標準ＰＬＭＣ－ＭⅡ対応
Ｇコード仕様テキストプログラム変換ライブラリ
リファレンスマニュアル

ＰＬＭ２－ＧＣＮＶ 説明書

Ｖｅｒ１．０
2007.１.22

― 目 次 ―

１．形式

ＰＬＭ２－ＧＣＮＶ

２．製品構成

ＰＬＭ２ＧＣＮＶ．ＤＬＬ	･･･････････	ﾃｷｽﾄﾌﾟﾛｸﾞﾗﾑ変換ﾀﾞｲﾅﾐｯｸﾘﾝｸﾗｲﾌﾞﾗﾘ
ＰＬＭ２ＧＣＮＶ．ＬＩＢ	･･･････････	ﾃｷｽﾄﾌﾟﾛｸﾞﾗﾑ変換ｲﾝﾎﾟｰﾄﾗｲﾌﾞﾗﾘ
ＰＬＭ２ＧＣＮＶ．Ｈ	･･･････････	ﾌﾟﾛｸﾞﾗﾑ変換関数定義ﾌｧｲﾙ
ＰＬＭ２ＧＣＤＡＰＩ．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 L10;
G04 P1.0;
G90 G00 X100 Y100 Z100 A100;
M98 P100 L5;
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

「ＰＬＭＣ－ＭⅡユーザーズマニュアル」	（ＴＢ００－０８６０）
「標準ＰＬＭＣ－ＭⅡ対応 セッティングＰＣマニュアル」	（ＴＢ００－０８６１）
「標準ＰＬＭＣ－ＭⅡ対応 ＲＯＭＳＷ設定ソフトマニュアル」	（ＴＢ００－０８６２）
「標準ＰＬＭＣ－ＭⅡ対応 サンプルラダープログラム説明書」	（ＴＢ００－０８１７）
「標準ＰＬＭＣ－ＭⅡ対応 通信ライブラリリファレンスマニュアル」	（ＴＢ００－０８６３）
「標準ＰＬＭＣ－ＭⅡ対応 送受信データ説明書」	（ＴＢ００－０８６４）
「標準ＰＬＭＣ－ＭⅡ対応 Ｔコード変換ライブラリリファレンスマニュアル」	（ＴＢ００－０８６５）

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/XPが動作するｘ８６ＣＰＵ搭載機
対応Ｃコンパイラ　・・・　Ｖｉｓｕａｌ Ｃ＋＋　Ｖｅｒ６．０

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、「ＰＬＭＣ－ＭⅡ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、
「ＰＬＭＣ－ＭⅡ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＋＋ Ｖｅｒ６．０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Plm2gcnv.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、Plm2gcnv.h内で
定義しています。
Plm2gcnv.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 "Plm2gcnv.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ｺｰﾄﾞﾃｷｽﾄ -> ﾊﾞｲﾅﾘ変換関数へのﾎﾟｲﾝﾀ  */
MEMPRGRCNV      *MemGcodeRConv;         /* ﾊﾞｲﾅﾘ -> Gｺｰﾄﾞﾃｷｽﾄ変換関数へのﾎﾟｲﾝﾀ  */

// Ｇコードプログラム変換ライブラリ関数ロード処理
int LoadGcdcnv(void)
{
    // ＤＬＬのロード
    if(NULL == (hGcdInst = LoadLibrary(GLNCNVDLL))){
          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;
}
```

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。

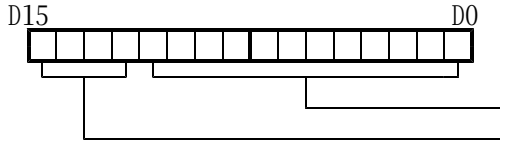
プロトタイプ

```
#include    "Plm2gcnv.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－ＭⅡ対応 送受信データ説明書4-1-4.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

