NonOS 対応 CAN 通信ミドルウェア

ユーザーズマニュアル

Ver.1.00　2007 年 10 月 25 日

株式会社サニー技研

TOPPERS NonOS CAN Middleware

Toyohashi Open Platform for Embedded Real-Time Systems

NonOS CAN Middleware

目次

１．　概要

本ミドルウェアは自動車用の CAN 通信機能として次の特徴を備えており、単独としての使用だけでなく、ネットワークに接続されたすべてのノードに適用することにより、車載用として信頼性の高い、高品質な通信を実現可能になっている。また CAN 通信そのものを本ミドルウェアで隠蔽化するため、設計者は CAN が接続されていることを意識せずに本来のプログラミングに注力することが可能な設計となっている。

・CAN プロトコルを意識せずに利用が可能
・データ通信として複数のパターンを設定
・エラー発生時のリカバリ機能を装備
・省 ROM/RAM 化を実現
・低消費電力モードへの対応
・ダイレクトモニタリングおよびインダイレクトモニタリングによるネットワークマネジメントの装備
・OSEK/VDX Communication Ver.3.02 および OSEK/VDX Netwark Management Concept and Application Programming Interface Version 2.5.2 に準拠
・MISRA-C 準拠の C 言語にて提供され、MCU 依存部分と汎用部分とを分離した構成

２．　システム要件

２．１ ハードウェア要件

本ミドルウェアは次の MCU を実装ターゲットとしてとして開発している。

MCU	: ルネサステクノロジ M32C/85
クロック	: 32MHz
CAN コントローラ	: M32C/85 搭載 Full CAN コントローラ チャンネル 0
ハードウェアタイマ	: ミドルウェアとしては未使用
割り込み	: CAN 受信完了割り込み、CAN 送信完了割り込み、 CAN エラー割り込み、CAN ウェイクアップ割り込み

２．２ ソフトウェア要件

（１）開発言語

本ミドルウェアは C 言語により記述され、以下のコンパイラを使用することを想定する。

ルネサステクノロジ NC308WA Ver.5.20 Release 02

コンパイラの最適化オプションはなしで実施し、動作を検証する。

（２）定周期タイミング通知

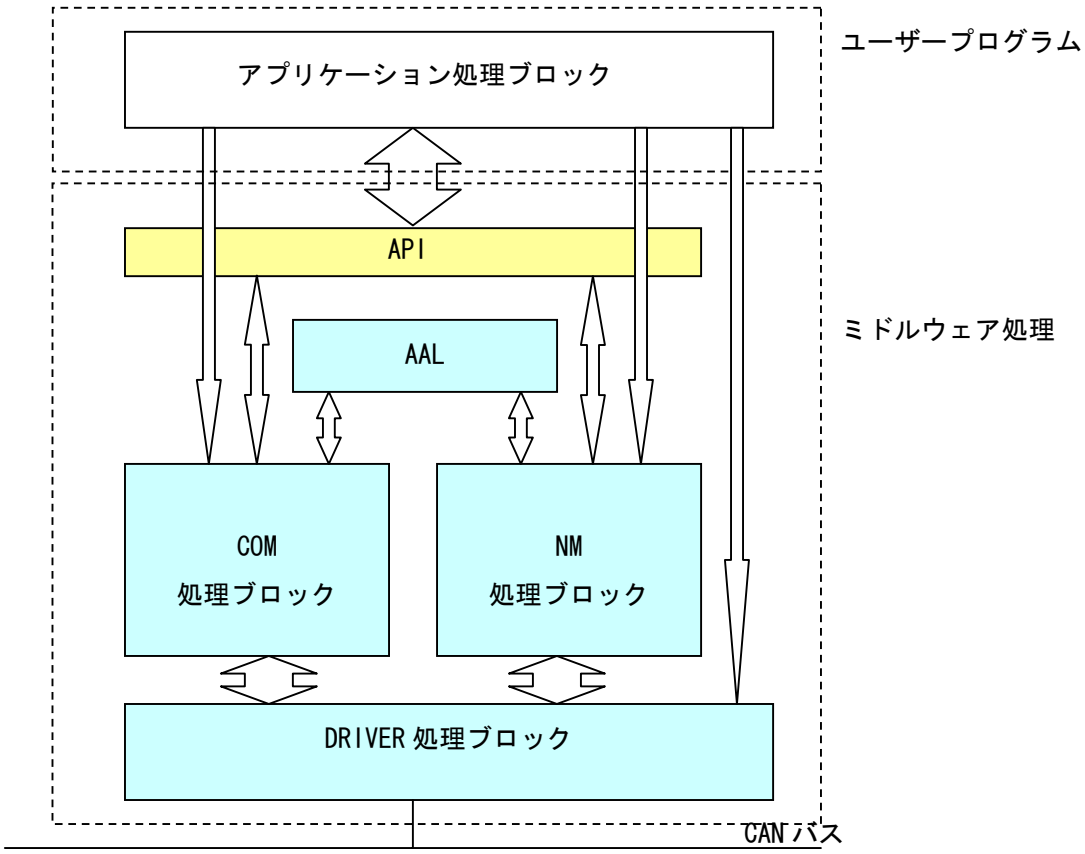
ミドルウェア内部の時間監視のために、アプリケーション側から定周期でミドルウェアの関数を呼び出す必要がある。この呼び出しは次の２つが必要である。

