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 |

# 目次

## 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<br>・MFC アプリケーション(32bit)<br>・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<br>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<br>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<br>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<br>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<br>バージョン情報等が含まれます |
| CANa31dll.cpp<br>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。<br>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。<br>本 DLL 構築時に本コードは利用されません。 |
| CanIF.cpp<br>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<br>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<br>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<br>IxxatSimple.h | HMS 社製、simplyCAN インターフェースの制御クラス |
| IxxatV2.cpp<br>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<br>stdafx.cpp<br>stdafx.h<br>targetvar.h | Visual Studio 利用時に自動作成されるファイル。<br>必要が有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<br>CANa31.vcproj | Visual Studio 用のソリューション/プロジェクトファイル。<br>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。<br>https://www.ixxat.com/ |
| | 2 | USB-TO-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。<br>vci-v4-windows-10-8-7.zip<br>注意<br>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、<br>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。<br>ixxat VCI Setup 4.0.939.0.exe<br>（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有ります）<br>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。<br>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<br>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。<br>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。<br>simplycan-driver-windows.zip<br>注意<br>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、<br>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。<br>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。.<br>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 | 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。<br><br>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。<br><br>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。<br>（「CANa31dll.cpp/h の利用方法」の項を参照）<br><br>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。 |
| 2 | MFC アプリケーション | 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。<br><br>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。<br><br>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。<br>（「CANa31dll.cpp/h の利用方法」の項を参照）<br><br>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。<br><br>注意点<br>64bit アプリケーションからの利用は想定していません。<br>32bit アプリケーションから御利用下さい。 |

## 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(nDLL == 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

<table>
<tr><th>要素</th><th>値</th></tr>
<tr><td>プロジェクト名</td><td>CanA31test</td></tr>
<tr><td>アプリケーション種類</td><td>Win32 コンソールアプリケーション</td></tr>
<tr><td>処理の流れ</td><td>・DLL(CanA31.dll)を読み込みます<br>・関数を楽に扱う為、CanA31dll.cpp/h を利用します<br>（構造体に関数のエントリを入れます）<br>・初期化の為、InitInstance 関数を呼びます<br>・接続されているインターフェース数を確認します<br>・通信速度（ボーレート）を設定します<br>・通信タイムアウト時間を設定します<br>・通信元(PC)と通信先(ABH3)の ID を設定します<br>・インターフェースを開きます<br>・指令を初期化(abh3_can_init 関数)します<br><br>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<br><br>・インタフェースを閉じます<br>・終了処理の為、ExitInstance 可数を呼びます<br>・DLL を開放します</td></tr>
<tr><td>備考</td><td>処理の流れを追うには、CanA31test.cpp を見て下さい。<br>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、<br>残りをコメント化してあります。</td></tr>
<tr><td>注意点</td><td>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
<table>
<tr><th>ファイル名</th><th>説明</th></tr>
<tr><td>CanA31.dll</td><td>本 DLL プロジェクト出力</td></tr>
<tr><td>simplyCAN.dll<br>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<br>事前にインストールが必要</td></tr>
</table></td></tr>
</table>

## 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

<table>
<tr><td>概要</td><td>インターフェースの利用開始</td></tr>
<tr><td>詳細</td><td>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</td></tr>
<tr><td>構文</td><td>CANA31API void InitInstance(int32_t nIFnum)</td></tr>
<tr><td>パラメータ</td><td>
<table>
<tr><th>変数名</th><th>内容</th></tr>
<tr><td>nIFnum</td><td>使用したいインターフェースを指定します
<table>
<tr><th>値</th><th>インターフェース</th></tr>
<tr><td>0</td><td>USB-to-CAN v2</td></tr>
<tr><td>1</td><td>simplyCAN</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>戻り値</td><td>無し</td></tr>
<tr><td>注意点等</td><td>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有りません。(OpenInterface を呼び出す時点で接続されていれば良い)<br><br>一度開いたインターフェースを切り替えたい場合は、以下手順で行って下さい。
<table>
<tr><th>No.</th><th>手順</th></tr>
<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>
</table>
</td></tr>
</table>

