1. Introdução:

Caro desenvolvedor. Essa biblioteca foi desenvolvida para facilitar a implementação dos equipamentos Companytec em seu software. Disponibilizando todas as funções necessárias para a integração, nossa DLL já está presente em aproximadamente 40% dos sistemas compatíveis com nossa solução.

2. Compatibilidade:

A fim de atender as linguagens mais utilizadas na atualidade, nossa DLL possui funções que se adaptam à sua linguagem, para que isso seja possível, existem várias funções que executam o mesmo trabalho, diferenciando-se apenas na sua maneira de passagem de parâmetros e retornos.

Entre as linguagens que já utilizam a DLL podemos citar:

- Delphi
- Visual Basic
- Fox Pro
- COBOL for Windows.

3. Funções:

a. Function InicializaSerial(np:byte): Boolean;

i. Função que abre a porta de comunicações para envio e recebimento de comandos. Essa função só necessita ser chamada uma vez, no inicio da aplicação que irá comunicar com nosso equipamento.

ii. Parâmetros de entrada:
   1. Variável: np
      a. Tipo: byte
      b. Dados: número da porta serial

iii. Retorno da função:
   1. Tipo: Boolean
   2. Dados: "True" se conectou ou "False" se não foi possível conectar;

b. Function AlteraPreco(bico:string;preco:double;decimais:byte):error;

i. Função utilizada para alterar o preço unitário de um bico do sistema. Essa função é enviada diretamente para o bico solicitado, o retorno deve ser analisado para confirmar a operação.

ii. Parâmetros de entrada:
   1. Variável: bico
      a. Tipo: string
      b. Dados: código do bico que deseja-se alterar seu preço
   2. Variável: preço
      a. Tipo: Double
      b. Dados: Valor para o qual desejamos alterar o preço unitário
   3. Variável: decimais
      a. Tipo: byte
      b. Dados: Número de casas decimais que o bico possui no preço unitário

iii. Retorno da função:
   1. Tipo: ERROR
   2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

c. Function AutoLibera(bico:string):Error;

i. Finalidade: Coloca o bico desejado em estado de Auto Liberação, esse estado permitirá que o bico abasteça sempre que requisitado.
ii. Parâmetros de entrada:
    1. Variável: Bico
        a. Tipo: String
        b. Dados: Código do bico desejado
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

d. Function AutorizaAbast(bico:string):Error;

i. Finalidade: Autoriza um abastecimento para o bico informado. Após o abastecimento o bico retorna para seu estado anterior.
ii. Parâmetros de entrada:
    1. Variável: Bico
        a. Tipo: String
        b. Dados: Código do bico desejado
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

e. Function BloqueiaBico(bico:string):Error;

i. Finalidade: Bloquear abastecimentos em um determinado bico.
ii. Parâmetros de entrada:
    1. Variável: Bico
        a. Tipo: String
        b. Dados: Código do bico desejado
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

f. Function CobAlteraPreco(a:PChar):error;

i. Finalidade: Alterar o valor unitário praticado por um determinado bico.
ii. Parâmetros de entrada:
    1. Variável: a
        a. Tipo: PChar
        b. Dados: "BBPPPP", onde BB é o código do bico, e PPPP é o novo preço a ser praticado.
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

g. Procedure CobLeEnc(var a:enc);

i. Finalidade: Ler o totalizador (encerrante) de determinado bico.
ii. Parâmetros de entrada:
    1. Variável: a
        a. Tipo: ENC
        b. Dados: ENC é uma estrutura definida na DLL, nela constam os campos:
            i. Bico: código do bico a ser consultado;
            ii. Tipo: "$" para totais em dinheiro, "L" para totais em volume;
            iii. Valor: campo que receberá, por referencia, os dados resultantes da solicitação;
iii. Retorno da função:
    1. Tipo: a.valor (por referencia)
    2. Dados: A DLL informará o valor do encerrante no campo "valor" da estrutura passada por referencia.

h. Function CobLePPL(var a:PChar):error;

i. Finalidade: Ler o preço unitário de determinado bico.
ii. Parâmetros de entrada:
    1. Variável: a
        a. Tipo: PChar
        b. Dados: Quando chamada a função, devemos informar "BB" na variável "a", significando o código do bico que desejamos consultar. Após finalizada, a função retornará "PPPP" na variável "a", significando o preço unitário lido da bomba.
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

i. Procedure CobLeStructIDSt(var ab:abast3);

i. Finalidade: Ler o abastecimento realizado mediante identificação.
ii. Parâmetros de entrada:
    1. Variável: ab
        a. Tipo: Abast3
        b. Dados: "Abast3" é uma estrutura que contem dados provenientes do abastecimento ocorrido. O campo "value" informa se a informação contida na estrutura é um abastecimento (TRUE) ou se a informação deve ser descartada (FALSE). Essa estrutura está descrita no final desse documento.
iii. Retorno da função:
    1. Tipo: Abast3 (por referencia)
    2. Dados: A DLL irá definir o campo "value" como TRUE e preencherá os campos da estrutura fornecida, com os dados do abastecimento ocorrido, ou definir o campo "value" como FALSE, caso não exista abastecimento a ser lido.