- メイン定周期（メイン周期から呼び出し）
- 割り込み定周期（割り込みから呼び出し可能）

３．　システム仕様

３．１ システム構成

ミドルウェアの内部ブロック図を下記に記す。



ミドルウェアは COM 処理ブロック/NM 処理ブロック/DRIVER 処理ブロック/ AAL(アプリケーションアシスタント層)の各ブロックに分かれ、アプリケーション処理ブロックからは共有メモリないし関数を介した API でインタフェースする方式とする。

各ブロックの機能を以下に説明する。

３．２ 機能仕様

１）アプリケーション処理ブロック

アプリケーション処理ブロック（以降、アプリケーションと略）はユーザがシステムの実制御処理を記載する部分を記す。

２）COM 処理ブロック

COM 処理ブロック（以降 COM と略）はアプリケーションデータの送受信制御を行うモジュールとして、本ミドルウェアの中核をなす機能を持つ。
COM が制御する内容を以下に記す。

- ・送信バッファ・受信バッファのデフォルト値設定
- ・フレーム単位の一括書き込み・読み出し
- ・送信イベント検出（COM が持つ送信バッファを使用）
- ・周期送信時間管理
- ・COM メッセージ送信間隔の確保
- ・送信途絶判定
- ・受信途絶判定
- ・データ定義長（DLC）チェック

３） NM 処理ブロック

NM 処理ブロック（以降 NM と略）は、通常のデータ通信とは独立してネットワークに接続されたノードの状態監視をするもので、以下の機能を有する。

- ・アプリケーション(API)で相互作用させるインタフェース
- ・ノードモニターするためのアルゴリズム
  （接続ノードの管理）
- ・スリープモードへの変遷のためのアルゴリズム
  (Sleep移行の有効／無効判定、Sleep／WakeUp制御)
- ・NM プロトコルデータユニット(NMPDU)

４）DRIVER 処理ブロック

DRIVER 処理ブロック（以降 DRIVER と略）は COM/NM から呼び出され、CAN 基本的な送受信制御を行う。COM/NM からはデバイスが持つ CAN コントローラは隠蔽化され、抽象化された CAN 入出力手段として取り扱われる。DRIVER の持つ機能は以下の通りである。

・COM・NM の送信メッセージ送信
・送信割り込みによる送信完了通知
・受信メッセージの、COM・NN への振り分け
・CAN バスオフ割り込みによる通信不能状態の通知
・ハードウェアチェック

５）AAL(アプリケーションアシスタント層)

AAL は本来ならばアプリケーションが COM、NM に行うサービスを提供することで、処理の自動化、共通化を行う。

現状行っている処理を以下に示す。

・NM から COM 送受信許可状態を取得し、COM に伝える。

６）API

アプリケーションと COM/NM つなぐインタフェースを示す。これは実際に処理を行うブロックではなくアプリケーションとデータを交換するためのバッファや関数サービスを表すものである。