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

<table>
<tr><td>概要</td><td>使用可能な CAN インターフェース数を取得</td></tr>
<tr><td>詳細</td><td>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</td></tr>
<tr><td>構文</td><td>CANA31API int32_t GetInterfaceCount()</td></tr>
<tr><td>パラメータ</td><td>無し</td></tr>
<tr><td>戻り値</td><td>指定済みのインターフェースにより、値が異なります
<table>
<tr><th>インターフェース</th><th>戻り値</th></tr>
<tr><td>USB-to-CAN v2</td><td>PC に接続されている本数が戻ります</td></tr>
<tr><td>simplyCAN</td><td>1 本以上接続されている場合は、1<br>1 本も接続されていない場合は、0<br>が戻ります</td></tr>
</table></td></tr>
<tr><td>注意点等</td><td>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。</td></tr>
</table>

OpenInterface

<table>
<tr><td>概要</td><td>指定インターフェースを開く。</td></tr>
<tr><td>詳細</td><td>インターフェースを指定して、その回線を開きます</td></tr>
<tr><td>構文</td><td>CANA31API int32_t OpenInterface(int32_t nDeviceNum)</td></tr>
<tr><td>パラメータ</td><td>
<table>
<tr><th>変数名</th><th>内容</th></tr>
<tr><td>nDeviceNum</td><td>開く対象のインターフェース番号を指定します<br>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
<table>
<tr><th>インターフェース</th><th>値</th></tr>
<tr><td>USB-to-CAN v2</td><td>何本目か（開始：0）</td></tr>
<tr><td>simplyCAN</td><td>COM ポート番号（開始：1）</td></tr>
</table></td></tr>
</table></td></tr>
<tr><td>戻り値</td><td>
<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>上記以外</td><td>異常終了</td></tr>
</table></td></tr>
<tr><td>注意点等</td><td>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。<br><br>インターフェースを既に開いている時に本関数を呼びだした場合、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。</td></tr>
</table>

CloseInterface

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

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 | <table><tr><th>戻り値</th><th>内容</th></tr><tr><td>0 以上</td><td>OpenInterface 関数に指定した値が戻ります</td></tr><tr><td>上記以外</td><td>インターフェースを開いていません</td></tr></table> |
| 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 | <table><tr><th>戻り値</th><th>内容</th></tr><tr><td>0 以外</td><td>インターフェースを開いています</td></tr><tr><td>0</td><td>インターフェースを開いていません</td></tr></table> |
| 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: nTimeoutMS<br>内容: インターフェースを開く処理に許容する時間[ms]<br>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。<br>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 | インターフェースへデ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: nTimeoutMS<br>内容: 送信処理に許容する時間[ms]<br>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<br>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。<br>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: nTimeoutMS<br>内容: 受信処理に許容する時間[ms]<br>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<br>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: nAdrs / 内容: 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: nAdrs / 内容: 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

<table>
<tr><td>概要</td><td>通信速度を指定</td></tr>
<tr><td>詳細</td><td>通信速度を指定します</td></tr>
<tr><td>構文</td><td>CANA31API void SetBaudrate(uint32_t nBaudrateKbps)</td></tr>
<tr><td>パラメータ</td><td>
<table>
<tr><th>変数名</th><th>内容</th></tr>
<tr><td>nBaudrateKbps</td><td>HMS 社製インターフェス（USB-TO-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
<table>
<tr><th>値</th><th>通信速度</th></tr>
<tr><td>10</td><td>10[Kbps]</td></tr>
<tr><td>20</td><td>20[Kbps]</td></tr>
<tr><td>50</td><td>50[Kbps]</td></tr>
<tr><td>100</td><td>100[Kbps]</td></tr>
<tr><td>125</td><td>125[Kbps]</td></tr>
<tr><td>250</td><td>250[Kbps]</td></tr>
<tr><td>500</td><td>500[Kbps]</td></tr>
<tr><td>800</td><td>800[Kbps]</td></tr>
<tr><td>1000</td><td>1000[Kbps]</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>戻り値</td><td>無し</td></tr>
<tr><td>注意点等</td><td>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。<br>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<br>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。</td></tr>
</table>