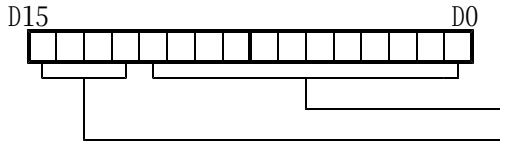
```
#include    "Plm2gcnv.h"        関数宣言に必要なインクルードファイル
ASISSAPI short MemGcodeRConv(LPVOID bin, LPWORD size, LPBYTE text);
```

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。



エラー発生ステップ番号
発生エラーコード

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。

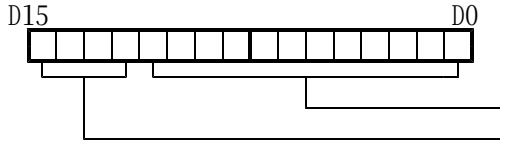
プロトタイプ

```
#include    "Plm2gcnv.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－ＭⅡ対応 送受信データ説明書4-1-4.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

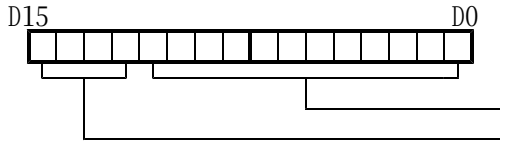
```
#include    "Plm2gcnv.h"        関数宣言に必要なインクルードファイル
ASISSAPI short FileGcodeRConv(LPBYTE fname, LPWORD size, LPVOID text);
```

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。



エラー発生ステップ番号
発生エラーコード

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

ＤｎｃＧｃｏｄｅＣｏｎｖ

機能　　テキスト形式プログラムをＤＮＣ運転用バイナリー形式プログラムに変換します。

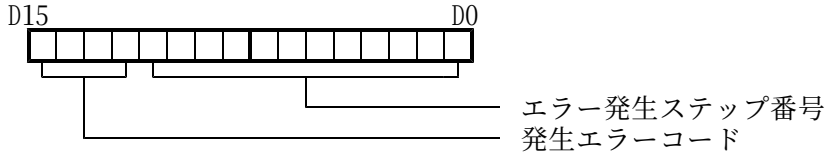
プロトタイプ

```
#include    "Plm2gcnv.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*	バイナリープログラムサイズ格納ポインタ
LPVOID	*bin*	バイナリー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

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。

Ｍ９８　：サブプログラム呼出
Ｇ１００：ポイント指定ＰＴＰ移動
ＰＮＴ　：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）

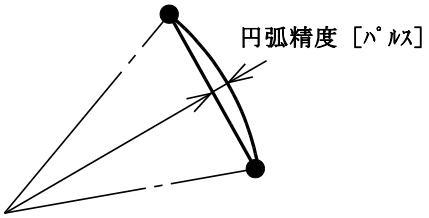
プロトタイプ

```
#include    "Plm2gcnv.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    "Plm2gcnv.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	最大ｽﾃｯﾌﾟ数	最大ﾗﾍﾞﾙ数	最大位置決ﾎﾟｲﾝﾄ数
PROG_SIZE_64K	６１９	６５	６１９
PROG_SIZE_32K	３０９	３３	３０９
PROG_SIZE_16K	１５４	１６	１５４

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    "Plm2gcnv.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　　１～２０００００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　　ＰＬＭＣ－ＭⅡの制御周期を設定します。

プロトタイプ

```
#include    "Plm2gcnv.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*　　ＰＬＭＣ－ＭⅡの制御周期をμｓｅｃで指定します。(500～4000)
ＰＬＭＣ－ＭⅡの制御周期はＲＯＭＳＷ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    "Plm2gcnv.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　　５０msec～５００msec

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。
（”Ｘ１．”や”Ｒ１．”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    "Plm2gcnv.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　*flag*　乗数フラグ
０：１　［パルス］
１：１０
２：１００
３：１０００
４：１００００
５：１０００００

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    "Plm2gcnv.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　*md*　円弧補間解析処理モード　0:プリ解､1:コントローラ内部解析

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    "Plm2gcnv.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    "Plm2gcnv.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 １～１０００００［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    "Plm2gcnv.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    "Plm2gcnv.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。