

関数リファレンス for CANa31.dll

Date	Auth	Notice
2021/01/13	Y. OGAWA	1st release
2021/03/03	Y. OGAWA	Update
2021/03/04	Y. OGAWA	Update
2021/05/14	Y. OGAWA	Update

目次

目次	1
注意点	2
環境	3
ファイル構成	3
基本的な利用方法	5
CANa31dll.cpp/h の利用方法	6
構造体	8
関数	10
InitInstance	10
ExitInstance	10
GetInterfaceCount	11
OpenInterface	11
CloseInterface	11
GetCurrentInterface	12
IsOpenInterface	12
SetOpenTimeout	13
SetSendTimeout	13
SetRecvTimeout	13
SetTargetID	14
GetTargetID	14
SetHostID	14
GetHostID	14
SetBaudrate	15
GetBaudrate	15
GetTm	15
abh3_can_init	16
abh3_can_cmdAY	17
abh3_can_cmdBX	17
abh3_can_cmd	17
abh3_can_inSet	18
abh3_can_inBitSet	18
abh3_can_reqPulse	19
abh3_can_reqBRD	19
cnvVel2CAN	20
cnvCAN2Vel	20
cnvCur2CAN	20
cnvCAN2Cur	20
cnvCAN2Load	21
cnvCAN2Analog	21
cnvCAN2Volt	21
値の単位	22

注意点

- ・ 本 DLL プロジェクトはソースコードを含んだ Visual Studio 用のプロジェクトとして提供されます
利用する Visual Studio は、バージョン 2015 又はそれ以降を想定しています
- ・ 本 DLL の利用には、以下の知識がある事が前提となります
DLL を Win32 プログラムから利用する為の知識
- ・ 本 DLL から HMS 製の特定 CAN インターフェースが利用可能ですが、他社の CAN インターフェースは
利用不可です。又、HMS 社の CAN インターフェースによっては、動作環境でドライバのインストールが必要と
なる場合があります
- ・ 高速に CAN 通信を行いたい場合は、HMS 社の USB-to-CAN V2 を御利用下さい。
同社の simplyCAN はスレッドセーフなドライバでは無い為、動作速度が落ちます。
- ・ 本 DLL は、32bit アプリケーション用の DLL として設計されています

環境

本 DLL の作成環境と想定利用環境は以下の通りです

要素	作成環境	想定利用環境
OS	Windows10 pro version 2004	Windows10 version 2004 又はそれ以降
CPU	Intel i7-3930K	Intel 系 CPU
コンパイラ	Microsoft Visual Studio 2015 pro	Microsoft Visual Studio 2015 又はそれ以降
DLL 利用先		32bit アプリケーション ・ MFC アプリケーション (32bit) ・ Win32 アプリケーション (32bit)

ファイル構成

本 DLL は以下のファイルで構成されます

ファイル名	内容
ABH3.cpp ABH3.h	ABH3 ドライバ固有機能 C++クラス
Can1939.h	CAN J1939 仕様のデータ作成用 C++クラス
CANa31.cpp CANa31.h	本 DLL でエクスポートされる関数が格納された C ソースコード ユーザーが利用する関数は、このコード内の関数が出入口となります
CANa31.def	本 DLL でエクスポートされる関数を定義したファイル 本 DLL では、このファイルをプロジェクト設定で明示的に指定してあります
CANa31.rc	DLL に内包されるリソース定義 バージョン情報等が含まれます
CANa31dll.cpp CANa31dll.h	本 DLL をユーザー側でダイナミックロードする場合に、関数を楽に扱う為のコード。 ユーザー側の上位アプリケーションを MFC で作成する場合に利用可能。 本 DLL 構築時に本コードは利用されません。
CanIF.cpp CanIF.h	本 DLL で扱うインターフェースの制御を行う C++クラス 利用可能な CAN インターフェースは、本クラスから継承して実装します
dllmain.cpp	DLL エントリ ユーザー側でアタッチ/デタッチ時に処理が必要な場合は、本ソースコードを改造して 利用します
IxxatSimple.cpp IxxatSimple.h	HMS 社製、simplyCAN インターフェースの制御クラス
IxxatV2.cpp IxxatV2.h	HMS 社製、USB-to-CAN v2 インターフェースの制御クラス
typedef.h	本 DLL でユーザーが使用する構造体の定義
resource.h stdafx.cpp stdafx.h targetvar.h	Visual Studio 利用時に自動作成されるファイル。 必要があればユーザー側で変更して下さい
readme.txt	本プロジェクトの履歴
CANa31.sln CANa31.vcproj	Visual Studio 用のソリューション/プロジェクトファイル。 本プロジェクトは、Visual Studio 2015 で構築しています。

