Especificação Sk_access_tcp

Especificação Sk_access_tcp.dll V1.96
libsk_access_tcp.so V1.2

Descrição:

Sk_access_tcp.dll / libsk_access_tcp.so é uma biblioteca de funções desenvolvida pela Smak Teclados, com o objetivo de facilitar o trabalho daqueles que desenvolvem softwares para os produtos SMAK. Este documento destina-se a desenvolvedores e contém as informações técnicas necessárias ao uso da biblioteca

Rev. 1.8

Smak Tecnologia e Automação Ltda
Av. Sto. Antonio, 201 – Bela Vista - Osasco - SP Fone / Fax:11-3681-1809 /3699-4004/ 3699-4007
Email: comercial@smak.com.br - Site: www.smak.com.br

Especificação Sk_access_tcp

Índice

Histórico de alterações:

Revisão 1.8 (04-07-2018) :
- Acrescido nome e posição do arquivo de log
- Variáveis de ambiente grafadas em maiúsculo.
- Localização das variáveis de ambiente no SO Windows.

Revisão 1.7 (20-08-2015) :
- Revisão de textos.
- Acrescentado tópico importação da lib e Convenção de chamada.
- Integrado Manual_Estados.pdf
- Integrado Manual_Estados_Vb.pdf
- Introdução de sk_access_tcp.dll versão 1.95
- Introdução de libsk_access_tcp.so versão 1.2.....

Quem deve ler este manual:

Programadores que desejam se comunicar com um equipamento da Smak e procuram um driver/API/DLL. A sk_access_tcp.dll/libsk_access_tcp.so fornece funções que podem tornar mais rápido e fácil o desenvolvimento do programa aplicativo, pois já trata de várias questões de programação relativas ao relacionamento com o hardware do PC e do equipamento Smak em si. Neste manual estão as descrições de todas as funções disponibilizadas pela dll/lib.

Programadores que desejam se comunicar diretamente com o hardware devem consultar a Especificação do referido equipamento, no caso, por exemplo, do SKO-44 TCP, consultar **Especificacao_sko44_TCP.pdf**

Descrição SK_access_tcp:

Aqueles que desejam se comunicar diretamente com dispositivos de rede da Smak, devem consultar a documentação de hardware de cada equipamento, como por exemplo: Especificacao_sko44_TCP.pdf ou Especificação_MTC.pdf, onde constam o protocolo de comunicação do equipamento com seus comandos disponíveis.

Para simplificar e agilizar o desenvolvimento de uma aplicação a SMAK desenvolveu a lib *“sk_access_tcp.dll”/”libsk_access_tcp.so”* que gerencia os dispositivos de rede e disponibiliza uma API com funções que permitem flexibilidade no acesso aos equipamentos.

A lib cria um servidor (thread) para cada terminal que se conecta, permitindo a comunicação bidirecional com os mesmos, a lib opera em quatro modos de forma que o programador pode escolher o que mais se adapte às suas necessidades, entretanto no Modo 3 (Estados Multi thread), com poucas linhas pode-se desenvolver uma aplicação sendo este então, o modo o mais indicado.

Algumas linguagens não aceitam Multi thread (como algumas versões do VB), então pode-se usar o Modo 4 (Estados single thread).

Principio de funcionamento:

A lib (*sk_access_tcp.dll"/"libsk_access_tcp.so)* implementa um servidor Telnet para se comunicar com os terminais. Para ativar o servidor chamar **Start_Server** e para desativar chamar **Kill_Server**.

Após a chamada a **Start_Server**, os terminais conectados à rede começarão a se logar, cada terminal logado vai receber um número da lib(O primeiro terminal logado recebe o número 1). Algumas funções da lib usam este número para referenciar os terminais, na lib os terminais logados são chamados de clientes.

A lib disponibiliza várias funções e muitas delas vão interagir com o cliente em foco, de forma que a primeira coisa à se fazer para se comunicar com um cliente é colocá-lo em foco. Por exemplo:

```
Set_Client(1);      //coloca o cliente 1 em foco
Clr_Disp();         //Apaga o display do cliente 1
Disp("Teste")       //Coloca A String "Teste" no display do cliente 1
Set_Client(2);      //coloca o cliente 2 em foco
Clr_Disp();         //Apaga o display do cliente 2
Disp("Teste")       //Coloca A String "Teste" no display do cliente 2
```

No ambiente da lib, o IP do teclado nem sempre é importante, mas é possível saber o IP de um cliente:

```
char    ip[20];     //Lê IP do cliente em foco
Get_Client_IP(ip); //Lê string com ip no formato "aaa.bbb.ccc.ddd"
printf("ip\n");
```

Nos Modos 3 e 4 a lib coloca automaticamente o cliente correto em foco e não é necessário se preocupar com isso.

As vezes um cliente é especial e é necessário associá-lo sempre ao mesmo número de IP, por exemplo existe uma impressora conectada no cliente de IP 192.168.1.107.

Para que o terminal 192.168.1.107 sempre receba o número 3 use o código abaixo.

```
Lock_Ip(3,"192.168.1.107")  //Associa IP ao cliente 3.
```

Importação da lib:

Executando os procedimentos abaixo, todas as funções da lib estarão disponíveis para uso.

Do VB: Está disponibilizado o arquivo ***sk_access_tcp.vb6*** com o “Declare” de todas as funções, podendo ser usado para Ctrl C + Ctrl V e facilitar o trabalho de digitação.

Do Delphi/Lazarus: Para auxiliar a importação estática, está disponibilizado o arquivo ***sk_access_tcp.inc***.
Somente acrescentar a diretiva de compilação ***{$I sk_access_tcp.inc}***.

Do C++:

Windows:

Para auxiliar a importação implícita*, está disponibilizado o arquivo de header ***sk_access_tcp.h*** e o arquivo para linkagem implícita ***sk_access_tcp.lib***.

Acrescentar o arquivo “***sk_access_tcp.lib***” na lista de linkagem.
Acrescentar o Header:

#include "***sk_access_tcp.h***"

e chamar as funções:

Loadsk_access_tcp(); //Para carregar os nomes das funções

e

Freesk_access_tcp(); //Para liberar os recursos da dll.

Linux:

Para auxiliar a importação implícita, está disponibilizado o arquivo de header ***sk_access_tcp.h***.
Acrescentar a lib “***libsk_access_tcp.so***” na lista de linkagem.
Acrescentar o Header:

#include "***sk_access_tcp.h***"***.***

**No Windows a linkagem implícita (estática) com sk_access_tcp.lib aponta para uma dll Stub que faz o carregamento explícito da sk_access_tcp.dll, a manobra é necessária porque a sk_access_tcp.dll não está codificada em MSC++ e não gera o arquivo .lib necessário para a linkagem.*

Portas do teclado SKO-44 ETH:

O teclado **SKO-44 ETH** possui duas portas TCP (Teclado e AUX) e quando loga na lib, recebe dois números, um para o teclado e um para a porta AUX. O cliente “Teclado” é chamado pela lib de cliente Master e o cliente AUX como cliente Slave. Os clientes não logam sempre na mesma sequência, às vezes o Master pode ser 1 e o Slave ser 2 e às vezes ao contrário Slave = 1 e Master =2.

Para facilitar a operação, a lib reconhece os clientes Master + Slave como um par e fornece as funções Set_Mclient (Seleciona cliente Master) e Set_Sclient (Seleciona cliente Slave), Afinal eles tem o mesmo IP somente portas diferentes.