３．３　ファイルの種類

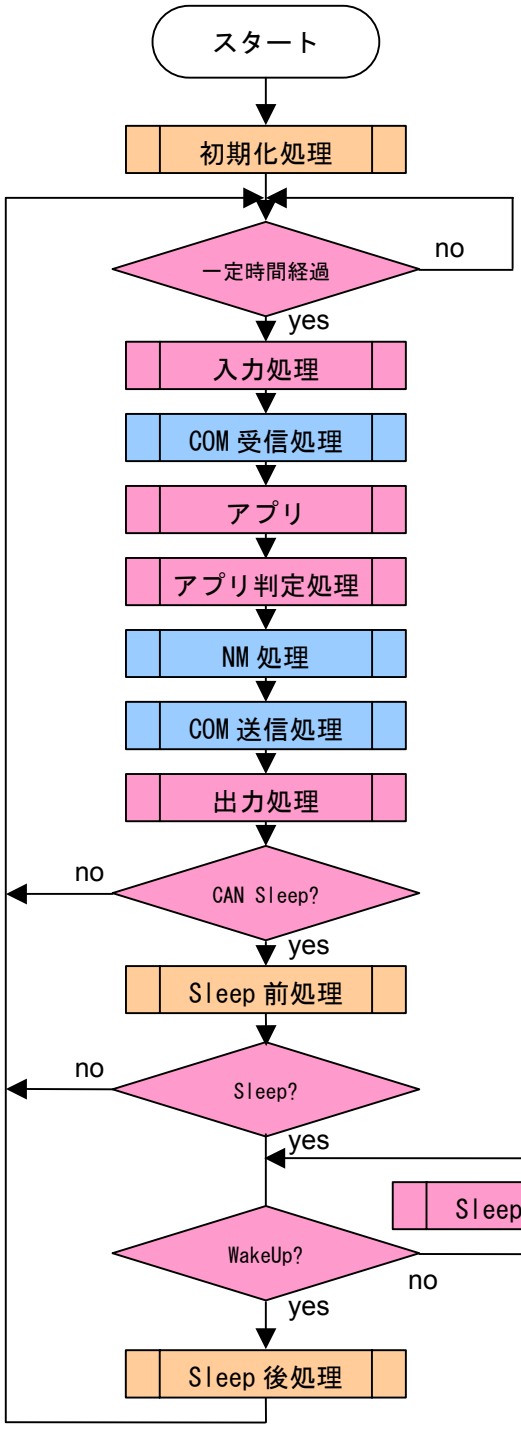
本ソフトウェアソフトは、以下のファイルで構成される。これらのファイル名は、全て固定とする。メインプログラムなどのプログラムは、ファイル名は自由とする。

フォルダ名	ファイル名	属性	内容	備考
AAL	can_aal.c	AAL	AAL ソース	変更可能
COM	can_com.c	COM	COM ソース	変更禁止
NM	can_nm.c	NM	NM ソース	変更禁止
DRV	can_drv.c	DRIVER	DRIVER ソース	変更禁止
COM	can_com.h	COM	COM 専用ヘッダ	変更禁止
NM	can_nm.h	NM	NM 専用ヘッダ	変更禁止
DRV	can_drv.h	DRIVER	DRIVER 専用ヘッダ	変更禁止
DRV	can_drv_sfr.h	DRIVER	CAN 用 SFR 定義	変更禁止
INCLUDE	can_cmn.h	共通	各種定義	変更禁止
INCLUDE	can_com_ext.h	COM	COM 公開ヘッダ	変更禁止
INCLUDE	can_nm_ext.h	NM	NM 公開ヘッダ	変更禁止
INCLUDE	can_drv_ext.h	DRIVER	DRIVER 公開ヘッダ	変更禁止
INCLUDE	can_nm_csw.h	NM	コンパイルスイッチ用ヘッダ	変更禁止
INCLUDE	can_drv_csw.h	DRIVER	コンパイルスイッチ用ヘッダ	変更禁止
任意	can_com_def.h	COM	システム依存情報	SG 生成
任意	can_nm_def.h	NM	システム依存情報	SG 生成
任意	can_drv_def.h	DRIVER	システム依存情報	SG 生成
任意	can_com_table.c	COM	システム個別通信情報	SG 生成
任意	can_nm_table.c	NM	システム個別通信情報	SG 生成
任意	can_drv_table.c	DRV	システム個別通信情報	SG 生成
任意	can_com_api.h	COM	送受信データ定義	SG 生成
任意	can_com_api.c	COM	送受信データ実体	SG 生成

