標準ＳＰＸ通信ライブラリリファレンスマニュアル

ＳＰＸＣＯＭ－ＮＴ説明書

Ｖｅｒ１．３
2004.08.09

TB00-0724-1

－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 ……3

６．インストール ……3

７．ライブラリ使用上の注意事項 ……3

８．関数使用例 ……4

８－１ ＭＳ－ＶＣ以外のＣ言語処理系での利用方法 ……4

９．関数仕様 ……6

９－１．通信 ……6

ＩｎｉｔＣｏｍｍＰｒｏｃ ……6
ＱｕｉｔＣｏｍｍＰｒｏｃ ……7
ＳｅｎｄＤａｔａ ……8
ＲｅｃｅｉｖｅＤａｔａ ……9
ＳｅｎｄＣｏｍｍａｎｄ ……10

９－２．タイマー ……11

ＳｅｔＴｉｍｅＯｕｔ ……11
ＣｈｅｃｋＴｉｍｅＯｕｔ ……11

１．形式

ＳＰＸＣＯＭ－ＮＴ

２．製品構成

ＳＸＣＯＭＮＴ　．ＤＬＬ	･･･････	ダイナミックリンクライブラリーファイル
ＳＸＣＯＭＮＴ　．ＬＩＢ	･･･････	インポートライブラリーファイル
ＳＸＣＯＭＮＴ　．Ｈ	･･･････	インクルードファイル(関数定義)
ＳＸＤＡＴＡ　．Ｈ	･･･････	インクルードファイル(構造体定義)
ＳＹＮＣＤＥＦ　．Ｈ	･･･････	インクルードファイル(マクロ定義)
ＳＸＣＯＭＡＰＩ．ＴＸＴ	･･･････	ビジュアルベーシック用データ型･関数定義

３．概要

本ライブラリ（標準ＳＰＸシリーズ対応通信ライブラリ）は、パソコンとＳＰＸコントローラ間で、ＲＳ－２３２Ｃ／ＰＣＩ／ＵＳＢを利用して、通信を行うための処理をライブラリ化したものです。
本ライブラリの機能は、Ｃ言語の関数形式で呼び出して、利用できます。

本ライブラリを利用して以下のような処理が行えます。
・データ送信
・データ受信
・動作指示
各データの詳細は送受信データ説明書を参照下さい。

４．関連資料

「ＳＰＸ－８０００ユーザーズマニュアル」	（ＴＢ００－０７３６）
「標準ＳＰＸ対応 セッティングＰＣマニュアル」	（ＴＢ００－０７３７）
「標準ＳＰＸ対応 ＲＯＭＳＷ設定ソフトマニュアル」	（ＴＢ００－０７３８）
「標準ＳＰＸ対応 送受信データ説明書」	（ＴＢ００－０７２３）
「標準ＳＰＸ対応 Ｔコード変換ライブラリリファレンスマニュアル」	（ＴＢ００－０７２５）
「標準ＳＰＸ対応 Ｇコード変換ライブラリリファレンスマニュアル」	（ＴＢ００－０７２６）

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　　・・・　Windows98/NT/2000/XP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。

※ ＰＣＩバス／ＵＳＢ接続で使用する場合は、デバイスドライバのインストールが必要です。
詳細はＳＰＸ－８０００ユーザーズマニュアルを参照下さい

７．ライブラリ使用上の注意事項

ＳＰＸシリーズ用通信ライブラリーを使用して、アプリケーションを作成するときは、以下の事を注意して下さい。

・ ＵＳＢ通信はＷｉｎｄｏｗｓ９８／２０００／ＸＰでのみ使用可能です。
ＷｉｎｄｏｗｓＮＴでは使用できません。

・ プログラムデータの作成には、弊社プログラム変換ライブラリが必要です。

・複数のアプリケーションで通信を利用する場合、以下に注意する必要があります。

＜ＲＳ２３２Ｃ通信＞
1)１つのＣＯＭポートに対しては、１つのアプリケーションしかアクセスすることはできません。
2)１台のＳＰＸに対して、最大２つのＣＯＭポートから同時にアクセスできます。
但し、この場合は別途２３２通信用基板／ケーブル（ＡＳ２３２／ＣＢ－Ｃ－０１２）をご購入いただく必要があります。

＜ＤＰＲＡＭ通信＞
1)１台のＳＰＸに対して、最大１００のアプリケーションからアクセスすることができます。
但し、アクセスするアプリケーションの数が増えると、通信速度が低下します。