Se Master = Cliente 1 e Slave = Clente 2 chamando Set_MClient(1) ou Set_MClient(2) vai colocar em foco o Client 1 que é o Master e chamando Set_SClient(1) ou Set_SClient(2) vai colocar em foco o Client 2 que é o Slave.

Existem outras funções que tratam do par Master + Slave.

Ainda supondo Master = Cliente 1 e Slave = Cliente 2:

Get_Client_Type(1) devolve 1=Master e Get_Client_Type(2) devolve 0=Slave.

Get_Same_IP(1) devolve 2 e Get_Same_IP(2) devolve 1.

Get_MClient(1) ou Get_MClient(2) devolve 1 = Cliente 1 =Master.

Get_SClient(1) ou Get_SClient(2) devolve 2 = Cliente 2 = Slave.

Obs. A porta AUX pode ser configurada para Server, ver “Especificação_TCP_IP_SKO44.pdf”, neste caso o SKO-44 vai receber somente um número.

Protocolo do MTC:

O Micro terminal de consulta MTC, não aceita todos os comandos da lib.

Comandos da lib reconhecidos pelo MTC:

Clear_Disp
Clear_L1 - Apaga linha 1 e posiciona cursor no início da linha1
Clear_L2 - Apaga linha 2 e posiciona cursor no início da linha2

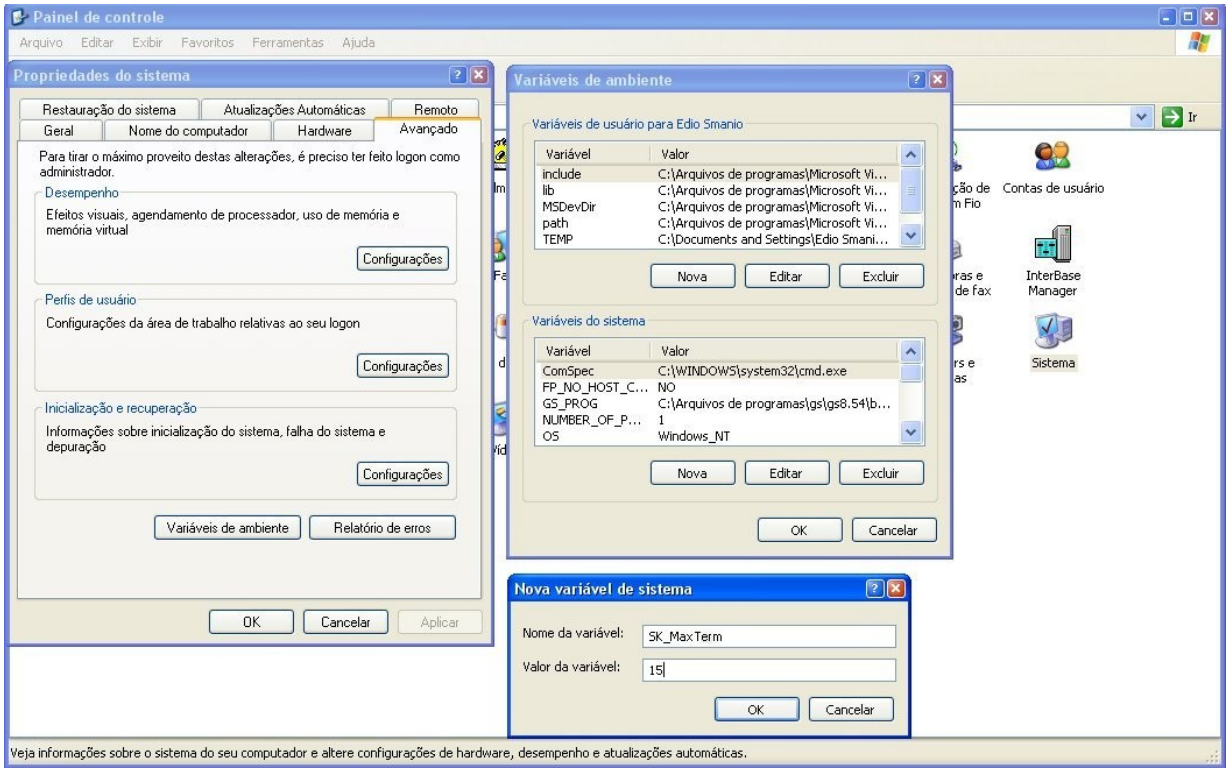
Quantidade de terminais:

Por padrão o servidor Telnet iniciado pela lib atende à somente 2 conexões.

A variável de ambiente **SK_ MAXTERM** redefine o número de terminais que vão ter suas conexões aceitas, para criar/editar esta variável, no windows, deve-se acessar:

Painel de controle -> Sistema -> Avançado -> Variáveis de ambiente -> Nova/Editar

E criar a variável **SK_ MAXTERM** com o número de conexões que se deseja aceitar.



No Linux para criar a variável (por exemplo com o valor 15), colocar o comando

export SK_MAXTERM=15

dentro do arquivo .profile do diretório $HOME

Variáveis de ambiente:

Além da variável de ambiente **SK_MAXTERM**, existem também as variáveis:

SK_LOG e **SK_TTATO**.

Se a variável não foi criada, seu valor é lido como 0 pela lib.

SK_LOG define o nível de registro de LOG e pode assumir valores de 0 a 7, quanto maior o valor mais detalhado é o LOG gerado.

O nível de registro de LOG assumido vai ser o maior valor entre o valor da variável **SK_LOG** e o valor solicitado pelo programa com a chamada à função Log_On.

Se **SK_LOG** é definido, o arquivo sk_access_tcp_log será gerado no mesmo sub-diretório do aplicativo que importou a lib.

SK_TATTO. Define o tempo em milisegundos que uma resposta vai ser esperada antes de sair por limite de tempo (Time-Out).

O valor padrão é de 500ms, mas dependendo do sistema pode ser necessário aumentar esse valor, dependendo do tráfego de rede valores de 1000ms ou ainda maiores podem ser usados.

Respostas são esperadas em diversas situações, muitas funções da lib esperam um retorno de status ou ACK, um valor de **SK_ TATTO** mal dimensionado pode impedir a comunicação com o teclado.

As variáveis de ambiente são armazenadas no registro do Windows em:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment

Demonstração e testes das funções da biblioteca Smak:

O utilitário de teste:

