

標準 P L M C 対応 Gコード仕様テキストプログラム変換ライブラリー

P L M - G C N V 説明書

Ver 1.0
2002.06.29

- 目 次 -

1 . 形式	3
2 . 製品構成	3
3 . 概要	3
4 . 関連資料	3
5 . 使用環境	4
6 . インストール	4
7 . ライブラリ使用上の注意事項	4
8 . 関数使用例	5
8 - 1 . MS - VC以外のC言語処理系での利用方法	5
9 . 関数仕様	7
MemGcodeConv	7
MemTaskGcodeConv	7
MemGcodeRConv	8
FileGcodeConv	9
FileTaskGcodeConv	9
FileGcodeRConv	10
DncGcodeConv	11
GcdCircleSet	12
GcdSetProgramMemSize	12
GcdFeedUnitSet	13
GcdSTSelect	13
GcdPaccTimeSet	13
GcdPositionUnitSet	14
GcdCircleMode	14
GcdDiametralAxis	15
GcdSetMultiplier	15
GcdGetErrLine	16
GcdSetZXPlane	16

1. 形式

PLM - GCNV

2. 製品構成

PLMGCNV.DLL	テストプログラム変換ライブラリ
PLMGCNV.LIB	テストプログラム変換インポートライブラリ
PLMGCNV.H	プログラム変換関数定義ファイル
PLMGCDAPI.TXT	ビジュアルベーシック用データ型・関数定義

3. 概要

プログラム変換ライブラリは、テキストファイル形式のGコードプログラムを解析し、バイナリ形式のプログラムに変換するものです。
本ライブラリの入出力は、以下の2通りがあります。

テキスト入力	バイナリー出力
メモリ (メモリアドレス指定)	メモリ
ファイル (ファイル名指定)	メモリ

《テキストプログラム例》

```
G90 G00 X0 Y0 Z0 A0;  
M98 P100;  
G04 P1.0;  
G90 G00 X100 Y100 Z100 A100;  
M98 P100;  
G04 P1.0;  
G03 X400 Y0 Z100 I200 J0 F1000;  
M30;  
  
N100  
G00 X100 Y-100 Z100 A-100;  
G00 X-100 Y100 Z-100 A100;  
G01 X1500 Y2000 Z2500 A3000 F1000;  
M99;
```

4. 関連資料

「PLMC40ユーザーズマニュアル」	(TB00-0810)
「標準PLMC対応 セットアップPCマニュアル」	(TB00-0812)
「標準PLMC対応 ROMSW設定ソフトマニュアル」	(TB00-0811)
「標準PLMC対応 サンプルラダープログラム説明書」	(TB00-0817)
「標準PLMC対応 通信ライブラリリファレンスマニュアル」	(TB00-0813)
「標準PLMC対応 送受信データ説明書」	(TB00-0814)
「標準PLMC対応 Tコード変換ライブラリリファレンスマニュアル」	(TB00-0815)

5 . 使用環境

本ライブラリが対応するパソコン及び、Cコンパイラは以下の通りです。

対応パソコン . . . Windows95/98/NT/2000が動作するx86CPU搭載機
対応Cコンパイラ . . . Visual C++ Ver6.0以降

6 . インストール

DLLファイルをプログラム検索パスの通っているディレクトリ(フォルダ)に格納して下さい。

7 . ライブラリ使用上の注意事項

- ・ 円弧補間指令をプリ解析指定で変換する場合、複数の微小直線補間指令ステップに分解されます。プリ解析にて円弧ステップが何ステップに分解されるかは、「PLMC40ユーザーズマニュアル機能編 補足資料・1 円弧プリ解析処理(ステップ数/処理時間)」を参照して下さい。円弧補間指令を含むプログラムの場合、分解後のステップ数が最大ステップ数をこえないようにして下さい。
- ・ 位置決めポイントデータとプログラムステップは共通のバッファを使用しています。このため、位置決めポイントデータを指定すると最大プログラムステップ数は、関数"SetProgMemSize"にて設定したステップ数以下になります。プログラムステップと位置決めポイントデータ数の関係についてはユーザーズマニュアルを参照下さい。
- ・ PositionUnitSet関数は全ての軸指定の小数点位置を変更します。従って、PositionUnitSet関数をSetMultiplier関数の後で呼び出すと、SetMultiplier関数で設定した軸毎の小数点位置がPositionUnitSet関数で指定した小数点位置に置き換わってしまいます。SetMultiplier関数は、必ずPositionUnitSet関数の後で呼び出すようにしてください。
- ・ メインプログラムは必ず先頭に記述してください。
- ・ テキストプログラムの詳細な説明については、「PLMC40ユーザーズマニュアル」を参照してください。