＜ＵＳＢ通信＞
1)１つのＵＳＢに対しては、１つのアプリケーションしかアクセスすることはできません。

＜共通＞
1)１つのアプリケーションから最大４つの通信を行うことができます。
例1) 通信1:COM1 －SPX1、通信2:COM2－SPX2、通信3:COM3－SPX3、通信4:COM4－SPX4
例2) 通信1:DPRAM－SPX1、通信2:COM1－SPX1、通信3:USB －SPX2、通信4:COM2－SPX2
2)複数のアプリケーションから、複数の通信形態を併用して１台のＳＰＸにアクセスすることが可能です。
例）アプリケーション１：ＤＰＲＡＭ　－ＳＰＸ１
アプリケーション２：ＲＳ２３２Ｃ－ＳＰＸ１
アプリケーション３：ＲＳ２３２Ｃ－ＳＰＸ１
アプリケーション４：ＵＳＢ　　－ＳＰＸ１
3)１台のＳＰＸに対して、複数の通信を利用してデータ書き込みや動作指示を行う場合、それぞれのデータ書き込みや動作指示が干渉することがありますので、運用で干渉を回避する必要があります。

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　InitCommProc(PSXDEF *psxdef,* int **phCom*)の場合
関数名　・・・・・　_InitCommProc@8
これらの定義がSxcomnt.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、Sxcomnt.h内で定義しています。
Sxcomnt.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_SXDLL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、通信ライブラリの関数取得例を示します。

```
#define  LOADLIBRARY_SXDLL          // テクノ製ライブラリ明示的リンク指定
#include "Sxcomnt.h"                // 通信ライブラリ宣言

// グローバルデータ定義
HINSTANCE       hComInst;               /* 通信ライブラリハンドル                */
INITPROC        *InitCommProc;          /* 通信初期化関数へのﾎﾟｲﾝﾀ               */
QUITPROC        *QuitCommProc;          /* 通信終了処理関数へのﾎﾟｲﾝﾀ             */

// 通信ライブラリ関数ロード処理
int LoadTcdcnv(void)
{
    // ＤＬＬのロード
    if(NULL == (hComInst = LoadLibrary("SXCOMNT.DLL"))){
          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
    InitCommProc = (INITPROC *)GetProcAddress(hComInst, FNINITPROC);
    QuitCommProc = (QUITPROC *)GetProcAddress(hComInst, FNQUITPROC);

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

    return 1;
}
```

９．関数仕様

９－１．通信

ＩｎｉｔＣｏｍｍＰｒｏｃ

機能

通信インタフェース（ＤＰＲＡＭまたはＲＳ２３２Ｃ）の初期化及び、通信処理の組み込みを実行します。

プロトタイプ

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

int WINAPI InitCommProc(PSXDEF psxdef, int *phCom);
```

PSXDEF　　*psxdef*　通信定義構造体

通信処理の各種定義を行います。