-Em Windows com fonte disponível em Delphi7

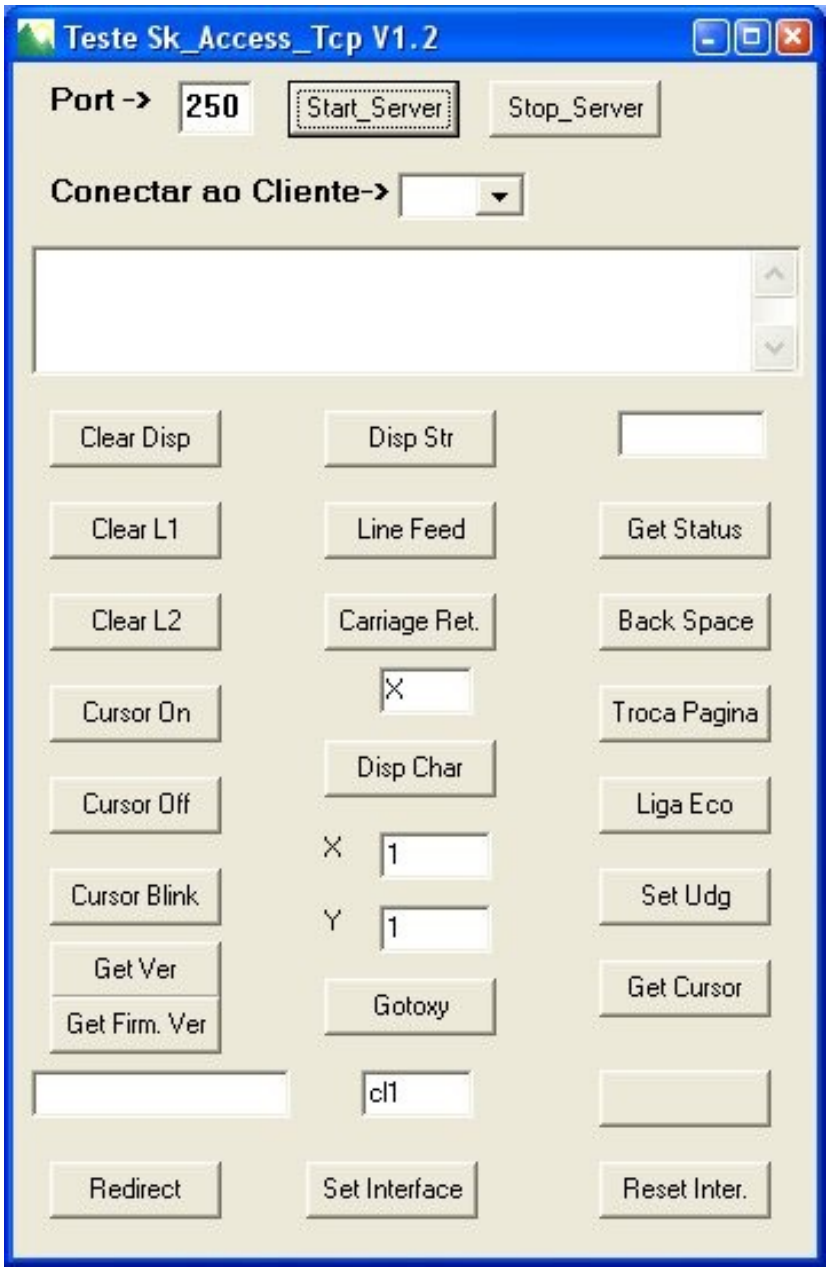
~\SKO\exemplos\sk_access_tcp\exemplo_Delphi\modo1\Use_Skaccess_Tcp \Use_Skaccess_Tcp.exe

-Em Linux

\Linux\so_driver\exemplos\sk_access_tcp\exemplo_Lazarus\modo1\TCP_Server

Foi desenvolvido usando o Modo 1 da lib especificamente como um exemplo para verificação da operação das funções exportadas pela lib. A figura a seguir ilustra sua aparência no Windows.

Cada botão tem o nome da função sendo testada. É importante lembrar que Este documento se refere à última versão da lib sendo que versões anteriores podem não suportar todas as funções descritas aqui.



Instalando a Lib:

Windows:

A instalação da biblioteca consiste em obter o pacote de software SKO_Tool_BoxVxxx.exe disponível no site www.smak.com.br, executá-lo e responder as perguntas de instalação.

Linux:

Instalação em Linux Kernel 2.6.xx

Copiar libsk_access_tcp.so para \usr\lib

Testando o ambiente:

Conecte o computador e um teclado SKO-44 Ethernet à mesma rede.

IMPORTANTE A faixa de IP do teclado e do computador tem que ser a mesma.

Como regra simplória isso quer dizer que só o último número do IP pode ser deferente, exemplo:

-Computador 192.168.1.10 acessa o teclado 192.168.1.20,

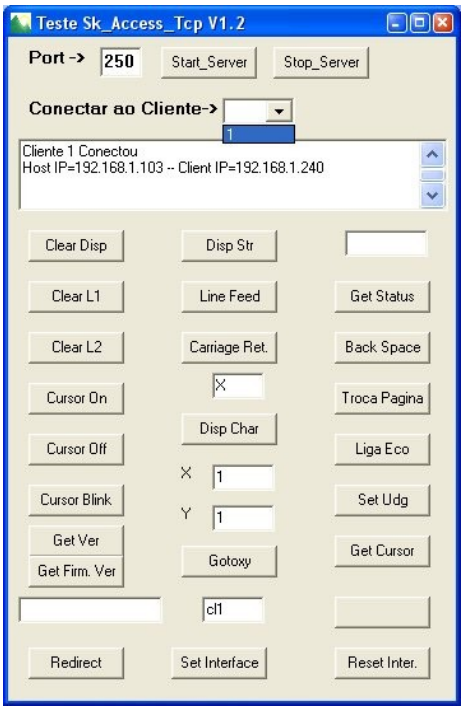
-Computador 10.0.0.10 não acessa o teclado 192.168.1.20,

É interessante que o conceito de Classe de IP, mascara de IP e número de IP seja entendido.

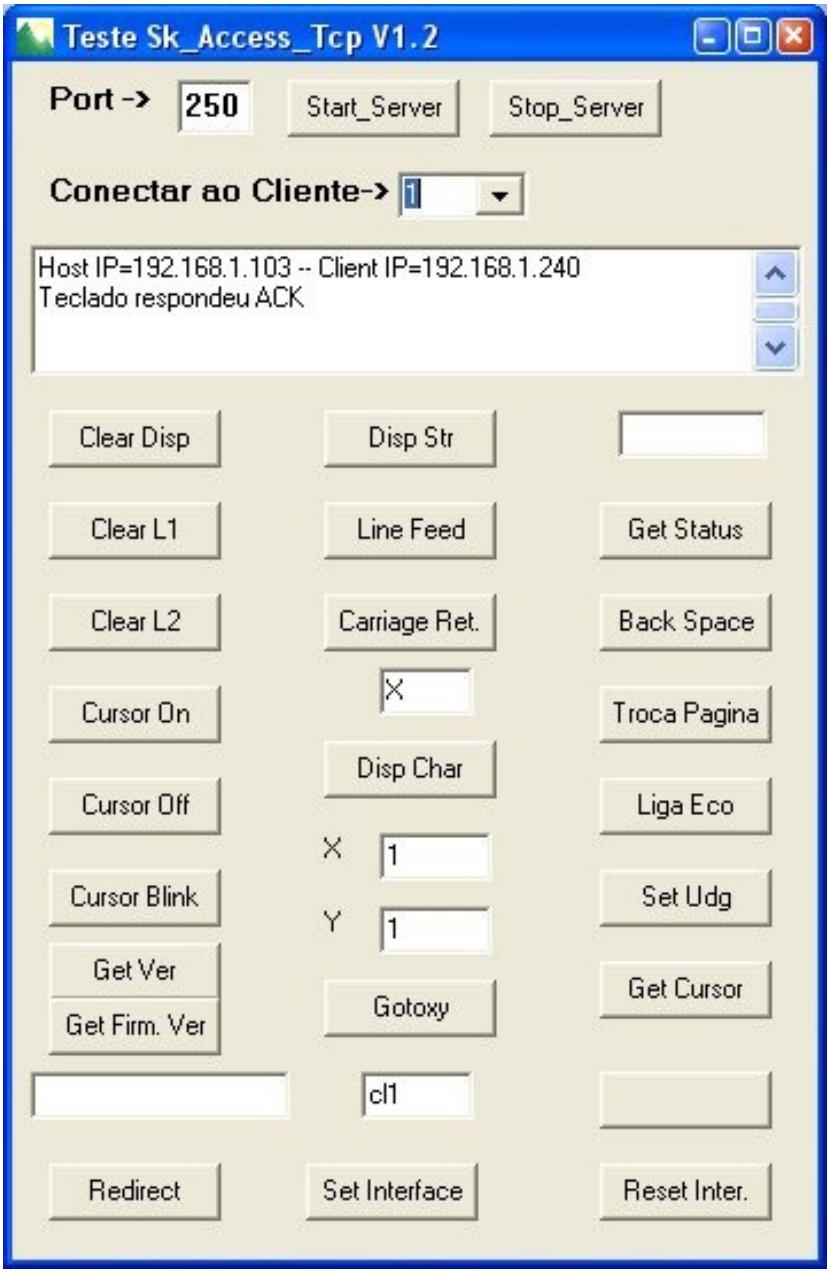
Com os teclados programados como cliente veja “Especificação_TCP_IP_SKO44.pdf”.