インターフェースの利用準備

本 DLL では以下 2 種類のインターフェースに対応しています。

メーカー	HMS	
名称	USB-to-CAN V2	
	このインターフェースは、利用するだけでもインストール作業が必要です。	
準備手順	No.	手順
	1	インターネットブラウザで HMS 社のサイト（以下 URL）を開きます。 https://www.ixxat.com/
	2	USB-T0-CAN v2 のプロダクトから、以下のファイルをダウンロードします。 vci-v4-windows-10-8-7.zip 注意 プロダクト内の Download を選んでもファイルが表示されない為、 プロダクト画面の右の方にあるリンクからダウンロードします。
	3	取得したファイルを解凍して以下のファイルを取り出し、実行してインストールします。 ixxat VCI Setup 4.0.939.0.exe （ファイル名内の数字はバージョンの為、上記と多少異なる場合があります） 本 DLL の再ビルドを行う場合、インストール先に必要なファイルが格納されています。 Visual Studio のプロジェクト設定は、ixxatV2.cpp の先頭にある記述を確認して下さい。

メーカー	HMS	
名称	simplyCAN	
	このインターフェースは、利用するだけなら何も作業は不要ですが、本 DLL の再ビルド等を行う場合には、準備が必要です。	
準備手順	No.	手順
	1	インターネットブラウザで HMS 社のサイト（以下 URL）を開きます。 https://www.ixxat.com/
	2	simplyCAN のプロダクトから、以下のファイルをダウンロードします。 simplycan-driver-windows.zip 注意 プロダクト内の Download を選んでもファイルが表示されない為、 プロダクト画面の右の方にあるリンクからダウンロードします。
	3	取得したファイルを解凍します。 本 DLL の再ビルドを行う場合は、解凍先に必要なファイルが格納されています。 Visual Studio のプロジェクト設定は、ixxatSimple.cpp の先頭にある記述を確認して下さい。

基本的な利用方法

本 DLL の利用想定アプリケーションと利用方法は、以下となります

No.	アプリケーション種類	利用方法
1	Win32 アプリケーション	<p>本プロジェクトの生成物（DLL/LIB）をユーザー側のアプリケーションでスタティック又はダイナミックリンクで御利用下さい。</p> <p>但し、Visual Studio 2015 以外のコンパイラを利用される場合は、本 DLL プロジェクトをお客様の環境で再ビルドしてから御利用下さい。</p> <p>DLL の動的ロードを行う場合、プロジェクト内の CanA31dll.cpp/h を利用する事で、本 DLL を楽に扱う事が可能です。 （「CANa31dll.cpp/h の利用方法」の項を参照）</p> <p>処理の流れ等は、サンプルアプリケーションを御確認下さい。</p>
2	MFC アプリケーション	<p>本プロジェクトの生成物（DLL/LIB）をユーザー側のアプリケーションでスタティック又はダイナミックリンクで御利用下さい。</p> <p>但し、Visual Studio 2015 以外のコンパイラを利用される場合は、本 DLL プロジェクトをお客様の環境で再ビルドしてから御利用下さい。</p> <p>DLL の動的ロードを行う場合、プロジェクト内の CanA31dll.cpp/h を利用する事で、本 DLL を楽に扱う事が可能です。 （「CANa31dll.cpp/h の利用方法」の項を参照）</p> <p>処理の流れ等は、サンプルアプリケーションを御確認下さい。</p> <p>注意点 64bit アプリケーションからの利用は想定していません。 32bit アプリケーションから御利用下さい。</p>