j. Procedure CobLeStructSt(var ab:abast2);

i. Finalidade: Ler o abastecimento realizado sem a informação de identificação.
ii. Parâmetros de entrada:
    1. Variável: ab
        a. Tipo: Abast2
        b. Dados: "Abast2" é uma estrutura que contem dados provenientes do abastecimento ocorrido. O campo "value" informa se a informação contida na estrutura é um abastecimento (TRUE) ou se a informação deve ser descartada (FALSE). Essa estrutura está descrita no final desse documento.
iii. Retorno da função:
    1. Tipo: Abast2 (por referencia)
    2. Dados: A DLL irá definir o campo "value" como TRUE e preencherá os campos da estrutura fornecida, com os dados do abastecimento ocorrido, ou definir o campo "value" como FALSE, caso não exista abastecimento a ser lido.

k. Procedure CobLeVis(var st:visualizacao);

i. Finalidade: Ler a andamento dos abastecimentos em progresso.
ii. Parâmetros de entrada:
    1. Variável: st
        a. Tipo: visualizacao
        b. Dados: "visualizacao" é uma estrutura que contem dados provenientes do abastecimento em andamento em um campo único chamado "stfull". Essa estrutura está descrita no final desse documento.
iii. Retorno da função:
    1. Tipo: visualizacao (por referencia)
    2. Dados: A DLL retornará no campo "stfull" uma STRING de tamanho variável de acordo com o número de bicos que estiverem abastecendo no momento, no seguinte formato: "BBLLLLLLBBLLLLLL.....BBLLLLLL", onde "BB" é o número do bico, e "LLLLLL" é a sua litragem atual.

l. Function CobPreset(a:PChar):Error;

i. Finalidade: Pré determinar a quantidade de produto máxima que um bico poderá fornecer em um abastecimento.
ii. Parâmetros de entrada:
    1. Variável: a
        a. Tipo: PChar
        b. Dados: "BBLLLLLL", onde "BB" é o bico que queremos pré determinar, e "LLLLLL" é a quantidade de produto máxima a ser abastecida pelo mesmo no próximo abastecimento.
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

m. Function CobSetClock(par:PChar):boolean;

i. Finalidade: Ajustar o relógio do equipamento;
ii. Parâmetros de entrada:
    1. Variável: par
        a. Tipo: PChar
        b. Dados: "AUTO" para ajuste automático utilizando horário atual do computador ou "ddhhmm" para ajuste manual, representando dia, hora e minuto.
iii. Retorno da função:
    1. Tipo: Boolean
    2. Dados: TRUE para ajuste bem sucedido, ou FALSE caso contrário.

n. Function ConsultaEncerrante(modo:char;bico:string):Encerrante;

i. Finalidade: Consultar o totalizador (encerrante) de um determinado bico;
ii. Parâmetros de entrada:
    1. Variável: modo
        a. Tipo: char
        b. Dados: "$" para leitura do totalizador em dinheiro, ou "L" para leitura do totalizador em volume;
    2. Variável: bico
        a. Tipo: String
        b. Dados: Bico a ser consultado;
iii. Retorno da função:
    1. Tipo: Encerrante
    2. Dados: "Encerrante" é uma estrutura definida na DLL Companytec, esta, possui os campos "bico", onde retornará o código do bico lido, e "valor", que retornará o valor proveniente do pedido, em dinheiro ou volume.

o. Function EnviaComando(comando:string;timeout:cardinal):Pchar;

i. Finalidade: Enviar um comando genérico para o concentrador.
ii. Parâmetros de entrada:
    1. Variável: comando
        a. Tipo: String
        b. Dados: Comando em conformidade com o protocolo de comunicação Companytec.
    2. Variável: Timeout
        a. Tipo: Cardinal
        b. Dados: O tempo máximo que a DLL poderá aguardar pela resposta do comando.
iii. Retorno da função:
    1. Tipo: PChar
    2. Dados: Ponteiro para caracteres (em algumas linguagens poderemos utilizar STRING) onde estará armazenado o resultado do comando enviado.

p. Function FechaSerial: DWORD;

i. Finalidade: Fechar a porta serial de comunicação.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
   1. Tipo: DWORD; (Também representado por "Longword", é um inteiro de 4 bytes)
   2. Dados: Retornará 0 (Zero) se foi executado com sucesso, ou 1 (um) se não obteve sucesso na tentativa de fechar a porta de comunicação.

q. Function FechaSerialVB: boolean;

i. Finalidade: Fechar a porta serial de comunicação.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
   1. Tipo: Boolean;
   2. TRUE, se obteve sucesso na tentativa de fechar a porta de comunicação, ou FALSE, caso contrário.

r. Function FidAciona(endereco:string;minutos,segundos:byte):integer;

i. Finalidade: Acionar uma saída analógica (máquina de lavagem, fechadura elétrica, etc) por tempo determinado.

ii. Parâmetros de entrada:
   1. Variável: endereco
      a. Tipo: String;
      b. Dados: Código da saída a ser acionada;
   2. Variável: minutos;
      a. Tipo: byte; (inteiro de 8 bits)
      b. Dados: Tempo em minutos que a saída manter-se-á acionada;
   3. Variável: segundos;
      a. Tipo: byte; (inteiro de 8 bits)
      b. Dados: Tempo em segundos que a saída manter-se-á acionada;

iii. Retorno da função:
   1. Tipo: Integer;
   2. Dados: A função retornará um código numérico de acordo com a execução do comando. Sendo eles:
      a. 1: Comando executado com sucesso;
      b. 2: Atingido limite máximo de espera pelo resultado do comando;
      c. 3: O endereço informado nos parâmetros de entrada esteja incorreto;
      d. 4: O caractere de modo, informado no parâmetros de entrada está incorreto;
      e. 0: Ocorreu um erro desconhecido.