４．　ミドルウェアの呼び出しとシステム構成法

４．１　メインプログラム仕様

下記に本ミドルウェアを使用する際に基本となる処理アルゴリズムと、必須呼び出しとなるCOM・NM・DRIVERの関数を示す。




```
#include "can_cmn.h"        /* ヘッダファイル */
#include "can_drv_csw.h"    /* DRV 関連 */
#include "can_drv_def.h"
#include "can_drv_ext.h"
#include "can_com_def.h"    /* COM 関連 */
#include "can_com_ext.h"
#include "can_nm_csw.h"     /* NM 関連 */
#include "can_nm_def.h"
#include "can_nm_ext.h"
#include "can_com_api.h"
```

初期化処理

```
SetCANIntLVLDRV(); /* CAN 割込みレベル初期化 */
InitCOM();         /* COM 初期化関数 */
InitNM();          /* NM 初期化関数 */
```

```
RcvCOM();          /* COM 受信処理 */
```

```
TickNM();          /* NM 処理*/
```

```
SndCOM();          /* COM 送信関数 */
```

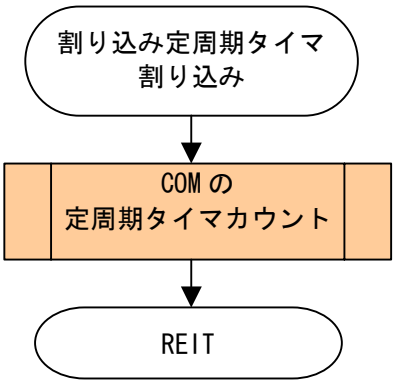
```
GetStatus();       /* NM ネットワーク情報取得 */
```

```
FncBeforeSlpNM()   /* Sleep 前処理 */
```

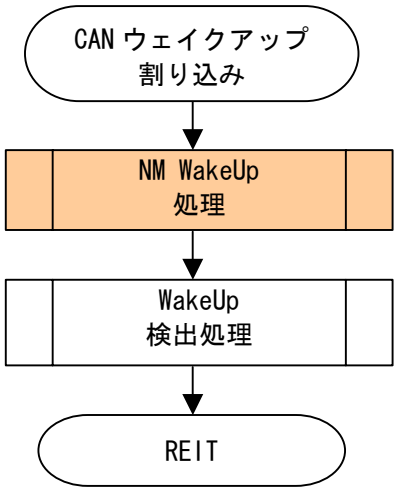
```
GetWkupStsNM();    /* ウェイクアップ割り込み
                                発生有無確認 */
```

```
FncAfterSlpNM();   /* Sleep 後処理 */
```