CANa31dll.cpp/h の利用方法

本ファイルは DLL の関数を簡単に扱う為に用意されています。

ファイルをユーザーアプリケーションのプロジェクトにコピーして利用します。

コード例

```
#include "CANa31dll.h"
static CAN_FUNCLIST g_func;

int test()
{
    //DLL の読み込み
    HANDLE hDLL = LoadLibrary("CANa31.dll");
    if(hDLL == NULL)
        return(-1); //DLL 読み込みエラー

    //DLL に含まれる関数の取得 (CANa31dll.cpp 内に関数実体)
    GetFunctions(hDLL, &g_func);

    //DLL 関数の使用例
    g_func.InitInstance(0); //初期化関数の呼び出し

    g_func.<各種関数> を使用して処理を行う

    g_func.ExitInstance(); //開放前の呼び出し

    //DLL 開放
    FreeLibrary(hDLL);

    return(0);
}
```

サンプルアプリケーション

本 DLL プロジェクトには以下のサンプルが付属します

要素	値								
プロジェクト名	CanA31test								
アプリケーション種類	Win32 コンソールアプリケーション								
処理の流れ	<ul style="list-style-type: none"> ・ DLL (CanA31.dll) を読み込みます ・ 関数を楽に扱う為、CanA31dll.cpp/h を利用します (構造体に関数のエントリを入れます) ・ 初期化の為、InitInstance 関数を呼びます ・ 接続されているインターフェース数を確認します ・ 通信速度 (ボーレート) を設定します ・ 通信タイムアウト時間を設定します ・ 通信元 (PC) と通信先 (ABH3) の ID を設定します ・ インターフェースを開きます ・ 指令を初期化 (abh3_can_init 関数) します <p style="color: red;">ソースコードではここに処理がコメント化された状態で記述されています</p> <ul style="list-style-type: none"> ・ インタフェースを閉じます ・ 終了処理の為、ExitInstance 可数を呼びます ・ DLL を開放します 								
備考	<p>処理の流れを追うには、CanA31test.cpp を見て下さい。</p> <p>DLL の関数呼び出しに関しては、ソースコード上では最低限だけ残し、残りをコメント化してあります。</p>								
注意点	<p style="color: red;">リンクしたアプリケーションの実行には、以下の要素が必要です</p> <table border="1"> <thead> <tr> <th>ファイル名</th><th>説明</th></tr> </thead> <tbody> <tr> <td>CanA31.dll</td><td>本 DLL プロジェクト出力</td></tr> <tr> <td>simplyCAN.dll simplyCAN-64.dll</td><td>HMS 社 simplyCAN 利用時に必要</td></tr> <tr> <td>USB-to-CAN V2 のドライバ</td><td>HMS 社 USB-to-CAN V2 利用時に必要 事前にインストールが必要</td></tr> </tbody> </table>	ファイル名	説明	CanA31.dll	本 DLL プロジェクト出力	simplyCAN.dll simplyCAN-64.dll	HMS 社 simplyCAN 利用時に必要	USB-to-CAN V2 のドライバ	HMS 社 USB-to-CAN V2 利用時に必要 事前にインストールが必要
ファイル名	説明								
CanA31.dll	本 DLL プロジェクト出力								
simplyCAN.dll simplyCAN-64.dll	HMS 社 simplyCAN 利用時に必要								
USB-to-CAN V2 のドライバ	HMS 社 USB-to-CAN V2 利用時に必要 事前にインストールが必要								