Execute o utilitário de testes.

Com o utilitário na tela click em “Start_Server”, isso deve iniciar o servidor Telnet de forma que os terminais poderão ter suas solicitações de conexão aceitas.



Após observar na janela de status que um cliente se conectou, selecionar o cliente para torná-lo o cliente ativo (colocá-lo em foco) de forma que dados serão recebidos e enviados exclusivamente para ele.



- Botão *“Disp Str”. A* primeira linha da janela de status será enviada para o terminal selecionado.

- Click os botões *“Clear_L1” e “Clear_L2” p*ara limpar as linhas do display.

Para saber sobre todas as funções exportadas pela lib, consulte a informação na tabela no final deste documento.

Utilizando a lib da SMAK em seu ambiente de desenvolvimento:

Uma vez que você conseguiu verificar o correto funcionamento do utilitário de testes, isto significa que *“sk_access_tcp.dll”/“libsk_access_tcp.so”* está corretamente disponível para uso no sistema.

A lib pode operar em 4 modos dependendo do conjunto de funções que forem utilizadas, é melhor estabelecer uma filosofia de trabalho e não misturar os modos.

O Modo 1 trabalha com Callbacks. Os eventos dos terminais geram mensagens para o aplicativo e o desenvolvedor tem que tratar estas mensagens.

O Modo 2 Pooling, como o nome já especifica depende do aplicativo consultar o status dos clientes e proceder de acordo.

O Modo mais simples para o desenvolvedor é o Modo 3 Estados Multi thread, onde a lib torna transparente o tratamento dos vários clientes, só é definido um conjunto de estados e a lib acompanha qual cliente está em qual estado, linguagens que não aceitam Multi thread(como VB6), podem usar o Modo 4 Estados Single thread.

A chamada à Start_Server define o modo de operação do servidor,

É interessante lembrar que em caso de dúvidas sobre o uso das funções da lib, o profissional desenvolvedor poderá consultar o código fonte dos exemplos disponibilizados.

Modos de operação da sk_access_tcp:

Modo 1: Mensagens
Ativado com Start_Server(Hwnd, WM_CONNECT, WM_COMMUNIC).
O aplicativo deve tratar as mensagens de conexão/desconexão e de comunicação.

Windows:
Exemplo para SKO44 em Delphi = use_Skaccess_Tcp.dpr
Exemplo para SKO44 em Delphi = cadastro.dpr
Exemplo para SKO44 em VB = use_Skaccess_Tcp.vbp
Exemplo para SKO44 em VB = comanda.vbp

Linux:
Exemplo para SKO44 em Lazarus = TCP_Server.lpi
Exemplo para SKO44 em C = foo.c

Modo 2: Pooling
Ativado com Start_Server(0,0,0)
A lib não gera mensagens, então a leitura e escrita dos dados é feita por pooling.
“List_Clients” É usado para obter a lista de clientes conectados.
“Get_Data” lê os dados dos clientes.

Modo 3: Estados Multi thread

Ativado com Start_Server(0,0,0), Register_State e Start_Machine

A dll gerência o sequenciamento dos estados definidos controle de terminais em foco e a separação de variáveis, sendo o mais simples de utilizar.

Windows:

Exemplo para SKO44 em Delphi = Comanda.dpr
Exemplo para MTC em Delphi = MTC_Estados.dpr
Exemplo para SKO44 em C = Comanda.dsw
Exemplo para SKO44 em VB = Comanda.vbp (problemas com multithread no VB6)

Linux:

Exemplo para SKO44 em Lazarus = Comanda.lpi
Exemplo para MTC em C = mtc.c

Modo 4: Estados Single thread

Ativado com Start_Server(Hwnd,0,WM_COMMUNIC), Register_State, Set_Single_Thread e Start_Machine

Windows:

Exemplo para sko44 em VB = Comanda.vbp

Linux:

Exemplo para MTC em C = mtcs.c

Mais informações sobre o Modo 1:

Ativado com Start_Server(Hwnd, WM_CONNECT, WM_COMMUNIC).

Quando o servidor é ativado desta maneira, será gerada uma mensagem de sistema do tipo WM_CONNECT toda vez que um terminal se conectar ou se desconectar do servidor da lib e será gerada uma mensagem do tipo WM_COMMUNIC toda vez que o servidor da lib receber uma comunicação de um cliente.

O programa aplicativo deve implementar as funções para tratar as mensagens de sistema WM_CONNECT e WM_COMMUNIC que serão enviadas pela lib.

Se for necessário, para entender melhor o conceito de mensagens de sistema, verificar no MSDN online: a função Send_message e windows messages, além dos exemplos já disponibilizados pela Smak.

As mensagens WM_CONNECT e WM_COMMUNIC recebem os dados nos parâmetros padrão do Windows Wparam e Lparam, sendo:

-Para WM_CONNECT: Wparam = IP do cliente
LparamHi = ID do cliente
LparamLo = 1 para conexão e 0 para desconexão

-Para WM_COMMUNIC: WparamLo = Scan code da tecla do Cliente
LparamHi = ID do cliente
LparamLo = ASCII code da tecla do cliente

Para colocar um cliente em foco, podemos utilizar as funções: Set_Interface (ou Set_Client como já colocado anteriormente).

Set_Interface recebe uma string do tipo “cl1” .. “cln” para selecionar cliente 1 até cliente n, retorna -1 se cliente não está conectado ou 1..n se estiver.

A string “cl0” coloca em foco cliente nenhum.

Talvez a maneira mais fácil de gerenciar vários terminais no Modo 1 seja criar um objeto que trata toda a operação de um terminal e criar uma thread nova para cada novo terminal que loga no sistema, assim cada thread trabalha de forma síncrona com seu terminal.