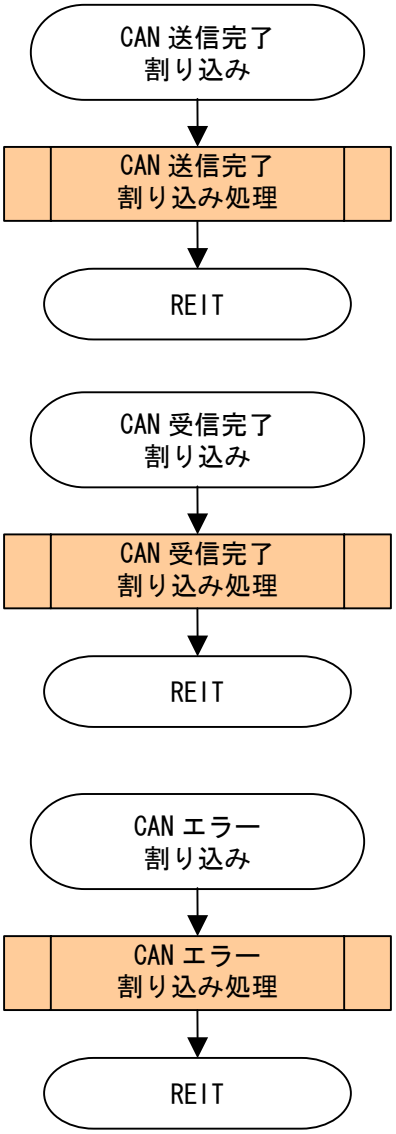
本ミドルウェアはシングルタスク上での動作を想定しており、マルチスレッドや割り込みによる API 動作保証はなされていない。よってシステムを構成する場合は１つの処理ループ中（他のアプリケーションによりロックされないループ処理中）に COM・NM・DRIVER の指定関数を呼び出す必要がある。加えて COM/NM/DRIVER 処理自体は内部に監視用のタイマを持っていないため、アプリケーションより一定周期で COM・NM・DRIVER 毎により時間管理を行う構造をとる。よってミドルウェアを含むメイン周期は必ず一定周期である必要がある。他にも COM 送信処理や DRIVER オーバータイム処理に必要な割り込み定周期タイマでグローバルなカウンタを保持する必要がある。




```
CntCOM();       /* COM の定周期タイマカウント */
```




```
FncAfterSlpNM ();/* NMWakeUp 処理
                    （Sleep 後処理） */
```




```
FncCANSndIntDRV();/* CAN 送信完了割り込み処理 */
```

```
FncCANRcvIntDRV();/* CAN 受信完了割り込み処理 */
```

```
FncCANErrIntDRV();/* CAN エラー割り込み処理 */
```

４．２　API バッファ仕様

アプリケーションは、任意のタイミングで、CAN フレームを意識せずに、送信 API バッファへのアプリケーションデータの書き込み、及び受信 API バッファからのアプリケーションデータの読み出しを行うことが可能である。ただし、割り込み処理内での書き込み・読み出しは禁止とする。

・送信 API バッファ

送信するアプリケーションデータを格納するバッファである。送信するデータ ID 分の送信データが配列として構成される。各データは、あらかじめ CAN で送信するイメージで構成していくこととする。

配列名は、`api_snddata` である（SG にて can_com_api.c に自動生成される）。

COM の送信データテーブルと、送信 API バッファの配列の並びは１対１で対応しているものとする。

・受信 API バッファ

受信する受信メッセージを格納するバッファである。受信するデータ ID 分の受信データが配列として構成される。各データは、あらかじめ CAN で送信するイメージで構成していくこととする。

配列名は、`api_rcvdata` である（SG にて can_com_api.c に自動生成される）。

COM の受信データテーブルと、送信 API バッファの配列の並びは１対１で対応しているものとする。

尚、送受信 API バッファは COM 初期化(InitCOM 関数呼び出し)時か送受信 API バッファ初期化処理(InitBufferCOM 関数呼び出し時)にクリアされる。

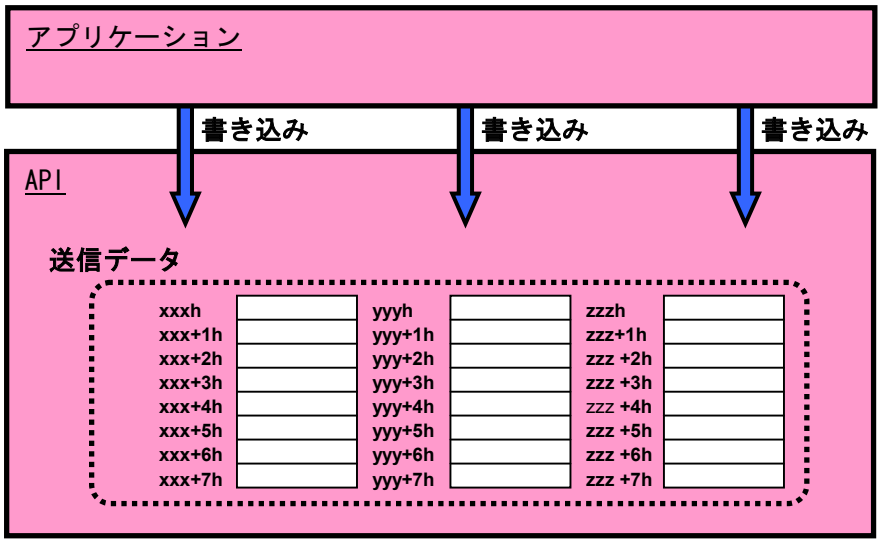
なお InitBufferCOM 関数はウェイクアップ処理時やエラー検出時等、ユーザ任意のタイミングで呼び出す必要がある。