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

<table>
<tr><td>概要</td><td>指令の初期化</td></tr>
<tr><td>詳細</td><td>以下の要素を一括設定します
<table>
<tr><th>要素名</th><th>設定値</th></tr>
<tr><td>A/Y 指令</td><td>0</td></tr>
<tr><td>B/X 指令</td><td>0</td></tr>
<tr><td>入力(bit 対応)</td><td>0</td></tr>
</table></td></tr>
<tr><td>構文</td><td>CANA31API int32_t abh3_can_init()</td></tr>
<tr><td>パラメータ</td><td>無し</td></tr>
<tr><td>戻り値</td><td>
<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></td></tr>
<tr><td>注意点等</td><td></td></tr>
</table>

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)<br>CANA31API int32_t abh3_can_cmdBX(int16_t cmd,pCANABH3_RESULT pPtr) |
| パラメータ | 変数名 / 内容<br>cmd / A/Y 又は B/X 指令値<br>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<br>0 / 正常終了<br>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<br>cmdAY / A/Y 指令値<br>cmdBX / B/X 指令値<br>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<br>0 / 正常終了<br>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

<table>
<tr><td>概要</td><td>入力の送信（一括）</td></tr>
<tr><td>詳細</td><td>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<br>入力(bit 対応) = (入力(bit 対応) & ~mask) | (data & mask)</td></tr>
<tr><td>構文</td><td>CANA31API int32_t abh3_can_inSet(int32_t data,int32_t mask,pCANABH3_RESULT pPtr)</td></tr>
<tr><td>パラメータ</td><td>
<table>
<tr><th>変数名</th><th>内容</th></tr>
<tr><td>data</td><td>データ値</td></tr>
<tr><td>mask</td><td>マスク値</td></tr>
<tr><td>pPtr</td><td>通信結果を受け取る領域へのポインタ<br>（pPtr->u.DPOR に格納される）</td></tr>
</table>
</td></tr>
<tr><td>戻り値</td><td>
<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>
</td></tr>
<tr><td>注意点等</td><td></td></tr>
</table>

abh3_can_inBitSet

<table>
<tr><td>概要</td><td>入力の送信（ビット）</td></tr>
<tr><td>詳細</td><td>現在の入力(bit 対応)の特定ビットを操作し、送信します<br>入力(bit 対応) = 入力(bit 対応) & ~(1 << num) | (data << num)</td></tr>
<tr><td>構文</td><td>CANA31API int32_t abh3_can_inBitSet(int8_t num,int8_t data,pCANABH3_RESULT pPtr)</td></tr>
<tr><td>パラメータ</td><td>
<table>
<tr><th>変数名</th><th>内容</th></tr>
<tr><td>num</td><td>ビット番号(0～31)</td></tr>
<tr><td>data</td><td>設定データ(0～1)</td></tr>
<tr><td>pPtr</td><td>通信結果を受け取る領域へのポインタ<br>（pPtr->u.DPOR に格納される）</td></tr>
</table>
</td></tr>
<tr><td>戻り値</td><td>
<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>
</td></tr>
<tr><td>注意点等</td><td>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</td></tr>
</table>

abh3_can_reqPulse

| 概要 | 積算値のリクエスト |
|---|---|
| 詳細 | 以下の要素を取得します<br>要素名<br>A パルス積算値<br>B パルス積算値 |
| 構文 | CANA31API int32_t abh3_can_reqPulse(pCANABH3_RESULT pPtr) |
| パラメータ | 変数名：pPtr<br>内容：通信結果を受け取る領域へのポインタ<br>（pPtr->u.DP1R に格納される） |
| 戻り値 | 戻り値：0 / 内容：正常終了<br>戻り値：0 以外 / 内容：異常終了時のエラーコード |
| 注意点等 | |

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 | 変数名：num / 内容：番号(0x00～0xff)<br>変数名：pPtr / 内容：通信結果を受け取る領域へのポインタ<br>格納先は、「注意点等」を参照の事 |
| 戻り値 | 戻り値：0 / 内容：正常終了<br>戻り値：0 以外 / 内容：異常終了時のエラーコード |
| 注意点等 | 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<br>内容に関しては、構造体の項を参照の事 |

| 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：vel　内容：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：vel　内容：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：cur　内容：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：cur　内容：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: load / 内容: 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: analog / 内容: 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: volt / 内容: 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 | - | - |