Veja um exemplo de estrutura em C de operação no Modo 1:

```
#define WM_CONNECT     WM_USER       //Define número das mensagens
#define WM_COMMUNIC  WM_USER +1

/*--------------------------------------------------------------------------------
  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
     Processes messages for the main window.
--------------------------------------------------------------------------------*/
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     ……
     switch (message)
     {
        case WM_CREATE:
             break;
         case WM_COMMAND:
             ……..
             break;
         case WM_PAINT:
             ……..
             break;
         case WM_CONNECT:                //Acrescentado tratamento para
             Deal_CONNECT();             //as mensagens
             break;
         case WM_COMMUNIC:
             Deal_COMMUNIC();
             break;
         case WM_DESTROY:
             PostQuitMessage(0);
             break;
         default:
             return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

```
/*-------------------------------------------------------------
        Ativa Servidor de teclados Smak
-------------------------------------------------------------*/
void _mode _Start_Server();
{
    Set_Port(250);                  //Define a porta do servidor
    Start_Server(self.handle        //Handle para o aplicativo
            WM_CONNECT              //Número da mensagem de conexão
            WM_COMMUNIC);           //Número da mensagem de comunicação
}

/*-------------------------------------------------------------
  Procedimento que recebe evento de conexão
  Wparam = IP do cliente
  LparamHi = ID do cliente
  LparamLo = 1 para conexão e 0 para desconexão
-------------------------------------------------------------*/
void _mode Deal_CONNECT(TMessage &message)
{
    if(message.LparamLo==1){
        //Trata conexão do cliente message.LparamHi
    }else{
        //Trata desconexão do cliente message.LparamHi
    }
}
/*-------------------------------------------------------------
  Procedimento que recebe evento de comunicação
  LparamHi = ID do cliente
  LparamLo = caractere recebido
-------------------------------------------------------------*/
void _mode Deal_COMMUNIC(TMessage &message)
{
    //Trata caractererecebido = message.LparamLo do cliente message.LparamHi
}
```

Mais informações sobre o Modo 3:

O programa de um servidor para uma rede tem um aumento de complexidade devido ao fato de que existem vários terminais (Clientes) acessando a este programa e cada um (terminal) pode estar em um momento diferente de execução deste programa.

Imagine um exemplo clássico de controle de comandas em uma loja de conveniência na hora da compra: lê-se o número da comanda, depois o código de cada um dos produtos ai o processo é finalizado com uma tecla, talvez a tecla 'Total'.

Se existem vários terminais, um pode estar esperando para ler o número da comanda, outro lendo os códigos dos produtos e um terceiro calculando o total.

Para melhor organizar este tipo de programação, usamos o conceito de estado, ou seja, definimos vários estados que podem ser assumidos pelos terminais e a lib gerencia o sequenciamento e o controle dos dados destes estados.

No exemplo da comanda teríamos os seguintes estados:

1-) Lê número da comanda
2-) Lê código dos produtos (ou tecla total)
3-) Mostra total
4-) Espera fim.

O que são estados:

Um estado na realidade é uma Sub rotina que vai ser executada até que uma condição seja encontrada, ou podemos dizer que o programa está em um estado até que um evento aconteça.

No caso da lib a rotina de tratamento do estado vai ser chamada a cada atividade do terminal (Envio de dados do terminal para o computador).

Então um estado, enquanto for mantido, poderá ser chamado várias vezes para atender ao terminal, devido a isso o programa de comanda ficaria na realidade da forma abaixo, pois Mostra a mensagem e lê a comanda não podem estar no mesmo estado, mostra mensagem vai executar uma vez só e lê comanda vai processar caractere a caractere até ler todo o número da comanda.

1-) Mostra mensagem 'Entre com número da comanda'
2-) Lê número da comanda
3-) Mostra mensagem para entrada de produtos
4-) Lê código dos produtos
5-) Mostra produto lido
6-) Espera finalização

Veja abaixo um exemplo simples em C:

```
void WINAPI Estado1(long W_param,long L_param)
{
    Clear_Disp();
    Disp("Digite um numero");
    Gotoxy(1,2);                    //Posiciona para numero
    Set_Eco_On();                   //Liga eco
    Set_Next_State(0);              //Muda para prox. estado imediatamente
}

void WINAPI Estado2(long W_param,long L_param)
{
    Get_State_String(L_param,-1,char(0x0d));   //Le string, Enter finaliza
}

void WINAPI Estado3(long W_param,long L_param)
{
char _string[40];
    if(State_String(NULL)=="")
        Set_State(2,0);                 //Se string vazia retorna para o estado 2
    Clear_Disp();
    strcpy(_string,"Voce digitou : ");
    strcat(_string, State_String(NULL));
    Disp(_string);
    Set_Next_State(0);                  //Muda para proximo estado imediatamente
}

void WINAPI State4(long W_param,long L_param)
{
    if(State_Ascii(L_param) == 'A') Set_State(1,0); //Espera teclar A e retorna
}                                                   //para o estado 1
```

```
void main()
{
    Set_Port(250);                          //Seleciona porta do servidor
    Start_Server(0,0,0);                    //Ativa servidor no Modo 3

    Register_State(1 , &Estado1);           //Registra estados
    Register_State(2 , &Estado2);
    Register_State(3 , &Estado3);
    Register_State(4 , &Estado4);

    Start_Machine();                        //Ativa maquina de estados
}
```

Depois que Start_Machine é chamado todos os clientes são colocados no estado 0 e vão mudar de estado individualmente conforme forem atendendo as condições de mudança de estado.

Programa exemplo Modo 3:

comanda.dpr (Delphi) / comanda.lpi (Lazarus)

Observe a descrição das funções em C ao final deste manual e os protótipos das procedures em Delphi/Lazarus em sk_access_tcp.inc.

No estado 1 ao final vê-se a instrução Set_Next_State(0), essa instrução faz com que o próximo estado a atender o terminal seja o estado 2

Atenção ao estado 2, onde só existe a Sub Get_State_String. Esse estado vai ser executado para cada vez que uma tecla for pressionada no terminal, Quando a tecla Enter for pressionada a Sub Get_State_String muda execução para o próximo estado (estado 3)

No estado 3 é finalizada a entrada da comanda e ativado o estado 4.

O estado 4 lê a tecla enviada pelo terminal State_Ascii(Lparam) e conforme o valor recebido, vai para a totalização(estado 6), processar o código de produto recebido(estado 5) ou montar a string que representa o código do produto(se mantém no estado 4).

O estado 5 mostra o produto referente ao código lido no estado 4 e volta para leitura do próximo código no estado 4, veja instrução Set_State 4,0

O estado 6, simplesmente espera ser pressionada a tecla 'A' no terminal para voltar ao estado 1 e reiniciar o processo.

Mais informações sobre o Modo 4:

No Modo 4 algumas tarefas feitas em segundo plano, tem que ser executadas pelo programa principal: No Windows são State_Proccess e State_Timer e no Linux ProccessMessages, verifique sua utilização nos programas exemplo.

Vamos discutir a operação em Modo 4 dentro do ambiente VB6 usando como referência o programa exemplo disponível no pacote SKO_Tool_Box em:

~\SKO\exemplos\sk_access_tcp\exemplo_VB\modo3\ comanda.vbp.

Observe a descrição das funções em C em manual_skaccess_tcp_dll.pdf
e os protótipos das SUBs em VB em sk_access_tcp.vb6

No fim do manual uma descrição das Subs usadas no programa exemplo.

No programa exemplo na Sub State1 ao final vê-se a instrução Set_Next_State 0, essa instrução faz com que o próximo estado a atender o terminal seja o State 2.

Não é o nome da Sub que define quem é o estado 1 e quem é o 2, é a função Register_State que faz isto:

