

DLL PMTG V1.5



Manual DLL PMTG

Rev. 1.2

Maio 2012

ÍNDICE

Inicialização e Finalização	1
mt_startserver	1
mt_finishserver	1
mt_version	1
mt_connectlist	1
Funções de tratamento de IP.....	2
mt_gethostip.....	2
char * __stdcall mt_gethostip(char *oip);.....	2
mt_inet_ntoa	2
mt_inet_ntoa_inv	2
mt_inet_addr	2
mt_inet_addr_inv	2
mt_ipfromid	2
Comandos básicos para o terminal.....	3
mt_sendlive.....	3
mt_restart	3
mt_ftpmode	3
Rede	3
mt_sendconfig	3
mt_reqconfig.....	3
mt_getconfig.....	4
mt_sendexconfig.....	4
mt_reqconfig.....	4
mt_getexconfig	4
Display.....	4
mt_backspace	4
mt_carret	5
mt_linefeed.....	5
mt_formfeed.....	5
mt_gotoxy	5
mt_gotoxyref	5
mt_dispstr	5
mt_dispch	5
mt_dispclrln	6
mt_seteditstring.....	6

mt_reqeditstring.....	6
mt_geteditstring.....	6
Teclado.....	6
mt_setenablekey.....	6
mt_getenablekey.....	7
mt_reset.....	7
mt_setcapslock.....	7
mt_getcapslock.....	7
mt_setnumlock.....	7
mt_getnumlock.....	7
mt_programkbd.....	8
mt_setbeep.....	8
mt_setbeepkey.....	8
mt_getkey.....	8
Serial.....	8
mt_setenableserial.....	8
mt_getenableserial.....	9
mt_sendbinserial.....	9
mt_getserial.....	9
mt_sendconfigserial.....	9
mt_reqconfigserial.....	10
mt_getconfigserial.....	10
mt_settermserial.....	10
Cartão Magnético.....	11
mt_sendsetcard.....	11
mt_reqgetcard.....	11
mt_getcardbuf.....	11
Impressora.....	11
mt_sendinitprn.....	11
mt_reqgetstatusprn.....	11
mt_sendbinprn.....	12
A troca de mensagens do programa principal com a DLL.....	12
CONTATOS GERTEC.....	13

Este manual descreve as funções da DLL PMTG e os retornos esperados para as mesmas.

Inicialização e Finalização

mt_startserver

```
int __stdcall mt_startserver(HWND mywhnd, int conecmsg, int commmsg);
```

Esta é a primeira função que deve ser chamada. Se tiver sucesso na sua chamada, os terminais já conectarão

ao servidor.

mywhnd: Handle para a janela principal do programa do servidor, que é para onde a DLL irá mandar as mensagens para troca de dados. Se não quiser receber as mensagens deve seu valor deve ser NULL.

conecmsg: Valor da mensagem que a DLL enviará quando um terminal conectar/desconectar.

commmsg: Valor da mensagem que a DLL enviará quando terminal enviar dados.

retorna: 1 se servidor inicializado com sucesso, 0 se houve algum erro.

mt_finishserver

```
void __stdcall mt_finishserver(void);
```

Após chamar esta função, a DLL libera a memória armazenada, desconecta todos os terminais e para de aceitar novas conexões.

mt_version

```
char __stdcall mt_version(void);
```

Retorna versão da DLL. Ex.: 0x14 corresponde a versão 1.4.

mt_connectlist

```
TTABSOCK __stdcall mt_connectlist(void);
```

```
typedef struct
```

```
{
```

```
    DWORD ip[255];
```

```
} TTABSOCK;
```

```
TTABSOCK.ip[0]=0x00000000
```

```
TTABSOCK.tab[1]=0xB600A8C0
```

```
TTABSOCK.tab[2]=0x00000000
```

```
...
```

Indica que no ID "1", existe um terminal conectado com o IP "B600A8C0" (192.168.0.182).