構造体

通信の結果取得に構造体を使用します。

各関数を利用した時、どの要素として格納されるのかは、その関数説明内に記載があります。

以下定義内容

```
typedef struct _CANABH3_RESULT
{
    uint32_t nID;                //受信データの CAN-ID
    struct _DPOS
    {
        int16_t      nOrderAY;    //送信データの A/Y 指令値
        int16_t      nOrderBX;    //送信データの B/X 指令値
        int32_t      nInputBit;   //送信データの入力(bit 対応)値
    } DPOS;

    union
    {
        uint8_t      raw[8];      //一括アクセス用
        struct _DPOR
        {
            int16_t      nBackSpeedA;    //A 速度帰還
            int16_t      nBackSpeedB;    //B 速度帰還
            int16_t      nBackSpeedY;    //Y 速度帰還
            int16_t      nBackSpeedX;    //X 速度帰還
        } DPOR;
        struct _DP1R
        {
            int32_t      nInPulseA;    //A パルス積算値
            int32_t      nInPulseB;    //B パルス積算値
        } DP1R;
        struct _BR0
        {
            uint32_t      nErrorBit;    //異常フラグ
            uint32_t      nWarnBit;    //警告フラグ
        } BR0;
        struct _BR1
        {
            uint32_t      nCtrlBit;    //制御フラグ
            uint32_t      nIOflag;    //IO フラグ
        } BR1;
        struct _BR2
        {
            int16_t      nOrderSpeedAY;    //A/Y 速度指令
            int16_t      nOrderSpeedBX;    //B/X 速度指令
            int16_t      nBackSpeedAY;    //A/Y 速度帰還
            int16_t      nBackSpeedBX;    //B/X 速度帰還
        } BR2;
        struct _BR3
        {
            int16_t      nOrderCurrentAY;    //A/Y 電流指令
            int16_t      nOrderCurrentBX;    //B/X 電流指令
            int16_t      nLoadA;            //A 負荷率
        }
    }
};
```

```

        int16_t      nLoadB;          //B 負荷率
    } BR3;
    struct _BR4
    {
        int32_t      nInPulseA;       //A パルス積算値
        int32_t      nInPulseB;       //B パルス積算値
    } BR4;
    struct _BR5
    {
        int16_t      nAnalog0;         //アナログ入力 0
        int16_t      nAnalog1;         //アナログ入力 1
        int16_t      nPowerMain;       //主電源電圧
        int16_t      nPowerCtrl;       //制御電源電圧
    } BR5;
    struct _BR6
    {
        float         nMonitor0;       //モニタ 0 データ
        float         nMonitor1;       //モニタ 1 データ
    } BR6;
    struct _BUF
    {
        uint8_t       nData[8];        //8 バイトデータ
    } BUF;
} u;
} CANABH3_RESULT, *pCANABH3_RESULT;

```

関数

InitInstance

概要	インターフェースの利用開始										
詳細	使用したいインターフェースを指定して、利用を開始します										
構文	CANA31API void InitInstance(int32_t nIFnum)										
パラメータ	<table border="1"> <thead> <tr> <th>変数名</th><th>内容</th></tr> </thead> <tbody> <tr> <td>nIFnum</td><td> <div>使用したいインターフェースを指定します</div> <table border="1"> <thead> <tr> <th>値</th><th>インターフェース</th></tr> </thead> <tbody> <tr> <td>0</td><td>USB-to-CAN v2</td></tr> <tr> <td>1</td><td>simplyCAN</td></tr> </tbody> </table> </td></tr> </tbody> </table>	変数名	内容	nIFnum	<div>使用したいインターフェースを指定します</div> <table border="1"> <thead> <tr> <th>値</th><th>インターフェース</th></tr> </thead> <tbody> <tr> <td>0</td><td>USB-to-CAN v2</td></tr> <tr> <td>1</td><td>simplyCAN</td></tr> </tbody> </table>	値	インターフェース	0	USB-to-CAN v2	1	simplyCAN
変数名	内容										
nIFnum	<div>使用したいインターフェースを指定します</div> <table border="1"> <thead> <tr> <th>値</th><th>インターフェース</th></tr> </thead> <tbody> <tr> <td>0</td><td>USB-to-CAN v2</td></tr> <tr> <td>1</td><td>simplyCAN</td></tr> </tbody> </table>	値	インターフェース	0	USB-to-CAN v2	1	simplyCAN				
値	インターフェース										
0	USB-to-CAN v2										
1	simplyCAN										
戻り値	無し										
注意点等	<p>この関数を呼び出す時点では、使用したいインターフェースが接続されていなくても問題ありません。(OpenInterface を呼び出す時点で接続されていれば良い)</p> <p>一度開いたインターフェースを切り替えたい場合は、以下手順で行ってください。</p> <table border="1"> <thead> <tr> <th>No.</th><th>手順</th></tr> </thead> <tbody> <tr> <td>1</td><td>CloseInterface を呼び出す</td></tr> <tr> <td>2</td><td>ExitInstance を呼び出す</td></tr> <tr> <td>3</td><td>InitInstance で新しいインターフェースを指定する</td></tr> </tbody> </table>	No.	手順	1	CloseInterface を呼び出す	2	ExitInstance を呼び出す	3	InitInstance で新しいインターフェースを指定する		
No.	手順										
1	CloseInterface を呼び出す										
2	ExitInstance を呼び出す										
3	InitInstance で新しいインターフェースを指定する										