```
Register_State 1, AddressOf State1     'Faz Sub State1 ser o estado 1
Register_State 2, AddressOf State2     'Faz Sub State2 ser o estado 2
```

Atenção a Sub State2, onde só existe a Sub Get_State_String. Esse estado vai ser executado para cada vez que uma tecla for pressionada no terminal, Quando a tecla <Enter> for pressionada a Sub Get_State_String muda execução para o próximo estado (Sub State3)

Na Sub State3 é finalizada a entrada da comanda e ativado o estado 4.

O estado 4 lê a tecla enviada pelo terminal (C=State_Ascii(Lparam)) e conforme o valor recebido, vai para a totalização(estado 6), processar o código de produto recebido(estado 5) ou montar a string que representa o código do produto(se mantém no estado 4).

O estado 5 mostra o produto referente ao código lido no estado 4 e volta para leitura do próximo código no estado 4, veja instrução Set_State 4,0.

O estado 6, simplesmente espera ser pressionada a tecla 'A' no terminal para voltar ao estado 1 e reiniciar o processo.

Além das rotinas de tratamento de estados, temos também algumas rotinas de apoio e cálculo, dentre elas algumas rotinas merecem explicação: **SubWndProc , SubclassWindow e UnsubclassWindow,** são chamadas em Sub Form_Load na Sub Form_Unload, elas podem ser tratadas neste caso como “receita de bolo” e não precisam ser necessariamente entendidas, entretanto, segue sua explicação.

Estas 3 rotinas servem para podermos tratar a mensagem enviada pela lib e auxiliar na implementação do controle de estados.

Verifique que SubWndProc chama a função State_Proccess da lib, perceba ainda que também é necessário criar um Timer de 100ms para chamar a função State_Timer da lib.

Para uma explicação mais detalhada ver abaixo “O que são eventos”.

O que são eventos:

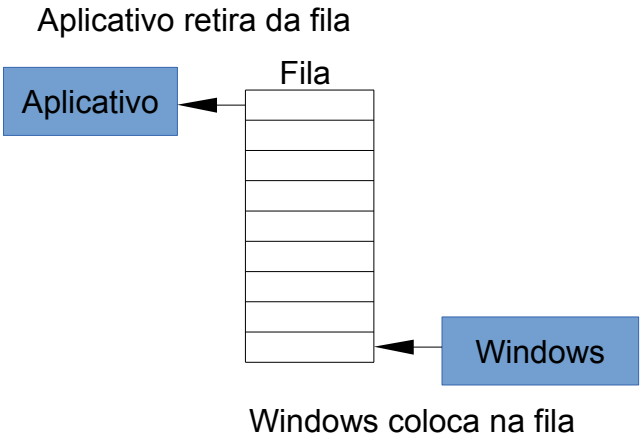
Como programador VB, você conhece o que é um evento:

Form load
Form unload
Comand click
Comand keypress
etc

Esta forma de tratamento não é uma forma definida pelo VB, mas a forma definida pela Microsoft como padrão do Windows.

Apesar do VB ser a primeira linguagem a facilitar a implementação deste esquema, todo aplicativo gráfico em Windows funciona desta forma, ou seja, recebendo mensagens do sistema(eventos).

Estes eventos são colocados em uma fila (queue) do programa aplicativo conforme o Windows julgue pertinente. Por exemplo: eventos de teclado só são colocados na fila do programa que está em foco.



Um programa Windows consequentemente tem como atividade criar, ler e processar as mensagens lidas do sistema.

O sistema tem definidas as mensagens padrões: KeyPress, Click, GotFocus, etc.

Uma função normalmente chamada WndProc recebe estas mensagens e chama a sub rotina apropriada para seu tratamento.

Em VB WndProc não é aparente fica “escondida”, entretanto é possível interferir com ela de forma a acrescentar tratamento para mensagens(eventos) que são criados por nós e não são padrões do sistema.

Acrescentando uma nova mensagem:

Para a utilização da sk_access_tcp.dll em VB é necessário que o programa possas atender a uma mensagem enviada pela lib.

Para que se possa atender a essa mensagem é necessário fazer um override(substituir) a rotina WndProc. No VB este processo de substituição se chama sub-classing.

Temos então uma nova “Function” WndProc e duas “Subs” para fazer e desfazer a substituição. Ver no programa exemplo Comanda.vbp Form_Load, Form_Unload, SubWndProc, SubClassWindow e UnSubClassWindow.

Estrutura de um programa Windows:

Um programa gráfico Windows é constituído basicamente de 4 seções:

Inicialização, leitura de mensagens, despacho das mensagens e tratamento das mensagens:

Seção de inicialização:

Simplificando a seção de inicialização é formada principalmente por:

Init_Queue
Flush_Queue
Init_App

Nesta seção é criada(registrada) a fila(Queue) de mensagens para a aplicação, ou seja, o Windows toma conhecimento da aplicação e passa a colocar mensagens em sua fila de eventos.

Leitura de mensagens:

Esta rotina recebe o nome de “Message Loop” e tem por finalidade ler a fila de mensagens e despachar as mensagens de volta para Wnd Proc.

“Message Loop”

```
While Get_Message <> 0      ' Se mensagem = 0, finaliza o programa.
    Translate_Message
    Dispatch_Message
End While
```

No código acima lê-se: enquanto mensagem <> 0, Dispatch Message

Disptach_Message (Microsoft vudu) chama o Windows que chama de volta seu programa na rotina WndProc(mecanismo conhecido como Call-Back).

A seção de inicialização e de leitura de mensagens fica dentro de uma rotina normalmente chamada de WinAPP.

Despacho das mensagens:

A rotina WndProc desvia cada uma das mensagens para a sua respectiva rotina de tratamento.

```
Select case
    case WM_KeyPress
        Call Form_KeyPress
    case WM_Load
        Call Form_Load
    .
    .
    .
End Select
```

Tratamento de mensagens:

Esta parte do trabalho é mais conhecida, são justamente as rotinas que tratam cada uma das mensagens e são implementadas nos módulos e formulários do VB.

Configuração do teclado SKO44:

Como visto em “Especificacao_sko44_TCP.pdf”. O teclado pode ser configurado via protocolo http. Entretanto é possível também programá-lo via telnet usando funções disponibilizadas pela DLL.

void _mode Config_Client(char *config);

Esta função envia para o terminal em foco uma nova configuração, essa configuração está em um formato interno da dll. E deve ser obtida se usando a função Build_Config.

char _mode *Build_Config(char *config , char *buffer);

Build_Config constrói a string de configuração a partir de uma configuração padrão tipo DOS.

O formato de entrada da função é Port , Baud , Bits , Paridade , Stops

Port : para o teclado SKO 44 ETH só pode ser o valor “Port0”

Baud: (110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400).

Bits: Tamanho do caractere (5, 6, 7, 8) bits.

Paridade: Modo do bit de paridade do caracter pode ser: n, o, e, m

- n = none, paridade sempre 0.
- m = mark, paridade sempre 1.
- o = odd, paridade inpar.
- e = even, paridade par.

Stops : Número de Stop bits, pode ser 1 ou 2.

Exemplo: Configura porta auxiliar do teclado para 9600 bauds, 8 bits , sem paridade, 1 stop bit

```
char buffer[50];
Set_Client(1);                                          //Coloca cliente em Foco.
Config_Client(Buid_config(“Port0,9600,8,n,1”, buffer)); //Configura cliente.
```

Veja a seguir a tabela com as funções e procedimentos disponíveis na DLL Smak.

Convenção de chamada:

Nas funções abaixo, substituir _mode pela convenção de chamada (Calling Convention) apropriada.

A dll / lib segue a convenção padrão do sistema operacional sendo:

_mode = _stdcall (ou WINAPI) em Windows.

_mode = _cdecl em Linux.