s. Function FidIdent:IFid;

i. Finalidade: Ler as informações de cartões reconhecidos e não presentes na memória do concentrador nos sensores Identfid®.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: IFid;
    2. Dados: Estrutura que fornece os dados da identificação do cartão nos sensores Identfid®, como código do cartão, horário e endereço da leitura, etc. (veja a definição do tipo IFid no final desse documento).

t. Procedure FidIncrementa;

i. Finalidade: Informar ao concentrador Identfid® que a informação atual já foi lida e manipulada, podendo o mesmo, passar para a próxima identificação.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função: Nenhum;

u. Procedure FidIncrementaAbast;

i. Finalidade: Informar ao concentrador Identfid® (Vr. 2), que o abastecimento atual já foi lido e manipulado, podendo o mesmo, passar a informar o próximo abastecimento, quando um for solicitado.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função: Nenhum;

v. Function FidLeRegistro(nro:integer):PChar;

i. Finalidade: Informar um abastecimento específico presente na memória do Identfid® (Vr. 2).
ii. Parâmetros de entrada:
    1. Variável: nro;
        a. Tipo: Integer;
        b. Dados: A posição do abastecimento desejado na memória do equipamento;
iii. Retorno da função:
    1. Tipo: PChar;
    2. Dados: Informação completa do abastecimento presente na posição requerida (consulte protocolo).

w. Function FidModo(endereco:string;option:char):integer;

i. Finalidade: Alterar o modo de funcionamento do sensor Identfid®.
ii. Parâmetros de entrada:
    1. Variável: endereço;
        a. Tipo: String;
        b. Dados: Código do endereço a ser alterado;
    2. Variável: option;
        a. Tipo: Char;
        b. Dados:
            i. "B": Para colocar sensor em modo de bloqueio, acendendo a luz vermelha para o operador;
            ii. "L": Para liberar o sensor para abastecimento, acendendo a luz verde para o operador;
iii. Retorno da função:
    1. Tipo: Integer;
    2. Dados: Retorna um valor numérico de acordo com a execução da função:
        a. 1: Comando executado com sucesso;
        b. 2: Atingido limite máximo de espera pelo resultado do comando;
        c. 3: O endereço informado nos parâmetros de entrada esteja incorreto;
        d. 4: O caractere de modo, informado no parâmetros de entrada está incorreto;
        e. 0: Ocorreu um erro desconhecido.

x. Function FidSetClock(dia,hora,minuto:byte):integer;

i. Finalidade: Atualizar o relógio do Identfid®;
ii. Parâmetros de entrada:
    1. Variável: dia;
        a. Tipo: Byte (8 bits)
        b. Dados: Dia atual;
    2. Variável: hora;
        a. Tipo: Byte (8 bits)
        b. Dados: Hora atual;
    3. Variável: minuto;
        a. Tipo: Byte (8 bits)
        b. Dados: Minuto atual;
iii. Retorno da função:
    1. Tipo: Integer;
    2. Dados: Valor numérico representando a execução da função;
        a. 1: Comando executado com sucesso;
        b. 0: Falhou ao atualizar relógio;

y. Function FidStatus:StFid;

i. Finalidade: Ler a situação atual dos sensores Identfid®;
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
   1. Tipo: stFid;
   2. Dados: Estrutura contendo 1 campo do tipo "string" chamado "status", onde serão informados caracteres de acordo com a situação atual de cada sensor. (veja a definição do tipo "stFid" no final desse documento).

z. Procedure Incrementa;

i. Finalidade: Informar ao concentrador que o abastecimento atual já foi lido e armazenado.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função: Nenhum;

aa. Function InicializaLogSerial(np:byte;LogFile:string):boolean;

i. Finalidade: Abre a porta de comunicações e cria um arquivo texto contendo um LOG de comandos e respostas;
ii. Parâmetros de entrada:
   1. Variável: np
      a. Tipo: Byte (8 bits)
      b. Dados: Porta serial onde está conectado o equipamento;
   2. Variável: LogFile
      a. Tipo: String
      b. Dados: Caminho para o arquivo que irá armazenar os dados trafegados na porta serial;
iii. Retorno da função:
   1. Tipo: Boolean
   2. Dados: Retorna TRUE se a porta serial foi aberta com sucesso, ou FALSE, caso contrário.

bb. Function LeAbastecimento:abast;

i. Finalidade: Ler o abastecimento atual em memória e passar o ponteiro de leitura de abastecimentos para o próximo;
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
   1. Tipo: abast
   2. Dados: Retorna os dados do abastecimento, como preço total, volume abastecido, número do bico, etc. em uma estrutura definida na DLL. (veja a definição do tipo "abast" no final desse documento).

cc. Function LeAbastecimentoFid:AbastFid;

i. Finalidade: Ler o abastecimento atual em memória juntamente com sua identificação Identfid® e passar o ponteiro de leitura de abastecimentos para o próximo;
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: abastFid
    2. Dados: Retorna os dados do abastecimento, como preço total, volume abastecido, número do bico, etc. juntamente com o código do cartão Identfid® que autorizou o abastecimento em uma estrutura definida na DLL. (veja a definição do tipo "AbastFid" no final desse documento).