Funções de tratamento de IP

mt_gethostip

char * __stdcall mt_gethostip(char *oip);

Retorna o IP da máquina local em ASCII formatada por pontos.

oip: array de bytes onde será escrito os dados.

mt_inet_ntoa

char * __stdcall mt_inet_ntoa(DWORD oip);

Retorna IP em ASCII formatado por pontos.

oip: IP no formato de rede (DWORD).

mt_inet_ntoa_inv

char * __stdcall mt_inet_ntoa_inv(DWORD oip);

Retorna IP em ASCII formatado por pontos, invertendo byte mais significativo pelo menos significativo.

oip: IP no formato de rede (DWORD).

mt_inet_addr

DWORD __stdcall mt_inet_addr(char *oip);

Retorna IP no formato de rede (DWORD).

oip: array de bytes em ASCII formatado por pontos.

mt_inet_addr_inv

DWORD __stdcall mt_inet_addr_inv(char *oip);

Retorna IP no formato de rede (DWORD) , invertendo byte mais significativo pelo menos significativo.

oip: array de bytes em ASCII formatado por pontos.

mt_ipfromid

char * __stdcall mt_ipfromid(int ID);

Retorna IP formatado por pontos de um terminal.

ID: ID do terminal.

Comandos básicos para o terminal

mt_sendlive

```
int __stdcall mt_sendlive(int ID);
```

Envia o comando de vivo para o terminal (IDvLive).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_restart

```
int __stdcall mt_restart(int ID);
```

Faz com que o terminal se reinicialize (IDRestart).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_ftpmode

```
int __stdcall mt_ftpmode(int ID);
```

Faz com que o terminal entre em modo FTP (IDvFTPMoDe).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

Rede

mt_sendconfig

```
int __stdcall mt_sendconfig(int ID, TSetupTCP *config);
```

Envia configuração para o terminal (IDvSetSetupTCP).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

```
typedef struct  
{  
    DWORD microT_IP;    // Endereço IP do terminal  
    DWORD server_IP;    // Endereço IP do servidor  
    DWORD msknet_IP;    // Máscara de rede  
    DWORD bDHCP;        // 1 = IP dinâmico, 0 = IP fixo.  
}TSetupTCP;
```

mt_reqconfig

```
int __stdcall mt_reqconfig(int ID);
```

Requisita configuração do terminal (IDvGetSetupTCP). A DLL enviará uma mensagem cada vez que receber a resposta a este comando. Para receber a configuração utilize a função `mt_getconfig`.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getconfig

int __stdcall mt_getconfig (int ID, TSetupTCP* config);

Recebe os dados enviados pelo terminal ao enviar o comando IDvGetSetupTCP para o terminal. A DLL enviará uma mensagem cada vez que tiver dados a serem recebidos.

config: estrutura onde será salvo os dados recebidos.

retorna: <=0 buffer vazio, 1 se comando realizado com sucesso.

mt_sendexconfig

int __stdcall mt_sendexconfig(int ID, TExSetupTCP *config);

Envia configuração estendida para o terminal (IDvSetExSetupTCP).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

```
typedef struct
{
    DWORD gateway;        // IP do gateway
    DWORD nameserver;     // IP do servidor de nomes
    DWORD myname;         // Nome do terminal
}TExSetupTCP;
```

mt_reqconfig

int __stdcall mt_reqexconfig(int ID);

Requisita configuração do terminal (IDvGetExSetupTCP). A DLL enviará uma mensagem cada vez que receber a resposta a este comando. Para receber a configuração utilize a função mt_getexconfig.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getexconfig

int __stdcall mt_getexconfig (int ID, TExSetupTCP* config);

Recebe os dados enviados pelo terminal ao enviar o comando IDvGetExSetupTCP para o terminal. A DLL enviará uma mensagem cada vez que receber a resposta ao comando.

config: estrutura onde será salvo os dados recebidos.

retorna: <=0 buffer vazio, 1 se comando realizado com sucesso.