```
Typedef unsigned char byte          /* 8 bits */
Typedef unsigned long int Dword     /* 32 bits */
```

Tabela de funções e procedimentos disponibilizadas por sk_access_tcp.dll:

Controle do servidor			
Index	**PROTÓTIPO**	**WIN**	**LINUX**
1	void _mode Start_Server(HWND hWnd , Dword WM_CONNECT , Dword WM_COMMUNIC); -Esse procedimento ativa o Servidor telnet e o envio de mensagens. hWnd : HWND = Handler do programa do usuário que vai receber as mensagens WM_CONNECT = Número da mensagem de conexão/desconexão WM_COMMUNIC = Número da mensagen de comunicação Se hWnd = NULL todas as mensagens são desativadas. Se WM_CONNECT ou WM_COMMUNIC = NULL a respectiva mensagem é desativada Parâmetros das mensagens enviadas: WM_CONNECT -> Wparam = IP do Cliente Lparam HI = ID do cliente Lparam LO = 1 se cliente conectou e 0 se desconectou WM_COMMUNIC -> Lparam HI = Codigo da tecla precionada no teclado Lparam LO = ID do cliente	X	X

Index	PROTÓTIPO	DESCRIÇÃO	WIN	LINUX
2	void _mode Kill_Server();	Desativa e remove o servidor Telnet.	X	X
3	void _mode Set_Port(integer port);	Define número da porta do servidor TELNET	X	X

IP				
Index	**PROTÓTIPO**	**DESCRIÇÃO**	**WIN**	**LINUX**
1	void _mode Get_Host_IP(char *ip);	Devolve em ip uma string com IP do servidor.	X	X
2	void _mode Get_Client_IP(char *ip);	Devolve em ip uma string com IP do cliente em foco.	X	X
3	Dword _mode IP_ascii2HEX(char *ip_ascii);	Converte endereço IP em ASCII para HEX.	X	X
4	char *IP_HEX2ascii (char *buffer,Dword ip);	Converte endereço IP em HEX para ASCII.	X	X
5	Dword _mode Inv_IP_HEX(Dword ip);	Inverte endereço IP HEXA MSB = LSB	X	X

Funções de tratamento do display

Index	PROTÓTIPO	DESCRIÇÃO	WIN	LINUX
1	bool _mode Send_Disp_Ctrl(byte data);	Envia comando direto para o display	X	X
2	bool _mode Clear_Disp();	Apagar o display	X	X
3	bool _mode Clear_L1();	Apagar a linha 1 do display	X	X
4	bool _mode Clear_L2();	Apagar a linha 2 do display	X	X
5	bool _mode Cursor_Off();	Desligar o cursor no display	X	X
6	bool _mode Cursor_On();	Ligar o cursor	X	X
7	bool _mode Cursor_Blink();	Piscar o cursor	X	X
8	bool _mode Back_Space();	Back space do cursor	X	X
9	bool _mode Line_Feed();	Line-feed no display	X	X
10	bool _mode Carriage_Return();	Carriage return	X	X
11	bool _mode Gotoxy(byte x,byte y);	Posiciona cursor na Coluna X, linha Y. (sendo: y: 1ou 2 e x:de 1 a 40)	X	X
12	bool _mode Disp_Char(char data);	Exibe um caracter no display	X	X
13	bool _mode Disp(char *data);	Exibe uma string de até 80 caracteres. Strings podem gerar um New_Line com scroll no display e isso gera um excesso de processamento que pode ocasionar um “overrun” e perda de caracteres. Deve-se ter controle destas strings.	X	X
14	byte _mode Get_Cursor(char *x_pos,char *y_pos);	Lê em x_pos e y_pos as coordenadas atuais do cursor (home = 1,1), Adicionalmente retorna a posição absoluta do cursor (0 a 79). Caso o teclado não responda retorna 255.	X	X
15	void _mode Set_Udg(byte caracter, byte def0 , byte def1 , byte def2 , byte def3 , byte def4 , byte def5 , byte def6 , byte def7);	UDG (User Defined Graphics) Permite o desenho de até oito caracteres especiais que ficam armazenados nas posições 0 a 7 da tabela interna do display. Na procedure, o parâmetro *“caracter”* representa o código a ser atribuído ao desenho. *“def 0”* a *“def7”* definem os elementos ativos da matriz 5x7 do caracter no display. Veja abaixo um exemplo dos valores de “caracter” e de *“def0 a 7”* para formar a letra 'R' na posição 01h da tabela interna caracter=01h def1 = x x x 1 1 1 1 0 = 1Eh def2 = x x x 1 0 0 0 1 = 11h def3 = x x x 1 0 0 0 1 = 11h def4 = x x x 1 1 1 1 0 = 1Eh def5 = x x x 1 0 1 0 0 = 14h def6 = x x x 1 0 0 1 0 = 12h	X	X

Comunicação				
Index	**PROTÓTIPO**	**DESCRIÇÃO**	**WIN**	**LINUX**
1	Bool _mode Send_Data(byte data);	Envia um byte para o terminal	X	X

2	int _mode Get_Data(PTkey_def buffer , int size);
	
	typedef struct key_def {
	char scan;
	char ascii;
	}Tkey_def;
	typedef Tkey_def *Ptkey_def;
		Transfere para "buffer" um máximo de "size" caracteres lidos da porta serial. Retorna o número de caracteres efetivamente lidos.	X	X

3	char _mode Ascii(byte Scan_code);	Obtem código ascii a partir do scan code	X	X
4	byte _mode Get_Status();	Lê status da última operação, Retorna: 00h se não houve erros; 01h se houve erro e FFh se saiu por time-out.	X	X
5	void _mode Redirect();	Captura dados do terminal e redireciona para o buffer do teclado.	X	
6	void _mode Set_Eco_On();	Liga eco	X	X
7	void _mode Set_Eco_Off();	Desliga eco	X	X
8	void _mode Flush();	Esvazia buffer de dados	X	X
9	Bool _mode Alive(Word client);	Verifica se cliente esta conectado	X	X
Controle				
1	void _mode Keyb_Reset();	Reseta o teclado.	X	X
2	void _mode Get_Firmware_Version(char *ver);	Retorna string no formato (TCP)=v.vvv	X	X
3	bool _mode Set_Page1();	Ativa pagina 1 de scancode.	X	X
4	bool _mode Set_Page0();	Ativa pagina 0 de scancode.	X	X
5	void _mode Set_Caps();	Ativa Caps-Lock do client	X	X
6	void _mode Clr_Caps();	Desativa Caps-Lock do cliente	X	X
7	void _mode Set_Num();	Ativa Num_Lock do cliente	X	X
8	void _mode Clr_Num();	Desativa Num-Lock do cliente	X	X
9	void _mode Set_Scrl();	Ativa Scroll-Lock do client	X	X
10	void _mode Clr_Scrl();	Desativa Scroll-Lock do cliente	X	X
11	bool _mode Get_Caps();	Le satus do Caps-Lock do cliente	X	X
12	bool _mode Get_Num();	Le satus do Num-Lock do cliente	X	X
13	bool _mode Get_Scrl();	Le satus do Scroll-Lock do cliente	X	X
14	void _mode Config_Client(char *config);	Configura Cliente, usar Build_Config para gerar string de configuração	X	
15	char _mode *Build_Config(char *config , char *buffer);	Monta String de configuração, Formato de entrada Port,baud,bits,paridade,stops. Ex. "Port0,9600,8,n,1"	X	