dd. Function LeAbFix:abast;

i. Finalidade: Ler o abastecimento atual em memória sem passar o ponteiro de leitura de abastecimentos para o próximo;
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: abast
    2. Dados: Retorna os dados do abastecimento, como preço total, volume abastecido, número do bico, etc. em uma estrutura definida na DLL. (veja a definição do tipo "abast" no final desse documento).

ee.function LeEvento(indice:integer):string;

i. Finalidade: Ler o evento ocorrido na CBC-06
ii. Parâmetros de entrada:
    1. Variável: índice
        a. Tipo: integer (0-10000)
        b. Dados: Número do registro a ser lido. A CBC-06 armazena até 10000 registros em memória;
iii. Retorno da função:
    1. Tipo: String
    2. Dados: Retorna a string de registro de evento padrão CBC-06 (consulte protocolo CBC-06);

ff. Function LePart(option:char):PChar;

i. Finalidade: Ler uma parte do abastecimento atual.
ii. Parâmetros de entrada:
    1. Variável: option
        a. Tipo: Char
        b. Dados: Caractere de controle da opção a ser lida:
            i. L: Volume abastecido;
            ii. T: Total a pagar;
            iii. P: Preço unitário;
            iv. C: Calendário;
            v. E: Encerrantes (totalizador);
iii. Retorno da função:
    1. Tipo: String
    2. Dados: Retorna o código de bico juntamente com a informação requisitada. Por exemplo: Para um pedido "P", teremos como resultado: BB – PPPP, onde "BB" é o código do bico, e "PPPP" é o preço unitário praticado no abastecimento atual;

gg. Function LePPL(bico:string):real;

i. Finalidade: Ler o preço unitário praticado em determinado bico.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: O código do bico a ser consultado;
iii. Retorno da função:
    1. Tipo: real; (8 bytes)
    2. Dados: Retorna o valor unitário praticado no bico requisitado;

hh. Function LeRegistro(NumReg:integer):abast;

i. Finalidade: Ler um abastecimento presente na memória do equipamento, informando sua posição.
ii. Parâmetros de entrada:
    1. Variável: NumReg
        a. Tipo: Integer;
        b. Dados: A posição do abastecimento na memória da placa;
iii. Retorno da função:
    1. Tipo: abast;
    2. Dados: Retorna os dados do abastecimento, como preço total, volume abastecido, número do bico, etc. em uma estrutura definida na DLL. (veja a definição do tipo "abast" no final desse documento).

ii. Function LeSerial(desc:string;timeout:cardinal):Pchar;

i. Finalidade: Ler a resposta do concentrador para um determinado comando.
ii. Parâmetros de entrada:
    1. Variável: desc
        a. Tipo: String;
        b. Dados: O comando a ser enviando para o concentrador;
    2. Variável: timeout
        a. Tipo: Cardinal; (unsigned 32 bits)
        b. Dados: Tempo máximo de espera pela resposta;
iii. Retorno da função:
    1. Tipo: PChar;
    2. Dados: Ponteiro para caractere onde se encontra a resposta do concentrador para o comando enviado;

jj. Function LeStatus:multistatus;

i. Finalidade: Ler a situação atual das bombas.
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
    1. Tipo: multistatus
    2. Dados: Estrutura de dados de 48 posições, que contem o atual estado das bombas conectadas ao concentrador. (veja a definição do tipo "multistatus" no final desse documento).

kk. Function LeStatusFid:multistatus;

i. Finalidade: Ler a situação atual dos sensores Identfid®.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: multistatus
    2. Dados: Estrutura de dados de 48 posições, que contem o atual estado dos sensores conectados ao concentrador. (veja a definição do tipo "multistatus" no final desse documento).

ll. Function LeStatusVB():StStatus2;

i. Finalidade: Ler a situação atual das bombas conectadas no concentrador.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: StStatus2; (pré definida na DLL)
    2. Dados: Estrutura de dados de 48 posições no formato STRING, que contem o atual estado das bombas conectadas no concentrador. (veja a definição do tipo "StStatus2" no final desse documento).

mm. Function LeSTEncerrante(modo:string;bico:string):PChar;

i. Finalidade: Ler o totalizador do bico desejado.
ii. Parâmetros de entrada:
    1. Variável: modo
        a. Tipo: String
        b. Dados: "$" para leitura em dinheiro ou "L" para leitura em volume;
    2. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico que desejamos consultar.
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para String)
    2. Dados: "VVVVVV,VV" contendo o valor do totalizador ou "FALHA", caso a DLL não obtiver sucesso na solicitação.

nn. Function LeStReduzida:PChar;

i. Finalidade: Ler o abastecimento atual em memória.
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para String)
    2. Dados: Abastecimento atual da memória. (consulte protocolo)

oo. Function LeStRegistro(NumReg:integer):PChar;

i. Finalidade: Ler o abastecimento, informando sua posição na memória do concentrador.
ii. Parâmetros de entrada:
    1. Variável: NumReg
        a. Tipo: Integer
        b. Dados: Número do registro de abastecimento desejado.
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para String)
    2. Dados: Abastecimento padrão Companytec® de 52 ou 75 caracteres. (consulte protocolo)

pp. Function LeStringAb(var resposta:PChar):PChar;