Display

mt_backspace

int __stdcall mt_backspace(int ID);

Envia o comando de *BackSpace* para o terminal (IDvBackSpace).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_carret

int __stdcall mt_carret (int ID);

Envia o comando de *CarriageReturn* para o terminal (IDvCarRet).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_linefeed

int __stdcall mt_linefeed (int ID);

Envia o comando de *LineFeed* para o terminal (IDvLineFeed).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_formfeed

int __stdcall mt_formfeed (int ID);

Envia o comando de *FormFeed* para o terminal (IDvFormFeed).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_gotoxy

int __stdcall mt_gotoxy (int ID, int lin, int col);

Envia o comando de *GotoXY* para o terminal (IDvGoToXY).

lin/col: Linha e Coluna onde será posicionado o cursor.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_gotoxyref

int __stdcall mt_gotoxyref(int ID, int lin, int col);

Envia o comando de *GotoXYRef* para o terminal (IDvGoToXYRef).

lin/col: Linha e Coluna onde será posicionado o cursor a partir da posição atual.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_dispstr

int __stdcall mt_dispstr(int ID, char *str);

Envia o comando de *DisplayString* para o terminal (IDvDispStr).

str: string que será enviada para o display.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_dispch

int __stdcall mt_dispch (int ID, char ch);

Envia o comando de *DisplayCharacter* para o terminal (IDvDispCh).

ch: Character que será enviado para o display.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_dispclrln

int __stdcall mt_dispclrln(int ID, int lin);

Envia o comando de *DisplayClearLine* para o terminal (IDvDispClrLn).

lin: linha do display que será apagada.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_seteditstring

int __stdcall mt_seteditstring(int ID, BYTE *String, BOOL OnOff, BOOL PassWord);

Envia o comando de *EditString* para o terminal (IDvSetEditString).

String: String inicial da edição de texto.

OnOff: ativa (1) e desativa (0) o modo de edição de string.

PassWord: habilita (1) e desabilita (0) o modo de edição em modo protegido (senha).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_reqeditstring

int __stdcall mt_reqeditstring(int ID);

Requisita o estado do *EditString* do terminal (IDbGetEditString). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_geteditstring

int __stdcall mt_geteditstring (int ID, char *buftecla);

Recebe os dados enviados pelo terminal ao enviar o comando IDvSetEditString para o terminal. A DLL enviará uma mensagem cada vez que tiver dados a serem recebidos.

buftecla: string que será recebida após pressionar a tecla Enter no microterminal.

retorna: <=0 buffer vazio, 1 se comando realizado com sucesso.

Teclado

mt_setenablekey

int __stdcall mt_setenablekey(int ID, const BOOL OnOff);

Envia o comando de *EnableKey* para o terminal (IDvSetEnableKey).

OnOff: ativa (1) e desativa (0) o teclado do terminal.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getenablekey

int __stdcall mt_getenablekey(int ID);

Requisita o estado do *EnableKey* do terminal (IDbGetEnableKey). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_reset

int __stdcall mt_reset(int ID);

Envia o comando de *Reset* para o terminal (IDvReset).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_setcapslock

int __stdcall mt_setcapslock(int ID, const BOOL OnOff);

Envia o comando de *SetCapsLock* para o terminal (IDvSetCapsLock).

OnOff: ativa (1) e desativa (0) o CapsLock.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getcapslock

int __stdcall mt_getcapslock(int ID);

Requisita o estado do *CapsLock* do terminal (IDbGetCapsLock). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_setnumlock

int __stdcall mt_setnumlock(int ID, const BOOL OnOff);

Envia o comando de *SetNumLock* para o terminal (IDvSetNumLock).

OnOff: ativa (1) e desativa (0) o NumLock.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getnumlock

int __stdcall mt_getnumlock(int ID);

Requisita o estado do *NumLock* do terminal (IDbGetNumLock). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_programkbd

int __stdcall mt_programkbd(int ID, BYTE *Codigo);

Envia o comando de *ProgramKeyboard* para o terminal (IDbProgramKbd). A DLL enviará uma mensagem com a resposta deste comando.