16	void _mode ProccessMessages();	Executa procedimentos pendentes		X

Seleção de Terminais				
Index	**PROTÓTIPO**	**DESCRIÇÃO**	**WIN**	**LINUX**

1	void _mode List_Clients(PTsockets lista);
	
	lista[1] = IP do terminal 1;
	lista[2] = IP do terminal 2;
	.
	.
	Se IP=0000, não há cliente conectado.
	
	Typedef struct sockets {
	unsigned int ip[255];
	} Tsockets;
	Typedef Tsockets *Ptsockets;
		Transfere para "lista" os dados dos terminais conectados.	X	X

2	int _mode Set_Client(Word client);	Seleciona cliente pelo número, Retorna -1 se falha ou nro do cliente se OK.	X	X
3	int _mode Set_Mclient(Word client);	Seleciona Cliente Master	X	X
4	int _mode Set_Sclient(Word client);	Seleciona Cliente Slave	X	X
5	int _mode Get_Mclient(Word client);	Retorna Cliente Master	X	X
6	int _mode Get_Sclient(Word client);	Retorna Cliente Slave	X	X
7	int _mode Get_Client_Type(Word client);	Retorna 0=Slave 1=Master	X	X
8	int _mode Get_Same_Ip(Word client);	Devolve outro cliente com o mesmo IP	X	X
9	bool _mode Lock_Ip(Word client , char *ip);	Associa IP ao ID do cliente	X	X
10	bool _mode Unlock_Ip(Word client);	Desassocia IP ao ID do cliente	X	X
11	void _mode Unlock_All();	Desassocia TODOS Ips	X	X
12	int _mode Set_Interface(char *interf);	Recebe string "cl1" a "cln" para selecionar cliente 1..n. Retorna 1..n se cliente conectado e -1 se não conectado	X	X
13	void _mode Reset_Interface();	Desliga eco e redirecionamento de todos os terminais, tira foco dos terminais	X	X

DLL				
Index	**PROTÓTIPO**	**DESCRIÇÃO**	**WIN**	**LINUX**
1	void _mode Get_Dll_Version(char *ver);	Retorna string com versão da DLL , max 20 caracteres.	X	X

2	int _mode GetOperatingSystem();	Identifica sistema operacional retorna um valor numérico de -1 a 6 sendo:
		cOsUnknown = -1;
		cOsWin95 = 0;
		cOsWin98 = 1;
		cOsWin98SE = 2;
		cOsWinME = 3;
		cOsWinNT = 4;
		cOsWin2000 = 5;
		CosXP = 6;
		cOsXPPro64 = 7;
		cOsServer2003 = 8;
		cOsHomeServer = 9;
		cOsServer2003R2 = 10;
		cOsWinVista = 11;
		cOsServer2008 = 12;
		cOsServer2008R2 = 13;
		COsWin7 = 14;
			X	

3	int _mode Select_Interface();	Retorna:-1, Somente para compatibilidade com sk_access.dll	X	
4	void _mode Free_Sk_Access();	Compatibilidade com sk_access.dll	X	X
5	void _mode Run_Silent();	Não envia mensagens de aviso aos clientes	X	X
6	void _mode Log_On(int dbg_level);	Registra atividade da DLL no arquivo 'sk_access_tcp_log'	X	X
7	void _mode Log_Off();	Desativa log	X	X

Estados				
Index	**PROTÓTIPO**	**DESCRIÇÃO**	**WIN**	**LINUX**
1	void _mode Register_State(int State, Tproc proc);	Registra procedimento que vai atender ao estado	X	X
2	void _mode Set_Next_State(int delay);	Ativa o próximo estado após o delay (módulo de 100ms, 10 = 1seg)	X	X
3	void _mode Set_State(int State , int delay);	Ativa estado especificado, após o delay (módulo 100ms, 10 = 1seg)	X	X
4	char _mode *State_String(char *buffer);	Retorna State_String	X	X
5	void _mode Get_State_String(unsigned int L_param , int size, char delimiter);	Lê string do terminal até encontrar uma condição de finalização (tamanho ou delimitador). Após encontrar a condição, ativa o próximo estado. 0(zero) desativa as condições e as duas podem ser usadas simultaneamente	X	X
6	int _mode Mount_State_String(unsigned int L_param);	Monta string com as teclas recebidas	X	X
7	char _mode State_Ascii(unsigned int L_param);	Retorna valor ascii da tecla em Lparam	X	X
8	char _mode State_Scan(unsigned int L_param);	Retorna código de scan da tecla em Lparam	X	X
9	bool _mode Save_String(char *str, int mem);	Salva string na posição de memória especificada	X	X
10	char _mode *Recall_String(char *str, int mem);	Retorna string salva na posição de memória	X	X
11	Bool _mode Save(double nro, int mem);	Salva double na posição de memória especificada	X	X
12	double _mode Recall(int mem);	Retorna double salva na posição de memória	X	X
13	void _mode State_Proccess(unsigned int L_param);	Processa máquina de estados	X	X
14	void _mode State_Timer();	Processa timer da máquina de estados	X	X
15	Void _mode Start_Machine();	Dispara máquina de estados, todos os terminais vão para o estado 1	X	X
16	Void _mode Stop_Machine();	Paralisa máquina de estados	X	X
17	void _mode Set_Single_Thread()	Ativa modo single thread na máquina de estados	X	X

Nota sobre a função Send_Disp_Ctrl:

O controle do display do teclado pode ser acessado diretamente através do comando "Send_Disp_Ctrl", a tabela abaixo mostra como devem ser composto um comando.

INSTRUÇÃO	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIÇÃO
Clear Disp	0	0	0	0	0	0	0	1	Apaga Display e coloca cursor em (1,1)
Home	0	0	0	0	0	0	1	*	Posiciona cursor em (1,1)
									Retorna mensagem deslocada
Mode	0	0	0	0	0	1	I/D	S	I/D = sentido de deslocamento do cursor
									S = Desloca ou não a mensagem
Control	0	0	0	0	1	D	C	B	D = Ativa/desativa display
									C = Ativa/desativa cursor
									B = Ativa/desativa intermitência do cursor
Shift	0	0	0	1	S/C	R/L	*	*	S/C = Movimento do cursor
									R/L = Sentido de deslocamento da mensagem
INTERFACE	0	0	1	DL	N	F	*	*	DL = Número de bits da interface
									N = Número de linhas do display
									F = Tipo de formatação do caracter
CG Ram	0	1	Endereço						Aponta endereço corrente da memoria gráfica
DD Ram	1	Endereço							Aponta endereço corrente da memoria de dados

Ex.: Para apagar o display enviar 0x01

Para colocar o cursor em home enviar 0x02

Aqui estão apresentados todos os comandos aceitos pelo display, alguns destes comandos podem desconfigurar o display tornando-se necessário um reboot (liga/desliga) do teclado.


Upper 4bit
Lower 4bit
LLLL
LLLH
LLHL
LLHH
LHLL
LHLH
LHHL
LHHH
HLLL
HLLH
HLHL
HLHH
HHLL
HHLH
HHHL
HHHH
CG RAM (1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)