ExitInstance

概要	インターフェースの利用終了
詳細	インターフェースの利用を終了し、DLL を開放可能な状態にします
構文	CANA31API void ExitInstance()
パラメータ	無し
戻り値	無し
注意点等	インターフェースを開いている場合は、先に CloseInterface を呼び出して下さい

GetInterfaceCount

概要	使用可能な CAN インターフェース数を取得	
詳細	InitInstance で指定した「使用したいインターフェース」に対して、現時点で利用可能な本数(PC に接続されているデバイス数)を取得します	
構文	CANA31API int32_t GetInterfaceCount()	
パラメータ	無し	
戻り値	指定済みのインターフェースにより、値が異なります	
	インターフェース	戻り値
	USB-to-CAN v2	PC に接続されている本数が戻ります
	simplyCAN	1 本以上接続されている場合は、1 1 本も接続されていない場合は、0 が戻ります
注意点等	どちらのインターフェースも、2 本以上接続する事は非推奨です。	

OpenInterface

概要	指定インターフェースを開く。	
詳細	インターフェースを指定して、その回線を開きます	
構文	CANA31API int32_t OpenInterface(int32_t nDeviceNum)	
パラメータ	変数名	内容
	nDeviceNum	開く対象のインターフェース番号を指定します 指定する値は、使用インターフェース毎に異なりますので注意が必要です
		インターフェース
		値
戻り値	0	正常終了
	上記以外	異常終了
注意点等	インターフェースに simplyCAN を使用する場合、本関数の実行前にどこの COM ポートに接続されているか、デバイスマネージャ等を利用して調べる必要が有ります。	
	インターフェースを既に開いている時に本関数を呼び出した場合、開いているインターフェースを閉じてから、新しい設定で開きなおします。	

CloseInterface

概要	開いたインターフェースを閉じる	
詳細	OpenInterface で開いたインターフェースを閉じます	
構文	CANA31API void CloseInterface()	
パラメータ	無し	
戻り値	無し	
注意点等	インターフェースを開いていない場合は、何もしません。	

GetCurrentInterface

概要	現在開いているインターフェース番号を取得	
詳細	OpenInterface を実行した時に指定したインターフェース番号を取得します	
構文	CANA31API int32_t GetCurrentInterface()	
パラメータ	無し	
戻り値	戻り値	内容
	0 以上	OpenInterface 関数に指定した値が返ります
	上記以外	インターフェースを開いていません
注意点等		

IsOpenInterface

概要	現在インターフェースを開いているか？	
詳細	現在、インターフェースを開いた状態かどうか判断します	
構文	CANA31API int32_t IsOpenInterface()	
パラメータ	無し	
戻り値	戻り値	内容
	0 以外	インターフェースを開いています
	0	インターフェースを開いていません
注意点等		

SetOpenTimeout

概要	インターフェースを開くタイムアウト時間を設定	
詳細	OpenInterface を実行する時、回線が開く迄待つ時間を[ms]単位で指定します	
構文	CANA31API void SetOpenTimeout(uint32_t nTimeoutMS)	
パラメータ	変数名	内容
	nTimeoutMS	インターフェースを開く処理に許容する時間[ms] 推奨値は 3000
戻り値	無し	
注意点等	インターフェースに simplyCAN を使用している場合、本関数の指定は無視されます。 インターフェースに USB-to-CAN v2 を使用している場合は、OpenInterface を呼び出す前に必ず設定する必要が有ります。	

SetSendTimeout

概要	送信タイムアウト時間を設定	
詳細	インターフェースヘータ送信する場合の、許容時間を[ms]単位で指定します	
構文	CANA31API void SetSendTimeout(uint32_t nTimeoutMS)	
パラメータ	変数名	内容
	nTimeoutMS	送信処理に許容する時間[ms] 送信処理実行時、この設定値以上の時間が掛かった場合は、異常と判定されます 推奨値は 1000
戻り値	無し	
注意点等	インターフェースに simplyCAN を使用している場合、本関数の指定は無視されます。 インターフェースに USB-to-CAN v2 を使用している場合は、OpenInterface を呼び出す前に必ず設定する必要が有ります。	