i. Finalidade: Ler o abastecimento atual da memória do concentrador.
ii. Parâmetros de entrada:
    1. Variável: resposta
        a. Tipo: PChar (ponteiro para String)
        b. Dados: Variável que irá receber a informação do abastecimento.
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para String)
    2. Dados: Abastecimento padrão Companytec® de 52 ou 75 caracteres. (consulte protocolo)

qq. Function LeStringAbVB:PChar;

i. Finalidade: Ler o abastecimento atual da memória do concentrador.
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para String)
    2. Dados: Abastecimento padrão Companytec® de 52 ou 75 caracteres. (consulte protocolo)

rr. Procedure LeStringX(var resposta:retorno2);

i. Finalidade: Ler o abastecimento atual da memória do concentrador, retornando em uma estrutura definida na DLL.
ii. Parâmetros de entrada:
    1. Variável: resposta
        a. Tipo: retorno2 (estrutura de dados pré-definida na DLL)
        b. Dados: Abastecimento atual da memória, informado na estrutura "retorno2". (veja a definição do tipo "retorno2" no final desse documento).
iii. Retorno da função:
    1. Tipo: retorno2 (por referência)
    2. Dados: Abastecimento disposto na estrutura "retorno2".

ss. Function LeStructEncerrante(modo:string;bico:string):stEncerrante;

i. Finalidade: Ler o totalizador de determinado bico, informando-o em uma estrutura pré-definida na DLL;
ii. Parâmetros de entrada:
    1. Variável: modo
        a. Tipo: String
        b. Dados: "$" para leitura em dinheiro ou "L" para leitura em volume;
    2. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico que desejamos consultar.
iii. Retorno da função:
    1. Tipo: stEncerrante (pré-definida na DLL)
    2. Dados: Totalizador do bico, informado na estrutura stEncerrante. (veja a definição do tipo "stEncerrante" no final desse documento).

tt. Function LeStructPPL(bico:string):stPPL;

i. Finalidade: Ler o preço unitário praticado por um determinado bico;
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String;
        b. Dados: Código do bico a ser consultado;
iii. Retorno da função:
    1. Tipo: stPPL (pré-definida na DLL)
    2. Dados: Preço unitário e bico consultado. (veja a definição do tipo "stPPL" no final desse documento).

uu. Procedure LeStructSt(var ab:abast2);

i. Finalidade: Ler o abastecimento da posição atual do concetrador;
ii. Parâmetros de entrada:
    1. Variável: ab
        a. Tipo: abast2. (estrutura pré-definida na DLL);
        b. Dados: Abastecimento disposto na estrutura;
iii. Retorno da função:
    1. Tipo: abast2 (pré-definida na DLL)
    2. Dados: Abastecimento completo, armazenado na estrutura "abast2". (veja a definição do tipo "abast2" no final desse documento).

vv. Function LeStStatus:StStatus;

i. Finalidade: Ler a situação atual dos bicos instalados no concentrador;
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
   1. Tipo: StStatus (pré-definida na DLL)
   2. Dados: Estrutura definida com 1 campo String de 100 caracteres. (veja a definição do tipo "StStatus" no final desse documento).

ww. Function LeStStatus2:PChar;

i. Finalidade: Ler a situação atual dos bicos instalados no concentrador;
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
   1. Tipo: PChar (ponteiro para string)
   2. Dados: Informação representada em string no formato de status Companytec®. (consulte protocolo)

xx. Function LeVisualizacao():OnLine;

i. Finalidade: Ler o volume que os bicos estão abastecendo no momento da consulta;
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
   1. Tipo: OnLine (estrutura pré-definida na DLL)
   2. Dados: Estrutura composta de 48 posições, contendo código de bico e o volume que o mesmo está abastecendo. As posições não utilizadas retornarão no campo "bico", o valor "00". (veja a definição do tipo "OnLine" no final desse documento).

yy. Procedure LimpaSerial;

i. Finalidade: Limpa as informações do BUFFER da serial.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função: Nenhum.

zz. Function ParaBomba(bico:string):Error;

i. Finalidade: Interromper o abastecimento atual, cortando o fluxo de combustível da mangueira. (função não suportada por todas as bombas)
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser parado.
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

aaa. Function PortOpen:boolean;

i. Finalidade: Verificar a situação atual da porta de comunicações;
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: Boolean;
    2. Dados: "TRUE" caso a porta de comunicações esteja aberta, ou "false", caso contrario.

bbb. Function Preset(bico:string;valor:double):Error;

i. Finalidade: Determinar o valor máximo da próxima venda, em um determinado bico; (função não suportada por todas as bombas).
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser parado.
    2. Variável: valor
        a. Tipo: Double (ponto flutuante de 64 bits)
        b. Dados: Valor máximo da próxima venda.
iii. Retorno da função:
    1. Tipo: ERROR
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

ccc. Function ReadSerial(timeout:cardinal):PChar;

i. Finalidade: Ler o conteúdo presente na porta serial de comunicação.
ii. Parâmetros de entrada:
    1. Variável: timeout
        a. Tipo: Cardinal (unsigned 32 bits)
        b. Dados: Tempo máximo de espera por dados.
iii. Retorno da função:
    1. Tipo: PChar (ponteiro para caractere)
    2. Dados: Informação lida da porta serial.