Codigo: ponteiro para onde está o conteúdo do arquivo .bin com tabela de códigos do teclado.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_setbeep

int __stdcall mt_setbeep(int ID, const BOOL OnOff);

Envia o comando de *SetBeep* para o terminal (IDvSetBeep).

OnOff: ativa (1) e desativa (0) o Beep.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_setbeepkey

int __stdcall mt_setbeepkey(int ID, const BOOL OnOff);

Envia o comando de *SetBeepkey* para o terminal (IDvSetBeepKey).

OnOff: ativa (1) e desativa (0) o Beep de tecla.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getkey

int __stdcall mt_getkey(int ID, char *buf);

Recebe uma tecla do terminal (IDcGetCharTerm).

buf: tecla recebida.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

Serial

mt_setenableserial

int __stdcall mt_setenableserial(int ID, const BYTE COM, const BOOL OnOff);

Envia o comando de *SetEnableSerial* para o terminal (IDvSetEnableSerial).

COM: porta serial, deve ser sempre = 0 (COM1).

OnOff: ativa (1) e desativa (0) a serial.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getenableserial

int __stdcall mt_getenableserial(int ID, const BYTE COM);

Requisita o estado do *EnableSerial* do terminal (IDvGetEnableSerial). A DLL enviará uma mensagem com a resposta deste comando.

COM: porta serial, deve ser sempre = 0 (COM1).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_sendbinserial

int __stdcall mt_sendbinserial(int ID, const BYTE COM, LPBYTE Bin, BYTE tam);

Envia o comando de *SendBinSerial* para o terminal (IDbSendBinSerial).

COM: porta serial, deve ser sempre = 0 (COM1).

Bin: dados que serão enviados para a serial do terminal.

tam: quantidade de dados que serão enviados.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getserial

int __stdcall mt_getserial(int ID, int *sercom, char *buf);

Envia o comando de *GetSerial* para o terminal (IDbGetBinSerial). A DLL enviará uma mensagem cada vez que tiver dados a serem recebidos.

sercom: de qual porta serial foram lidos os dados 0 = COM1.

buf: os dados da serial serão escritos nesta variavel

retorna: numero de bytes lidos da serial.

mt_sendconfigserial

int __stdcall mt_sendconfigserial(int ID, ARG_COM_SETUPSERIAL *config);

Envia o comando de *SendConfigSerial* para o terminal (IDbSetSetupSerial).

Config: estrutura onde será salvo os dados recebidos.

retorna: 0 houve erro, outro valor se comando realizado com sucesso.

```
typedef struct {  
    unsigned long baud;           // baudrate: 300 a 115.200  
    unsigned short bits;         // data bits  
    unsigned short parity;       // paridade  
    unsigned short stops;        // stop bits  
    unsigned char handshaking;    // 0 = sem handshaking, 1 = RTS/CTS  
} TSetupSerial;  
typedef struct {
```

```
    unsigned char Com;  
    TSetupSerial Setup;  
}ARG_COM_SETUPSERIAL;
```

mt_reqconfigserial

```
int __stdcall mt_reqconfigserial(int ID, BYTE COM);
```

Requisita a configuração da serial do terminal (IDvGetSetupSerial). A DLL enviará uma mensagem cada vez

que receber a resposta a este comando. Para receber a configuração utilize a função mt_getconfig.

COM: porta serial, deve ser sempre = 0 (COM1).

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getconfigserial

```
int __stdcall mt_getconfigserial(int ID, ARG_COM_SETUPSERIAL *config);
```

Recebe os dados enviados pelo terminal ao enviar o comando IDvGetSetupSerial para o terminal. A DLL enviará uma mensagem cada vez que tiver dados a serem recebidos.

config: estrutura onde será salvo os dados recebidos.

retorna: <=0 buffer vazio, 1 se comando realizado com sucesso.

mt_settermserial

```
int __stdcall mt_settermserial(int ID, const BYTE COM, const BYTE TERM, const BOOL OnOff);
```

Configura o caractere de terminador da porta serial ao enviar o comando IDvSetTermSerial para o terminal.

COM: porta serial, deve ser sempre = 0 (COM1).

TERM: Byte de terminador.