8 . 関数使用例

本ライブラリの使用法は、以下の2通りがあります。

- ・インポートライブラリを使用して関数をインポートする。
ライブラリーに付属のインポートライブラリをアプリケーションの作成時にリンクします。
この方法は、使用言語がMS - VC++ Ver 6 . 0の時のみ有効です。
- ・DLLロード関数を使用して関数をインポートする。
DLLロード関数を使用してDLLをメモリにロードし、関数アドレスを取得して関数を呼び出します。
この方法は、ウィンドウズ上で動作する言語処理系（32ビット版）全てで有効です。
詳細は次項を参照して下さい。

8 - 1 . MS - VC以外のC言語処理系での利用方法

MS - VC以外のC言語処理系でDLLを使用する場合は、以下の手順で行います。

- 1 . DLLのロード
- 2 . DLL内の任意の関数のアドレス取得
:
: DLL内の関数の利用
:
3 . DLLのアンロード

関数アドレス取得時に指定する関数名は、本説明書(ヘッダーファイル)の関数名と異なります。
アドレス取得時の関数名のフォーマットは、ヘッダーファイルの関数名の前に_(アンダーバー)を付け、関数名の後に@、その後に引数の数×4を付けます。
具体的には以下の通りとなります。

`_関数名@引数の数×4`

例 MemGcodeConv(LPBYTE *text*, LPLONG *size*, LPVOID *bin*)の場合
関数名 _MemGcodeConv@12

これらの定義がPlmgcnv.h内にあるので、独自の処理を作成する場合は参考にして下さい。

関数アドレス取得時に指定する関数名/関数ポインタのデータ型は、Plmgcnv.h内で定義しています。Plmgcnv.h をインクルードする前に LOADLIBRARY_SXDLL を定義 (define) することによってこれらの定義を使用できるようになります。(LOADLIBRARY_MCDLLを定義しなかった場合はインポートライブラリを使用する設定になります。)

以下の例は、プログラム変換ライブラリの関数取得例を示します。

```
#define LOADLIBRARY_MCDLL          // テクノ製ライブラリ明示的リンク指定
#include "Plmgcnv.h"              // Gコード変換ライブラリ宣言

// グローバルデータ定義
HINSTANCE      hGcdInst;          /* プログラム変換ライブラリハンドル */
MEMPRGCNV     *MemGcodeConv;     /* Gコードテキスト -> バイナリ変換関数へのポインタ */
MEMPRGRCNV    *MemGcodeRConv;    /* バイナリ -> Gコードテキスト変換関数へのポインタ */

// Gコードプログラム変換ライブラリ関数ロード処理
int LoadGcdcncv(void)
{
    // DLLのロード
    if(NULL == (hGcdInst = LoadLibrary(GLNLCNVDLL))){
        return 0;
    }
    // DLLが正常にロードできた場合
    // DLL関数ポインタの取得
    MemGcodeConv = (MEMPRGCNV *)GetProcAddress(hGcdInst, GFNMEMPRGCNV);
    MemGcodeRConv = (MEMPRGRCNV *)GetProcAddress(hGcdInst, GFNMEMPRGRCNV);

    if((NULL == MemGcodeConv) || (NULL == MemGcodeRConv)){
        // 関数アドレス取得が失敗した場合
        // DLLの解放
        FreeLibrary(hGcdInst);
        return 0;
    }

    return 1;
}
```