SetRecvTimeout

概要	受信タイムアウト時間を設定	
詳細	インターフェースからデータ受信する場合の、許容時間を[ms]単位で指定します	
構文	CANA31API void SetRecvTimeout(uint32_t nTimeoutMS)	
パラメータ	変数名	内容
	nTimeoutMS	受信処理に許容する時間[ms] 受信処理実行時、この時間内に何も受信出来なかった場合は、異常と判定されます 推奨値は 1000 ですが、異常に対して早くリカバリする為には、小さい値を指定して下さい
戻り値	無し	
注意点等	本関数は、OpenInterface を呼び出す前に、必ず設定する必要が有ります。	

SetTargetID

概要	通信対象 ABH3 のアドレスを設定	
詳細	通信を行う場合に、通信相手となる ABH3 を指定します	
構文	CANA31API void SetTargetID(uint8_t nAdrs)	
パラメータ	変数名	内容
	nAdrs	通信対象 ABH3 のアドレスを指定します
戻り値	無し	
注意点等	本関数は、OpenInterface を呼び出す前に、必ず設定する必要が有ります。	

GetTargetID

概要	通信対象 ABH3 のアドレスを取得	
詳細	現在の通信相手となる ABH3 のアドレスを取得します	
構文	CANA31API uint8_t GetTargetID()	
パラメータ	無し	
戻り値	SetTargetID で指定された通信対象 ABH3 のアドレスが戻ります	
注意点等		

SetHostID

概要	通信ホストのアドレスを設定	
詳細	PC が使用する通信アドレスを指定します	
構文	CANA31API void SetHostID(uint8_t nAdrs)	
パラメータ	変数名	内容
	nAdrs	通信ホスト (PC) のアドレスを指定します
戻り値	無し	
注意点等	本関数は、OpenInterface を呼び出す前に、必ず設定する必要が有ります。	

GetHostID

概要	通信ホストのアドレスを取得	
詳細	現在、PC が使用する通信アドレスを取得します	
構文	CANA31API uint8_t GetHostID()	
パラメータ	無し	
戻り値	SetHostID で指定された通信ホストのアドレスが戻ります	
注意点等		

SetBaudrate

概要	通信速度を指定			
詳細	通信速度を指定します			
構文	CANA31API void SetBaudrate(uint32_t nBaudrateKbps)			
パラメータ	変数名	nBaudrateKbps	内容	
			HMS 社製インターフェース（USB-T0-CAN v2 及び simplyCAN）が利用可能な通信速度[Kbps]を以下から指定します	
			値	通信速度
			10	10[Kbps]
			20	20[Kbps]
			50	50[Kbps]
			100	100[Kbps]
			125	125[Kbps]
			250	250[Kbps]
			500	500[Kbps]
800	800[Kbps]			
1000	1000[Kbps]			
戻り値	無し			
注意点等	本関数は、OpenInterface を呼び出す前に、必ず設定する必要が有ります。 インターフェースを開いた後で通信速度を変更する場合は、インターフェースを一度閉じてから本関数で新しい速度を設定し、再度インターフェースを開き直して下さい。			

GetBaudrate

概要	設定した通信速度を取得
詳細	設定済みの通信速度を取得します
構文	CANA31API uint32_t GetBaudrate()
パラメータ	無し
戻り値	SetBaudrate で指定された通信速度[Kbps]が戻ります
注意点等	

GetTm

概要	時間を[ms]単位で取得
詳細	PC が起動した時間を 0 として、現在迄の時間を[ms]単位で取得します
構文	CANA31API uint32_t GetTm()
パラメータ	無し
戻り値	PC が起動してから現在迄の時間が[ms]単位で戻ります
注意点等	32bit が最大の為、49.7 日程度でオーバーフローして 0 に戻ります

abh3_can_init

概要	指令の初期化	
詳細	以下の要素を一括設定します	
	要素名	設定値
	A/Y 指令	0
	B/X 指令	0
	入力(bit 対応)	0
構文	CANA31API int32_t abh3_can_init()	
パラメータ	無し	
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等		

abh3_can_cmdAY

abh3_can_cmdBX

概要	指令の送信（軸別）	
詳細	指定を送信します	
構文	CANA31API int32_t abh3_can_cmdAY(int16_t cmd, pCANABH3_RESULT pPtr) CANA31API int32_t abh3_can_cmdBX(int16_t cmd, pCANABH3_RESULT pPtr)	
パラメータ	変数名	内容
	cmd	A/Y 又は B/X 指令値
	pPtr	通信結果を受け取る領域へのポインタ (pPtr->u. DPOR に格納される)
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等	指定値以外に必要な値が有る場合、過去の値を使用します	