ddd. Procedure RefAltPreco(bico:string;preco:double;decimais:byte;var status:error);

i. Finalidade: Alterar o preço unitário de um determinado bico.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser alterado.
    2. Variável: preco
        a. Tipo: Double (ponto flutuante de 64 bits)
        b. Dados: Novo preço a ser praticado.
    3. Variável: decimais
        a. Tipo: Byte (unsigned 8 bits)
        b. Dados: Número de casas decimais do preço unitário da bomba.
    4. Variável: status
        a. Tipo: ERROR (variável definida na DLL)
        b. Dados: Passar a variável de retorno por referência, nesse caso, uma variável do tipo ERROR, que receberá o resultado da função após sua execução.
iii. Retorno da função:
    1. Tipo: ERROR (por referencia)
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

eee. Procedure RefAutoLibera(bico:string;var status:error);

i. Finalidade: Autorizar os abastecimentos automaticamente.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser autorizado automaticamente.
    2. Variável: status
        a. Tipo: ERROR (variável definida na DLL)
        b. Dados: Passar a variável de retorno por referência, nesse caso, uma variável do tipo ERROR, que receberá o resultado da função após sua execução.
iii. Retorno da função:
    1. Tipo: ERROR (por referencia)
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

fff. Procedure RefAutorizaAbast(bico:string;var status:error);

i. Finalidade: Autorizar um determinado bico a realizar um abastecimento e, logo após, retornar ao seu estado padrão (bloqueado ou livre).
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser autorizado automaticamente.
    2. Variável: status
        a. Tipo: ERROR (variável definida na DLL)
        b. Dados: Passar a variável de retorno por referência, nesse caso, uma variável do tipo ERROR, que receberá o resultado da função após sua execução.
iii. Retorno da função:
    1. Tipo: ERROR (por referencia)
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

ggg. Procedure RefBloqueiaBico(bico:string;var status:error);

i. Finalidade: Bloquear um determinado bico para abastecimentos.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico a ser autorizado automaticamente.
    2. Variável: status
        a. Tipo: ERROR (variável definida na DLL)
        b. Dados: Passar a variável de retorno por referência, nesse caso, uma variável do tipo ERROR, que receberá o resultado da função após sua execução.
iii. Retorno da função:
    1. Tipo: ERROR (por referencia)
    2. Dados: ERROR é um tipo de dados definido pelo usuário, "None", significa que o comando foi executado corretamente; (veja a definição do tipo ERROR no final desse documento).

hhh. Function SetClock(par:string):boolean;

i. Finalidade: Ajustar o relógio interno do concentrador.
ii. Parâmetros de entrada:
    1. Variável: par
        a. Tipo: String
        b. Dados: Dia, hora e minuto, no formato DDHHMM, ou "AUTO", caso opte pelo ajuste automático, no qual a DLL utilizará o relógio do computador.
iii. Retorno da função:
    1. Tipo: Boolean
    2. Dados: "TRUE" caso comando bem sucedido, ou "FALSE", caso contrário.

iii. Function SetIntClock(dia,hora,minuto:byte):boolean;

i. Finalidade: Ajustar o relógio interno do concentrador.
ii. Parâmetros de entrada:
    1. Variável: dia
        a. Tipo: Byte (unsigned 8 bits);
        b. Dados: Valor inteiro, representando o dia atual;
    2. Variável: hora
        a. Tipo: Byte (unsigned 8 bits);
        b. Dados: Valor inteiro, representando a hora atual;
    3. Variável: minuto
        a. Tipo: Byte (unsigned 8 bits);
        b. Dados: Valor inteiro, representando o minuto atual;
iii. Retorno da função:
    1. Tipo: Boolean (1 Byte)
    2. Dados: "TRUE" comando ok, ou "FALSE", caso contrário.

jjj. Function SetParaBomba(bico:string):boolean;

i. Finalidade: Interromper o abastecimento atual, cortando o fluxo de combustível da mangueira. (função não suportada por todas as bombas)
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: Código do bico que deve ser interrompido.
iii. Retorno da função:
    1. Tipo: Boolean (1 Byte)
    2. Dados: "TRUE" comando ok, ou "FALSE", caso contrário.

kkk. Function SetPreset(st:string):boolean;

i. Finalidade: Pré determinar a quantidade de produto máxima que um bico poderá fornecer em um abastecimento.
ii. Parâmetros de entrada:
    1. Variável: st
        a. Tipo: String
        b. Dados: "BBVVVVVV", onde "BB" é o código do bico a ser pré-determinado, e "VVVVVV" é o valor máximo da venda. (para determinar uma venda de R$ 10,00, devemos informar o valor "001000").
iii. Retorno da função:
    1. Tipo: Boolean (1 Byte)
    2. Dados: "TRUE" comando ok, ou "FALSE", caso contrário.

lll. Function STRefAltPreco(par:PChar):integer;

i. Finalidade: Pré determinar a quantidade de produto máxima que um bico poderá fornecer em um abastecimento.
ii. Parâmetros de entrada:
    1. Variável: st
        a. Tipo: String
        b. Dados: "BBVVVVVV", onde "BB" é o código do bico a ser pré-determinado, e "VVVVVV" é o valor máximo da venda. (para determinar uma venda de R$ 10,00, devemos informar o valor "001000").
iii. Retorno da função:
    1. Tipo: Boolean (1 Byte)
    2. Dados: "TRUE" caso comando bem sucedido, ou "FALSE", caso contrário.