9 . 関数仕様

MemGcodeConv

MemTaskGcodeConv

機能

テキスト形式プログラムをバイナリー形式プログラムに変換します。
マルチタスクのプログラムを変換する場合は、MemTaskGcodeConv()を使用下さい。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short MemGcodeConv(LPBYTE text, LPWORD size, LPVOID bin);  
ASISSAPI short MemTaskGcodeConv(LPBYTE text, LPWORD size, LPVOID bin, WORD task);
```

LPBYTE *text* テキストプログラム格納バッファポインタ

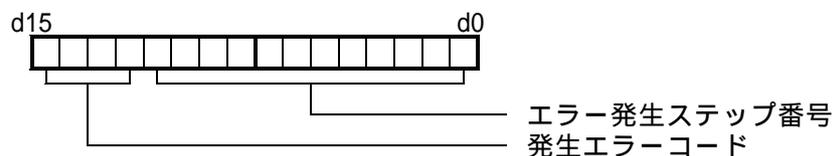
LPLONG *size* バイナリープログラムサイズ

LPVOID *bin* バイナリープログラム格納バッファポインタ

WORD *task* プログラムタスク指定番号
 タスク指定番号については「標準PLMC対応 送受信データ説明書
 3-1-4.マルチタスクプログラム書込/読込」を参照下さい。

戻り値

変換処理実行ステータスとして以下の値を返します。



エラーコード = 2 : プログラムバッファオーバーフロー
 = 3 : テキストプログラムフォーマットエラー
 = 4 : プログラム変換演算エラー
 = 5 : 作業メモリオーバーフロー

上記ステータスが全て0の場合、正常終了

MemGcodeRConv

機能

バイナリー形式プログラムをテキスト形式プログラムに変換します。

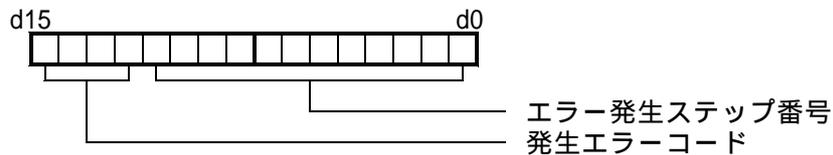
プロトタイプ

```
#include "Plmgcnv.h"      関数宣言に必要なインクルードファイル
ASISSAPI short MemGcodeRConv(LPVOID bin, LPWORD size, LPBYTE text);

LPVOID          bin      バイナリープログラム格納バッファポインタ
LPWORD          size     バイナリープログラムサイズ
LPBYTE          text     テキストプログラム格納バッファポインタ
```

戻り値

変換処理実行ステータスとして以下の値を返します。



エラーコード = 2 : プログラムバッファオーバーフロー
 = 5 : 作業メモリアーオーバーフロー

上記ステータスが全て0の場合、正常終了

FileGcodeConv

FileTaskGcodeConv

機能

テキスト形式プログラムをバイナリー形式プログラムに変換します。
マルチタスクのプログラムを変換する場合は、FileTaskGcodeConv()を使用下さい。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short FileGcodeConv(LPBYTE fname, LPWORD size, LPVOID bin);  
ASISSAPI short FileTaskGcodeConv(LPBYTE fname, LPWORD size, LPVOID bin, WORD task);
```

LPBTE *fname* テキストプログラム格納ファイル名

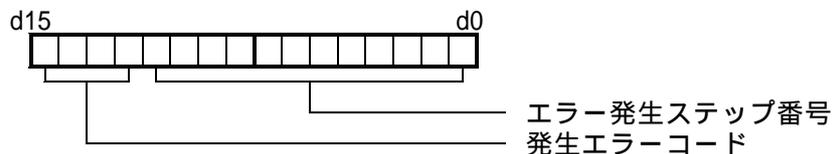
LPWORD *size* バイナリープログラムサイズ

LPVOID *bin* バイナリープログラム格納バッファポインタ

WORD *task* プログラムタスク指定ビットフラグ
タスク指定番号については「標準PLMC対応 送受信データ説明書
3-1-4.マルチタスクプログラム書込/読込」を参照下さい。

戻り値

変換処理実行ステータスとして以下の値を返します。



エラーコード = 1 : テキストプログラムファイルエラー
= 2 : プログラムバッファオーバーフロー
= 3 : テキストプログラムフォーマットエラー
= 4 : プログラム変換演算エラー
= 5 : 作業メモリアーオーバーフロー

上記ステータスが全て0の場合、正常終了

FileGcodeRConv

機能

バイナリー形式プログラムをテキスト形式プログラムに変換します。

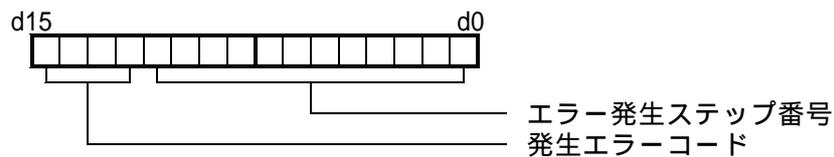
プロトタイプ

```
#include "Plmgcnv.h"      関数宣言に必要なインクルードファイル
ASISSAPI short FileGcodeRConv(LPBYTE fname, LPWORD size, LPVOID text);

LPBYTE      fname      バイナリープログラム格納ファイル名
LPWORD      size       テキストプログラムサイズ
LPVOID      text       テキストプログラム格納バッファポインタ
```