```
typedef struct {
    int         fComType;       // 通信種別ﾌﾗｸﾞ
    int         nComPort;       // RS232C通信ﾎﾟｰﾄ番号
    int         nLptPort;       // ﾊﾟﾗﾚﾙ通信ﾎﾟｰﾄ
    int         nDpramId;       // DPRAM通信ID
    int         fShare;         // 共有フラグ
    int         fSxType;        // SXｺﾝﾄﾛｰﾗ種別
    int         nSize;          // SXｺﾝﾄﾛｰﾗ定義構造体サイズ
    long        fLogging;       // 通信ﾛｷﾞﾝｸﾞﾌﾗｸﾞ
    char        *pLogFile;      // 通信ﾛｷﾞﾝｸﾞﾌｧｲﾙ名
    char        Reserved[28];   // 予約
} SXDEF, *PSXDEF;
```

各変数の内容は以下の通りです。

fComType　通信種別フラグ

COMRS232C	(0)	：ＲＳ２３２Ｃ通信
COMPARALLEL	(1)	：予約
COMDPRAM	(2)	：ＤＰＲＡＭ通信
COMUSB	(3)	：ＵＳＢ通信

nComPort　ＲＳ２３２Ｃ通信ポート番号　　１～４

nLptPort　パラレル通信ポート　　１～４
ＳＰＸでは使用しません。１を指定して下さい。

nDpramId　ＤＰＲＡＭ／ＵＳＢ通信用ボードＩＤ　　０～４

fShare　共有フラグ　　０～１
１を指定して下さい。

fSxType　ＳＸコントローラ種別
ＳＰＸでは使用しません。
TYPE_SX8000T(0x10)を指定して下さい。

nSize　ＳＸコントローラ定義構造体サイズ
常に本構造体のサイズを設定してください。

fLogging　通信ロギングフラグ
内容は以下の通りです。
上位の未定義ビットは全て０にして下さい。

pLogFile　通信ロギングファイル名
ＳＰＸとの通信のログをとる場合にログファイル名を指定します。
ログをとる場合、実行ファイルと同じディレクトリに以下のファイルが作成されます。
１．「通信ﾛｷﾞﾝｸﾞﾌｧｲﾙ名」で指定したﾌｧｲﾙ
２．「通信ﾛｷﾞﾝｸﾞﾌｧｲﾙ名」で指定したﾌｧｲﾙのﾍﾞｰｽﾌｧｲﾙ名の最後に1～5の数字を付加したﾌｧｲﾙ
３．SPXxoooo.tmp
　　RS232C通信の場合･･･Com□←COMﾎﾟｰﾄ番号
　　DPRAM通信の場合 ･･･Dpr□←ﾎﾞｰﾄﾞID番号
　　USB通信の場合　 ･･･Usb□←ﾎﾞｰﾄﾞID番号
ログファイル（１のファイル）が５１２Ｋバイトをこえると現ログファイルはリネームされて、履歴ファイルとなります。その後、新しいログファイルを作成してロギングを継続します。
履歴ファイル名は、指定されたログファイル名のベース名に履歴番号として1～5を付加した名前です。(最大５世代)
例）ログファイル名として”TMP.LOG”を指定すると、履歴ファイルとしてTMP1.LOG～TMP5.LOGが作成されます。

int　*phCom　通信ハンドル
初期化が正常に終了した場合に、ハンドルを格納するデータのポインタを指定します。

戻り値

通信処理実行ステータスとして以下の値を返します。

E$OK	：	正常終了
E$EMPTYHANDLE	：	ハンドル取得不可
E$PARAM	：	パラメータ設定異常
E$ERR	：	初期化処理異常終了

正常に初期化が終了した場合、接続されたＳＰＸコントローラを識別するためのハンドルを *phCom* の指し示すアドレスへ格納します。

ＱｕｉｔＣｏｍｍＰｒｏｃ

機能

関数ＩｎｉｔＣｏｍｍＰｒｏｃの実行によって組み込まれた、通信処理を切り放し、組み込み前の状態に戻します。

プロトタイプ

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

int WINAPI QuitCommProc(int hCom);
```

int　*hCom*　通信ハンドル
通信初期化処理で取得したハンドルを指定します。

戻り値

ステータスとして以下の値を返します。

E$OK	：	正常終了
E$NOHANDLE	：	無効ハンドル
E$ERR	：	終了処理異常終了

ＳｅｎｄＤａｔａ

機能

ＳＰＸコントローラに対して、各種データ設定を行います。

プロトタイプ

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

int WINAPI SendData(int hCom, int type, short prm, DWORD size, LPVOID data);
```

int　　*hCom*　　通信ハンドル
　通信初期化処理で取得したハンドルを指定します。

int　　*type*　　データタイプ
　DAT_PARAMETER等、送信するデータのタイプを指定します。

short　　*prm*　　付加パラメータ
　データタイプによって、以下のデータを指定して下さい。
　以下に示されていないデータタイプについては０を指定して下さい。

・DAT_PROGRAM ：プログラム番号　　１～１２
・DAT_TASKPROG：タスク番号　　０～４
・DAT_DNCDATA ：先頭／継続フラグ　０：先頭時、１：継続時

DWORD　　*size*　　送信データサイズ
　０～６５５３５バイト

LPVOID　　*data*　　送信データ格納バッファへのポインタ

本関数のデータタイプに指定できるデータについては、「標準SPX-8000対応送受信データ説明書」の"データ送受信機能"を参照して下さい。

戻り値

通信処理実行ステータスとして以下の値を返します。

E$OK	：	正常終了
E$DEVNRDY	：	デバイス未初期化
E$PARAM	：	通信パラメータ設定異常
E$TIME	：	タイムアウト発生
E$RTRY	：	リトライオーバー発生
E$MLTRTRY	：	多重リトライ発生
E$HARDER	：	通信ハードウェアエラー
E$PROTECT	：	送信データ書込不可
E$SEQ	：	通信データフォーマットエラー
E$PRGTERM	：	プログラム書込中断
E$PRGBUFF	：	プログラムバッファオーバーフロー
E$NOHANDLE	：	無効ハンドル

ＲｅｃｅｉｖｅＤａｔａ

機能

ＳＰＸコントローラ内部の、各種データ読出を行います。

プロトタイプ