abh3_can_cmd

概要	指令の送信（同時）	
詳細	A/Y 指令値と B/X 指令値を同時に送信します	
構文	CANA31API int32_t abh3_can_cmd(int16_t cmdAY, int16_t cmdBX, pCANABH3_RESULT pPtr)	
パラメータ	変数名	内容
	cmdAY	A/Y 指令値
	cmdBX	B/X 指令値
	pPtr	通信結果を受け取る領域へのポインタ (pPtr->u. DPOR に格納される)
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等	指定値以外に必要な値が有る場合、過去の値を使用します	

abh3_can_inSet

概要	入力の送信（一括）	
詳細	入力(bit 対応)の値をデータ値とマスク値で構築し、送信します 入力(bit 対応) = (入力(bit 対応) & ~mask) (data & mask)	
構文	CANA31API int32_t abh3_can_inSet(int32_t data, int32_t mask, pCANABH3_RESULT pPtr)	
パラメータ	変数名	内容
	data	データ値
	mask	マスク値
	pPtr	通信結果を受け取る領域へのポインタ (pPtr->u.DPOR に格納される)
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等		

abh3_can_inBitSet

概要	入力の送信（ビット）	
詳細	現在の入力(bit 対応)の特定ビットを操作し、送信します 入力(bit 対応) = 入力(bit 対応) & ~(1 << num) (data << num)	
構文	CANA31API int32_t abh3_can_inBitSet(int8_t num, int8_t data, pCANABH3_RESULT pPtr)	
パラメータ	変数名	内容
	num	ビット番号 (0~31)
	data	設定データ (0~1)
	pPtr	通信結果を受け取る領域へのポインタ (pPtr->u.DPOR に格納される)
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等	指定値以外に必要な値がある場合、過去の値を使用します	

abh3_can_reqPulse

概要	積算値のリクエスト（この関数は v1.1.1.7 で廃止されました）	
詳細	以下の要素を取得します	
	要素名	
	A パルス積算値	
	B パルス積算値	
構文	CANAPI_int32_t abh3_can_reqPulse(pCANABH3_RESULT pPtr)	
パラメータ	変数名	内容
	pPtr	通信結果を受け取る領域へのポインタ (pPtr->u.DP1R に格納される)
戻り値	戻り値	内容
	0	正常終了
	0 以外	異常終了時のエラーコード
注意点等	この関数は v1.1.1.7 で廃止されました	

abh3_can_reqBRD

概要	ブロードキャストパケットのリクエスト																
詳細	指定番号のブロードキャストパケットを送信し、指定番号に対する要素を取得します																
構文	CANA31API int32_t abh3_can_reqBRD(uint8_t num, pCANABH3_RESULT pPtr)																
パラメータ	<table> <tr> <th>変数名</th><th>内容</th></tr> <tr> <td>num</td><td>番号 (0x00~0xff)</td></tr> <tr> <td>pPtr</td><td>通信結果を受け取る領域へのポインタ 格納先は、「注意点等」を参照の事</td></tr> </table>	変数名	内容	num	番号 (0x00~0xff)	pPtr	通信結果を受け取る領域へのポインタ 格納先は、「注意点等」を参照の事										
変数名	内容																
num	番号 (0x00~0xff)																
pPtr	通信結果を受け取る領域へのポインタ 格納先は、「注意点等」を参照の事																
戻り値	<table> <tr> <th>戻り値</th><th>内容</th></tr> <tr> <td>0</td><td>正常終了</td></tr> <tr> <td>0 以外</td><td>異常終了時のエラーコード</td></tr> </table>	戻り値	内容	0	正常終了	0 以外	異常終了時のエラーコード										
戻り値	内容																
0	正常終了																
0 以外	異常終了時のエラーコード																
注意点等	<p>パラメータの num に対する通信結果を受け取る領域は以下の通り 内容に関しては、構造体の項を参照の事</p> <table> <tr> <th>num</th><th>格納先</th></tr> <tr> <td>0x28</td><td>pPtr->u.BR0</td></tr> <tr> <td>0x29</td><td>pPtr->u.BR1</td></tr> <tr> <td>0x2a</td><td>pPtr->u.BR2</td></tr> <tr> <td>0x2b</td><td>pPtr->u.BR3</td></tr> <tr> <td>0x2c</td><td>pPtr->u.BR4</td></tr> <tr> <td>0x2d</td><td>pPtr->u.BR5</td></tr> <tr> <td>0x2e</td><td>pPtr->u.BR6</td></tr> </table>	num	格納先	0x28	pPtr->u.BR0	0x29	pPtr->u.BR1	0x2a	pPtr->u.BR2	0x2b	pPtr->u.BR3	0x2c	pPtr->u.BR4	0x2d	pPtr->u.BR5	0x2e	pPtr->u.BR6
num	格納先																
0x28	pPtr->u.BR0																
0x29	pPtr->u.BR1																
0x2a	pPtr->u.BR2																
0x2b	pPtr->u.BR3																
0x2c	pPtr->u.BR4																
0x2d	pPtr->u.BR5																
0x2e	pPtr->u.BR6																