OnOff: ativa (1) e desativa (0) esta funcionalidade.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

Cartão Magnético

mt_sendsetcard

int __stdcall mt_sendsetcard(int ID, const BOOL OnOff);

Envia o comando de IDvSetCard para o terminal.

OnOff: ativa (1) e desativa (0) a serial.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_reqgetcard

int __stdcall mt_reqgetcard(int ID);

Requisita o estado do cartão magnético do terminal (IDbGetCard). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_getcardbuf

int __stdcall mt_getcardbuf(int ID, ARG_CARD *cardbuf);

Recebe os dados do cartão magnéticos enviados pelo terminal com o comando IDbReadBuffCard. A DLL enviará uma mensagem cada vez que tiver dados a serem recebidos.

config: estrutura onde serão salvos, os dados recebidos.

retorna: <=0 buffer vazio, 1 se comando realizado com sucesso.

```
typedef struct {  
    unsigned char card[128];  
    unsigned long status;  
} ARG_CARD;
```

Impressora

mt_sendinitprn

int __stdcall mt_sendinitprn(int ID);

Envia o comando de IDcInitPrn para o terminal.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_reqgetstatusprn

int __stdcall mt_reqgetstatusprn(int ID);

Requisita o estado da impressora do terminal (IDcGetStatusPrn). A DLL enviará uma mensagem com a resposta deste comando.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

mt_sendbinprn

int __stdcall mt_sendbinprn(int ID, LPBYTE Bin, BYTE tam);

Envia o comando de *IDcSendPrn* para o terminal (IDbSendBinSerial).

Bin: dados que serão enviados para a serial do terminal.

tam: quantidade de dados que serão enviados.

retorna: <=0 houve erro, 1 se comando realizado com sucesso.

A troca de mensagens do programa principal com a DLL

Para aumentar a agilidade de troca de informação da DLL com o programa principal do servidor e evitar processamento desnecessário, foi implementado a troca de mensagens. A DLL envia mensagens para o programa principal sempre que ocorrer em evento entre o terminal e a DLL. Para facilitar o entendimento, utilizaremos o C++ Builder® como exemplo.

Para receber estas mensagens, o servidor deve chamar a função da DLL **mc_startserver** da seguinte forma:

```
#define COMMUNICATION_MSG WM_USER + 1
#define CONNECT_MSG WM_USER + 2
mc_startserver(Form1->Handle,CONNECT_MSG,COMMUNICATION_MSG);
```

Onde Form1 corresponde ao formulário (janela) principal, CONNECT_MSG corresponde à mensagem que o servidor enviará quando um terminal conectar/desconectar e COMMUNICATION_MSG corresponde à mensagem que o servidor enviará quando um terminal enviar dados.

Devemos “redefinir” a função WndProc do formulário para podermos receber as mensagens. Para isso devemos seguir os seguintes passos:

1) No arquivo de header (geralmente unit1.h) devemos adicionar na classe do formulário, a chamada para a função:

```
...
private: // User declarations
    virtual void __fastcall WndProc(Messages::TMessage &Message);
...

```

2) Devemos então, “reescrever” esta função (unit1.cpp):

```
void __fastcall TForm1::WndProc(Messages::TMessage &Message)
{
    if (Message.Msg == COMMUNICATION_MSG)
    {
        //recebe mensagens enviadas pelo terminal
        return;
    }
    else if (Message.Msg == CONNECT_MSG)
    {
        //recebe mensagens quando um terminal conectou/desconectou
        return;
    }
}
```

```
TForm::WndProc(Message); //chama WndProc antiga
```

```
}
```

Para saber como tratar as mensagens recebidas, fornecemos o código fonte do servidor para ser tomado como exemplo.

CONTATOS GERTEC

Suporte Técnico:

suporte@gertec.com.br

Tel.: (11) 25751000

Assistência Técnica:

Gertec Telecomunicações Ltda.

Rua Guaicurus, 145 – Vila Conceição

Diadema – São Paulo

CEP 09911-630

Tel.: (11) 2173-6500