mmm. Function STVisualizacao(var visualizacao:PChar):PChar;

i. Finalidade: Ler o volume que os bicos estão dispensando no momento do pedido. Essa função responde tanto por valor, quanto por referencia, o valor lido da automação é o mesmo nas duas variáveis.
ii. Parâmetros de entrada:
    1. Variável: visualizacao
        a. Tipo: PChar (ponteiro para caractere).
        b. Dados: "BBVVVVVV", onde "BB" é o código do bico a ser pré-determinado, e "VVVVVV" é a quantidade que o bico está fornecendo no momento da consulta. (Veja comando de visualização, no protocolo de comunicação para mais detalhes).
iii. Retorno da função:
    1. Tipo: PChar. (por valor e referencia)
    2. Dados: O mesmo conteúdo da variável de referencia.

nnn. Function VBInicializaSerial(porta:string):boolean;

i. Finalidade: Abrir a porta de comunicação. Essa função é oferecida como opção, para linguagens que não possuem o tipo de dados "Byte", utilizado pela função InicializaSerial. Nesse caso, basta passar um caractere representando o número da serial a ser utilizada.
ii. Parâmetros de entrada:
    1. Variável: porta
        a. Tipo: String
        b. Dados: Caractere(s), representando a porta serial que será aberta para comunicação com o concentrador.
iii. Retorno da função:
    1. Tipo: Boolean
    2. Dados: "TRUE", caso sucesso na abertura de porta, ou "FALSE", caso contrário.

ooo. Function VBLeAbastecimento:abastVB;

i. Finalidade: Criada especialmente para uso com Visual Basic, essa função tem como objetivo ler o abastecimento atual em memória e passar o ponteiro de leitura de abastecimentos para o próximo.
ii. Parâmetros de entrada: Nenhum;
iii. Retorno da função:
    1. Tipo: abastVB
    2. Dados: Retorna os dados do abastecimento, como preço total, volume abastecido, número do bico, etc. em uma estrutura definida na DLL, compatível com Visual Basic. (veja a definição do tipo "abastVB" no final desse documento).

ppp. Procedure VBLePPL(var inf:string);

i. Finalidade: Ler o preço unitário praticado por determinado bico.
ii. Parâmetros de entrada:
    1. Variável: inf
        a. Tipo: String
        b. Dados: "BB", representando o bico que desejamos consultar.
iii. Retorno da função:
    1. Tipo: String (por referência)
    2. Dados: "PPPP", representando o preço unitário praticado no bico consultado.

qqq. Function VBLeVisualizacao:VBOnLine;

i. Finalidade: Ler o conteúdo que todos os bicos estão abastecendo no momento da consulta.
ii. Parâmetros de entrada: Nenhum
iii. Retorno da função:
    1. Tipo: VBOnLine. (veja a definição do tipo "VBOnLine" no final desse documento).
    2. Dados: Vetor de 48 posições, contendo o código do bico e o volume abastecido no momento da consulta.

rrr. procedure VBSetAutoLibera(var bico:string);

i. Finalidade: Definir um determinado bico para funcionamento em Auto-Liberação, ou seja, sempre que consultado, o concentrador permitirá que o abastecimento ocorra, liberando a bomba.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: "BB", representando o bico que desejamos definir como Auto-Liberado.
iii. Retorno da função:
    1. Tipo: String (por referencia)
    2. Dados: "BB", onde, após a execução da função, será "00", se ocorreu um problema no comando, ou, o mesmo número enviado, caso o comando tenha sido bem sucedido.

sss. procedure VBSetAutorizaAbast(var bico:string);

i. Finalidade: Autorizar um abastecimento em um bico anteriormente definido como bloqueado. Após esse abastecimento ter ocorrido, a bomba retornará para seu estado anterior.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: "BB", representando o bico que desejamos autorizar.
iii. Retorno da função:
    1. Tipo: String (por referencia)
    2. Dados: "BB", onde, após a execução da função, será "00", se ocorreu um problema no comando, ou, o mesmo número enviado, caso o comando tenha sido bem sucedido.

ttt. procedure VBSetBloqueiaBico(var bico:string);

i. Finalidade: Utilizada para definir um determinado bico como bloqueado. Nesse caso, para esse bico abastecer, o concentrador deverá permitir, enviando o comando de autorização.
ii. Parâmetros de entrada:
    1. Variável: bico
        a. Tipo: String
        b. Dados: "BB", representando o bico que desejamos bloquear.
iii. Retorno da função:
    1. Tipo: String (por referencia)
    2. Dados: "BB", onde, após a execução da função, será "00", se ocorreu um problema no comando, ou, o mesmo número enviado, caso o comando tenha sido bem sucedido.

uuu. Procedure VBSetPPL(var inf:string);

i. Finalidade: Alterar o preço unitário praticado pelo bico informado.
ii. Parâmetros de entrada:
    1. Variável: inf
        a. Tipo: String
        b. Dados: "BBPPPP", representando o bico que desejamos autorizar e o novo preço unitário, com três casas decimais.
iii. Retorno da função:
    1. Tipo: String (por referencia)
    2. Dados: (U"Bb"), onde: "Bb" é o número do bico, caso o comando tenha sido executado com sucesso, ou "?b", para código de bico inexistente ou, "?t", tempo de envio de comando esgotado.

vvv. Procedure Ver(var versao:info);

i. Finalidade: Coletar informações sobre a DLL.
ii. Parâmetros de entrada:
    1. Variável: versao
        a. Tipo: info (definida na DLL)
        b. Dados: título, versão, data da compilação e nome do autor.
iii. Retorno da função:
    1. Tipo: info (definida na DLL, por referência)
    2. Dados: Informações da DLL em uso.