戻り値

変換処理実行ステータスとして以下の値を返します。



エラーコード = 1 : バイナリープログラムファイルエラー
= 2 : プログラムバッファオーバーフロー
= 5 : 作業メモリアーオーバーフロー

上記ステータスが全て0の場合、正常終了

D n c G c o d e C o n v

機能

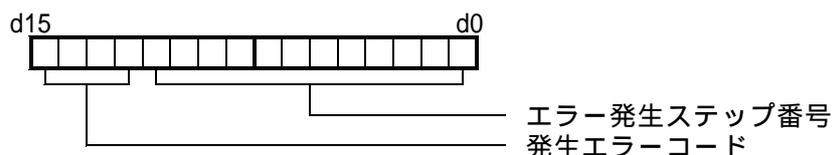
テキスト形式プログラムをDNC運転用バイナリ形式プログラムに変換します。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
ASISSAPI short DncProgramConv(LPBYTE *text, LPLONG size, LPVOID bin, short cont);
LPBYTE *text テキストプログラム格納バッファポインタ
LPLONG size バイナリプログラムサイズ格納ポインタ
LPVOID bin バイナリプログラム格納バッファポインタ
short cont 先頭 / 継続指定フラグ
```

戻り値

変換処理実行ステータスとして以下の値を返します。



エラーコード	= 2	:	プログラムバッファオーバーフロー
	= 3	:	テキストプログラムフォーマットエラー
	= 4	:	プログラム変換演算エラー
	= 5	:	作業メモリアーバーフロー

上記ステータスが全て0の場合、正常終了

概要

*text*が示すバッファに格納された、Gコードプログラム文字列を、null文字が見つかるまでDNCバイナリデータに変換します。

変換後のデータは*bin*が示すバッファに格納され、この時の総バイト数が*size*に返されます。また、一連のDNC運転プログラムにおける先頭部分の変換である時 *cont*=0を、続きのデータである場合 *cont*=1を指定します。

注意事項

サポートしているGコード機能中、下記のコードについてはDNC運転では使用できませんので注意して下さい。

M 9 8	:	サブプログラム呼出
G 1 0 0	:	ポイント指定PTP移動
P N T	:	位置決めポイント設定

GcdCircleSet

機能

円弧補間計算精度を設定します。(デフォルト値は10パルス)

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdCircleSet(short size);
```

```
short size 円弧補間計算精度 1 ~ 1000 (パルス)
```

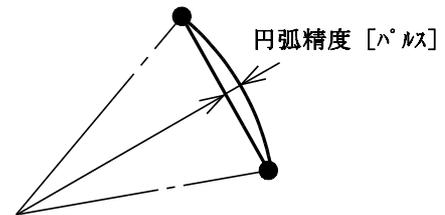
プリ解円弧軌跡計算をする際の理想円と実際の指令位置との最大のずれ量です。
(図参照)

本ライブラリでは、円弧形状を微小直線の集合に変換する事により実現しています。

円弧補間計算精度を高くする(=本設定値を小さくする)と微小直線の数が増えるため、変換後のデータ量が大きくなります。

円弧補間計算精度を低くする(=本設定値を大きくする)と、変換のデータ量を小さく押さえる事ができます。

プリ解円弧と内部円弧の違いについてはユーザーズマニュアルを参照下さい。



戻り値

正常終了の場合、戻り値として0を返します。
円弧精度の指定値が設定範囲外の場合、-1を返します。

GcdSetProgramMemSize

機能

プログラムメモリ容量(1プログラムあたりの最大プログラムステップ数)を設定します。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdSetProgMemSize(short mflg);
```

```
short mflg メモリー容量指定フラグ  
PROG_SIZE_64K : デフォルト  
PROG_SIZE_32K  
PROG_SIZE_16K
```

各フラグとプログラムメモリ容量の関係は下表の通りです。

フラグ	最大ステップ数	最大パルス数	最大位置決まり数
PROG_SIZE_64K	999	95	999
PROG_SIZE_32K	499	45	499
PROG_SIZE_16K	245	22	249

戻り値

正常終了の場合、戻り値として0を返します。
メモリ容量の指定値が設定範囲外の場合、-1を返します。

GcdFeedUnitSet

機能 補間移動時の F 指定の単位を設定します。(デフォルトは [パルス/秒])

プロトタイプ

