

# **DOCUMENTAÇÃO DLL PMTG V1.4**

## **PMTG.DLL**

**REV OUTUBRO/2007**

## **INICIALIZAÇÃO E FINALIZAÇÃO..... 4**

MT_STARTSERVER .....	4
MT_FINISHSERVER.....	4
MT_VERSION.....	4
MT_CONNECTLIST .....	4

## **FUNÇÕES DE TRATAMENTO DE IP..... 4**

MT_GETHOSTIP.....	4
MT_INET_NTOA .....	5
MT_INET_NTOA_INV.....	5
MT_INET_ADDR .....	5
MT_INET_ADDR_INV.....	5
MT_IPFROMID.....	5

## **COMANDOS BÁSICOS PARA O TERMINAL ..... 5**

MT_SENDLIVE .....	5
MT_RESTART .....	6
MT_FTPMODE .....	6

## **REDE ..... 6**

MT_SENDCONFIG .....	6
MT_REQCONFIG.....	6
MT_GETCONFIG.....	6
MT_SENDEXCONFIG .....	7
MT_REQCONFIG.....	7
MT_GETEXCONFIG.....	7

## **DISPLAY ..... 7**

MT_BACKSPACE.....	7
MT_CARRET.....	7
MT_LINEFEED .....	8
MT_FORMFEED .....	8
MT_GOTOXY .....	8
MT_GOTOXYREF .....	8
MT_DISPSTR.....	8
MT_DISPCH .....	8
MT_DISPCLRLN .....	9
MT_SETEDITSTRING.....	9

MT_REQEDITSTRING.....	9
MT_GETEDITSTRING.....	9

## **TECLADO ..... 9**

MT_SETENABLEKEY.....	9
MT_GETENABLEKEY.....	9
MT_RESET.....	10
MT_SETCAPSLOCK.....	10
MT_GETCAPSLOCK.....	10
MT_SETNUMLOCK.....	10
MT_GETNUMLOCK.....	10
MT_PROGRAMKBD.....	10
MT_SETBEEP.....	11
MT_SETBEEPKEY.....	11
MT_GETKEY.....	11

## **SERIAL..... 11**

MT_SETENABLESERIAL.....	11
MT_GETENABLESERIAL.....	11
MT_SENDBINSERIAL.....	12
MT_GETSERIAL.....	12
MT_SENDCONFIGSERIAL.....	12
MT_REQCONFIGSERIAL.....	12
MT_GETCONFIGSERIAL.....	13
MT_SETTERMSERIAL.....	13

## **CARTÃO MAGNÉTICO ..... 13**

MT_SENSETCARD.....	13
MT_REQGETCARD.....	13
MT_GETCARDBUF.....	13

## **IMPRESSORA..... 14**

MT_SENDINITPRN.....	14
MT_REQGETSTATUSPRN.....	14
MT_SENDBINPRN.....	14

## **A TROCA DE MENSAGENS DO PROGRAMA PRINCIPAL COM A DLL ..... 14**

## Inicialização e Finalização

### ***mt\_startserver***

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

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.

commumsg: 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\_dispcrln***

int \_\_stdcall mt\_dispcrln(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);

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.

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 variável

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.