```
#include    "sxcomnt.h"        関数宣言に必要なインクルードファイル
int WINAPI ReceiveData(int hCom, int type, LPDWORD size, LPVOID data);
```

int	*hCom*	通信ハンドル 通信初期化処理で取得したハンドルを指定します。
int	*type*	データタイプ DAT_PARAMETER等、受信するデータのタイプを指定します。
short	*prm*	付加パラメータ データタイプによって、以下のデータを指定して下さい。 以下に示されていないデータタイプについては０を指定して下さい。 ・DAT_PROGRAM　：プログラム番号　１～１２ ・DAT_TASKPROG　：タスク番号　　０～４
LPDWORD	*size*	受信データサイズ格納変数へのポインタ ログデータ読み出し時は、１度に読み出すデータ数をセットします。
LPVOID	*data*	受信データ格納バッファへのポインタ

本関数のデータタイプに指定できるデータについては、「標準SPX-8000対応送受信データ説明書」の"データ送受信機能"を参照して下さい。

戻り値

通信処理実行ステータスとして以下の値を返します。

E$OK	：	正常終了
E$DEVNRDY	：	デバイス未初期化
E$PARAM	：	通信パラメータ設定異常
E$TIME	：	タイムアウト発生
E$RTRY	：	リトライオーバー発生
E$MLTRTRY	：	多重リトライ発生
E$HARDER	：	通信ハードウェアエラー
E$NEXIST	：	要求データが存在しない
E$SEQ	：	通信データフォーマットエラー
E$NEXIST	：	要求データが存在しない
E$NOHANDLE	：	無効ハンドル

ＳｅｎｄＣｏｍｍａｎｄ

機能

ＳＰＸコントローラに対して、各種動作指示を行います。

プロトタイプ

```
#include    "sxcomnt.h"        関数宣言に必要なインクルードファイル
int WINAPL SendCommand(int hCom, int cmnd, LPVOID data);
```

int	*hCom*	通信ハンドル 通信初期化処理で取得したハンドルを指定します。
int	*cmnd*	動作指示コード REQ_RESET等、動作指示のタイプを指定します。
LPVOID	*data*	動作パラメータ格納バッファへのポインタ

本関数の動作指示コードに指定できるデータについては、「標準SPX-8000対応送受信データ説明書」の"動作コマンド付加データ詳細"を参照して下さい。

戻り値

通信処理実行ステータスとして以下の値を返します。

E$OK	：	正常終了
E$DEVNRDY	：	デバイス未初期化
E$PARAM	：	通信パラメータ設定異常
E$TIME	：	タイムアウト発生
E$RTRY	：	リトライオーバー発生
E$MLTRTRY	：	多重リトライ発生
E$HARDER	：	通信ハードウェアエラー
E$SEQ	：	通信データフォーマットエラー
E$CMDNOT	：	コマンド実行不可
E$NOHANDLE	：	無効ハンドル

９－２．タイマー

Ｗｉｎｄｏｗｓが持つ、１ｍｓｅｃ間隔のインターバルタイマーをもとに、タイマーによる処理のディレイや同期、またタイムアウト処理などが可能となります。
「タイマーチェック起点関数」及び「タイムアウトチェック関数」によりアプリケーションにて通信処理以外の目的で、タイマー処理を行うことができます。

ＳｅｔＴｉｍｅＯｕｔ

機能

タイムアウトチェックの起点設定を行います。

プロトタイプ

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

void WINAPI  SetTimeOut(LPDWORD  origin);

LPDWORD         origin     タイムアウトチェック起点時間格納ポインタ
```

ＣｈｅｃｋＴｉｍｅＯｕｔ

機能

タイムアウトの判定を行います。

プロトタイプ

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

BOOl WINAPI  CheckeTimeOut(DWORD  origin,  DWORD  limit);

DWORD         origin       関数SetTimeOut()で取得したタイムアウトチェック起点時間
DWORD         limit        タイムリミット値(単位：１ｍｓｅｃ)
```

戻り値

```
      ＝  ＦＡＬＳＥ：タイムアウトしていない
      ＝  ＴＲＵＥ  ：タイムアウトしている
```

例

```
#include  <conio.h>
#include  "stcomnt.h"

main()
{
int    ch;
DWORD  StartTime;

    ch＝getch();                          /*キー入力                */
    SetTimeOut(&StartTime);               /*タイマーチェック起点設定*/
    While(!CheckTimeOut(StartTime,20L));/*１．１秒待ち              */
    putch(ch);                            /*入力キーコード表示      */
}
```

このプログラムは、キー入力があってから１．１秒後に、入力キーを表示します。