```
#include "Plmgnv.h" 関数宣言に必要なインクルードファイル

ASISSAPI short GcdFeedUnitSet(long pluse, short timef);

long          pluse   単位時間あたりの移動量   1 ~ 2 0 0 0 0 0 パルス
short        timef   単位時間フラグ
                  SEC__UNIT   . . .   秒
                  MIN__UNIT   . . .   分
```

戻り値 正常終了の場合、戻り値として 0 を返します。
送り速度単位の指定値が設定範囲外の場合、- 1 を返します。

GcdSTSelect

機能 P L M C の制御周期を設定します。

プロトタイプ

```
#include "Plmgnv.h" 関数宣言に必要なインクルードファイル

ASISSAPI short GcdSTSelect(short type);

short          type   P L M C の制御周期を m s e c で指定します。
                  P L M C の制御周期は R O M S W 設定ソフトで御確認下さい。
```

戻り値 正常終了の場合、戻り値として 0 を返します。
指定値が設定範囲外の場合、- 1 を返します。

GcdPaccTimeSet

機能 プリ解加減速機能の加減速時間を指定します。

プロトタイプ

```
#include "Plmgnv.h" 関数宣言に必要なインクルードファイル

ASISSAPI short GcdPaccTimeSet(short Time);

short          Time   プリ解加減速時間   5 0 msec ~ 5 0 0 msec
```

戻り値 正常終了の場合、戻り値として 0 を返します。
指定値が設定範囲外の場合、- 1 を返します。

GcdPositionUnitSet

機能

全軸の座標 / 移動量指定、円弧半径指定の小数点単位を指定します。
(" X 1 ." や " R 1 ." と記述時の倍率指定)

PositionUnitSet関数は全ての軸指定の小数点位置を変更します。

従って、PositionUnitSet関数をSetMultiplier関数の後で呼び出すと、SetMultiplier関数で設定した軸毎の小数点位置がPositionUnitSet関数で指定した小数点位置に置き換わってしまいます。

SetMultiplier関数は、必ずPositionUnitSet関数の後で呼び出すようにしてください。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdPositionUnitSet(short flag);
```

short	<i>flag</i>	乗数フラグ
		0 : 1 [パルス]
		1 : 1 0
		2 : 1 0 0
		3 : 1 0 0 0
		4 : 1 0 0 0 0
		5 : 1 0 0 0 0 0

戻り値

正常終了の場合、戻り値として0を返します。
座標 / 移動量単位フラグが設定範囲外の場合、 - 1を返します。

GcdCircleMode

機能

円弧補間解析モード設定を指定します。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdCircleMode(short md);
```

short	<i>md</i>	円弧補間解析処理モード	0:プリ解、1:SLM内部解析
-------	-----------	-------------	-----------------

戻り値

正常終了の場合、戻り値として0を返します。
円弧補間解析処理モードが設定範囲外の場合、 - 1を返します。

GcdDiametralAxis

機能 直径指令軸設定処理を指定します。
本機能はオプションです。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdDiametralAxis(short ax);
```

short *ax* 直径指令軸フラグ

戻り値 正常終了の場合、戻り値として0を返します。
直径指令軸指定を無効軸に対して指定すると、-1を返します。

GcdSetMultiplier

機能 各軸の小数点位置を指定します。
本関数では、円弧半径指定には影響しません。
本関数と、PositionUnitSet関数を同時に使用するときは注意が必要です。
(PositionUnitSet関数の説明を参照して下さい。)

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル
```

```
ASISSAPI short GcdSetMultiplier(long (*tbl)[4]);
```

long (**tbl*)[4] 各軸小数点位置
 1 ~ 1 0 0 0 0 0 [パルス]
 必ず 10^N の値を設定してください。

戻り値 常に0 (正常終了) を返します。

GcdGetErrLine

機能 プログラムエラー行数を取得します。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル  
ASISSAPI long GcdGetErrLine(void);
```

戻り値 フォーマットエラーが発生した行数を返します。
エラー行がないときは最終行が返ります。

GcdSetZXPlane

機能 X軸 / Z軸円弧の平面を選択します。

プロトタイプ

```
#include "Plmgcnv.h" 関数宣言に必要なインクルードファイル  
ASISSAPI long GcdSetZXPlane(short zx_flg);  
short zx_flg ZX平面指定フラグ 0 : XZ平面、1 : ZX平面
```

戻り値 正常終了の場合、戻り値として0を返します。
ZX平面指定フラグの値が不正の場合、-1を返します。