cnvVel2CAN

概要	速度を「ABH3 の速度」に変換	
詳細	ユーザーが扱う [min ⁻¹] の速度値を、「ABH3 の速度」に変換	
構文	CANA31API int16_t cnvVel2CAN(float vel)	
パラメータ	変数名	内容
	vel	変換元の速度 [min ⁻¹]
戻り値	変換された速度が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCAN2Vel

概要	「ABH3 の速度」を変換	
詳細	「ABH3 の速度」の値を、ユーザーが扱う速度 [min ⁻¹] に変換	
構文	CANA31API float cnvCAN2Vel(int16_t vel)	
パラメータ	変数名	内容
	vel	変換元の値
戻り値	変換された速度 [min ⁻¹] が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCur2CAN

概要	電流値を「ABH3 の電流値」に変換	
詳細	ユーザーが扱う電流値 [%] を、「ABH3 の電流値」に変換	
構文	CANA31API float cnvCur2CAN(float cur)	
パラメータ	変数名	内容
	cur	変換元の値
戻り値	変換された電流値 [%] が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCAN2Cur

概要	「ABH3 の電流値」を変換	
詳細	「ABH3 の電流値」の値を、ユーザーが扱う電流値 [%] に変換	
構文	CANA31API float cnvCAN2Cur(int16_t cur)	
パラメータ	変数名	内容
	cur	変換元の値
戻り値	変換された電流値が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCAN2Load

概要	「ABH3 の負荷率」を変換	
詳細	「ABH3 の負荷率」の値を、ユーザーが扱う負荷率[%]に変換	
構文	CANA31API float cnvCAN2Load(int16_t load)	
パラメータ	変数名	内容
	load	変換元の値
戻り値	変換された負荷率[%]が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCAN2Analog

概要	「ABH3 のアナログ入力」を変換	
詳細	「ABH3 のアナログ入力」の値を、ユーザーが扱うアナログ入力値[V]に変換	
構文	CANA31API float cnvCAN2Analog(int16_t analog)	
パラメータ	変数名	内容
	analog	変換元の値
戻り値	変換されたアナログ入力値[V]が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

cnvCAN2Volt

概要	「ABH3 の電源電圧値」を変換	
詳細	「ABH3 の電源電圧値」の値を、ユーザーが扱う電源電圧値[V]に変換	
構文	CANA31API float cnvCAN2Volt(int16_t volt)	
パラメータ	変数名	内容
	volt	変換元の値
戻り値	変換された電源電圧値[V]が戻ります	
注意点等	ユーザー側で扱う値と ABH3 で扱う値の関係は、「値の単位」の項を参照の事	

値の単位

ユーザー側で使用する単位と ABH3 側に指定する値の関係は以下の通り

要素	ユーザー側単位	ABH3 分解能	変換関数	
			ユーザー → ABH3	ABH3 → ユーザー
速度（指令・帰還）	[min]	0.2 [min]	cnvVel2CAN	cnvCAN2Vel
電流（指令・帰還）	[%]	0.01 [%]	cnvCur2CAN	cnvCAN2Cur
パルス積算値	[Pulse]	1 [Pulse]	—	—
負荷率	[%]	1 [%]	—	cnvCAN2Load
主電源／制御電源電圧	[V]	0.1 [V]	—	cnvCAN2VOLT
アナログ入力	[V]	0.01 [V]	—	cnvCAN2Analog
モニタデータ	単位無し	単位無し	—	—