Outras funções podem ser invocadas após o **InitializeMarket** inicializado com sucesso. Para melhor compreensão, consulte os exemplos fornecidos em ‘DLL e Executavel’ , ‘Exemplo C++’ e ‘Exemplo Delphi’. O ‘DLL\_MARKET’ explora todas as funcionalidades oferecidas através de uma aplicação Delphi gráfica. Estes exemplos também se encontram como executáveis em ‘Exemplo C++’ e ‘Exemplo Delphi’, lembrando que necessitam da DLL para executarem (ProfitDLL.dll).

**Conversão de tipos Delphi para C**

A tabela no link a seguir apresenta equivalência de tipos entre as duas linguagens:  
<http://docwiki.embarcadero.com/RADStudio/Tokyo/en/Delphi_to_C%2B%2B_types_mapping>

**Conversão de tipos C para Python**

A tabela no link a seguir apresenta equivalência de tipos entre as duas linguagens:  
<https://docs.python.org/2/library/ctypes.html>

**Conversão de tipos Delphi para C#**

A tabela no link a seguir apresenta equivalência de tipos entre as duas linguagens:  
<http://www.netcoole.com/delphi2cs/datatype.htm>

Obs: Desabiliar a propriedade “Apenas Meu Código”, e compilar para x86