4. Estruturas de dados

a. abast

i. value :boolean;
ii. total_dinheiro :currency;
iii. total_litros :double;
iv. PU :currency;
v. tempo :string[8];
vi. canal :string[2];
vii. data :string[10];
viii. hora :string[5];
ix. st_full :string[55];
x. registro :integer;
xi. encerrante :real;
xii. integridade :boolean;
xiii. checksum :boolean;

b. AbastVB

i. registro :integer;
ii. value :boolean;
iii. integridade :boolean;
iv. checksum :boolean;
v. encerrante :double;
vi. total_dinheiro :double;
vii. total_litros :double;
viii. PU :double;
ix. tempo :string;
x. canal :string;
xi. data :string;
xii. hora :string;
xiii. st_full :string;

c. AbastFid

i.	value	:boolean;
ii.	total_dinheiro	:currency;
iii.	total_litros	:double;
iv.	PU	:currency;
v.	tempo	:string[8];
vi.	canal	:string[2];
vii.	data	:string[10];
viii.	hora	:string[5];
ix.	st_full	:string[75];
x.	registro	:integer;
xi.	encerrante	:real;
xii.	integridade	:boolean;
xiii.	checksum	:boolean;
xiv.	tag	:string[16];

d. Abast2

i.	Value	:string[1];
ii.	total_dinheiro	:string[6];
iii.	total_litros	:string[6];
iv.	PU	:string[4];
v.	Tempo	:string[8];
vi.	Canal	:string[2];
vii.	Data	:string[10];
viii.	Hora	:string[5];
ix.	st_full	:string[55];
x.	registro	:string[4];
xi.	encerrante	:string[10];
xii.	integridade	:string[1];
xiii.	checksum	:string[1];

e. Abast3

i.	Value	:string[1];
ii.	total_dinheiro	:string[6];
iii.	total_litros	:string[6];
iv.	PU	:string[4];
v.	Tempo	:string[8];
vi.	Canal	:string[2];
vii.	Data	:string[10];
viii.	Hora	:string[5];
ix.	st_full	:string[75];
x.	registro	:string[4];
xi.	encerrante	:string[10];
xii.	id	:string[16];
xiii.	integridade	:string[1];
xiv.	checksum	:string[1];

f. IFid

i.	Value	:boolean;
ii.	Código	:string[8];
iii.	Endereço	:string[2];
iv.	Dia	:string[2];
v.	Hora	:string[2];
vi.	Minuto	:string[2];
vii.	Mes	:string[2];
viii.	Registro	:integer;
ix.	Status	:boolean;
x.	StFull	:string[37];

g. VBOnLine

i.	Bico	:array [1..48] of string;
ii.	Volume	:array [1..48] of double;

h. StFid

i.	Status	:string[32];

i. StStatus2

i.	Posição	:array [1..48] of string[10];

j. stPPL

i.	Bico	:string[2];
ii.	PPL	:string[4];

k. stEncerrante

i.	Bico	:string[2];
ii.	Encerrante	:string[8];

l. visualizacao

i.	stfull	:string[250];

m. StStatus

i.	Value	:string[100];

n. Retorno

i.	Value	:string[100];

o. Retorno2

i.	Value	:string[60];

p. info

i.	titulo	:string[20];
ii.	versão	:string[5];
iii.	data	:string[10];
iv.	autor	:string[20];

q. OnLine

i.	Litragem	:array [1..48] of real;
ii.	Bico	:array [1..48] of string[2];

r. Encerrante

i.	Bico	:string[2];
ii.	Valor	:real;

s. canal

i.	canal	: array [1..48] of byte;
ii.	PuAux	: array [1..48] of double;

t. Enc

i.	Bico	:string[2];
ii.	tipo	:string[1];
iii.	valor	:string[8];

u. MultiStatus

i.	Status	:array [1..48] of StOptions;

5. Variáveis definidas pelo usuário

a. Error

i. ErroString
ii. None
iii. ErroCodBico
iv. ErroCaracterModo
v. ErroTimeout
vi. ErroResposta

b. StOptions

i. Livre
ii. Pronta
iii. Falha
iv. Concluiu
v. Abastecendo
vi. Bloqueada
vii. SolicitaLib

6. Considerações finais

Esse manual foi desenvolvido com o intuito de descrever as funções da nossa biblioteca de integração com o seu software. Esperamos que esse processo seja fácil e ocupe o menor tempo necessário de desenvolvimento, assim, colocamo-nos à disposição para eventuais modificações ou criações de funções específicas, para que isso venha a agilizar ainda mais a integração.

Tratando-se de uma biblioteca genérica, nosso objetivo é atender as exigências das linguagens de programação atuais, baseado nesse conceito, procuramos fornecer funções compatíveis com todas elas. Caso sua linguagem possua alguma particularidade em termos de acesso à DLL, tipos de variáveis, etc., procurem-nos, afim de que possamos juntos solucionar o problema.

Para encerrar, gostaríamos de agradecer pela escolha de nossos equipamentos e soluções, e colocarmo-nos à disposição para solucionar suas dúvidas, ouvir suas reclamações e desenvolver, em conjunto com o seu software, a melhor solução de automação para postos de combustíveis do mercado.