※InitBufferCOM 関数実施時には COM の送受信内部情報もクリアされる。そのため、InitBufferCOM 関数実施後は電源投入時と同じ動作で開始される。

４．２．１ 送信 API バッファ

送信 API バッファとは、アプリケーションが送信するアプリケーションデータを書き込むためのバッファである。アプリケーションが使用する送信データで構成される。COM は、この送信データから送信メッセージを構築して送信キューにセットする。これらの送信データは 8 バイト固定とし、RAM に連続して並んでいるものとする。なお、SG により生成される can_com_api.h において送信データがシンボル単位で定義されており、CAN フレームを意識せずに、アプリケーションデータを書き込むことができる。

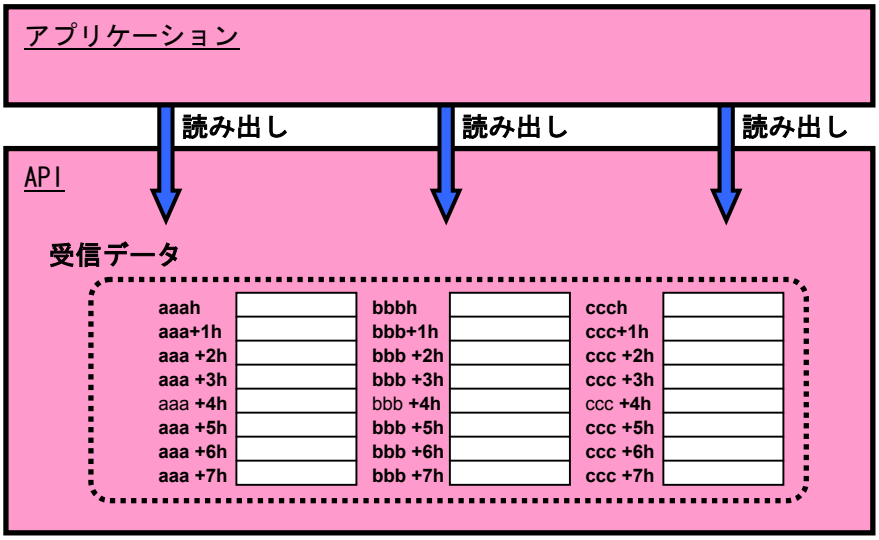
送信 API バッファの構成を以下に示す。



４．２．２　受信 API バッファ

受信 API バッファとは、アプリケーションが受信するアプリケーションデータを読み出すためのバッファである。アプリケーションが使用する受信データの形で構成される。COM は受信メッセージを、受信メッセージのデータ ID と一致する受信データに自動的に格納する。これらの受信データは８バイト固定とし、RAM に連続して並んでいるものとする。なお、SG により生成される can_com_api.h において受信データがシンボル単位で定義されており、CAN フレームを意識せずに、アプリケーションデータを読み出すことができる。

受信 API バッファの構成を以下に示す。



５．　アプリケーションインターフェース関数仕様

５．１　COM　API 関数

インタフェース名	実行タイミング	概要
void InitCOM(void)	APP 初期化処理時	COM 初期化処理
void InitBufferCOM(void)	任意	共有送受信バッファ初期化処理
StatusType ResetSndCOM(void)	送信途絶時	送信途絶時 CAN コントローラ初期化
void SndCOM(void)	メイン周期	COM 送信処理
StatusType SendMessage (UINT16 , UINT8 , UINT8 far *)	任意	直接送信処理
void RcvCOM(void)	メイン周期	COM 受信処理
StatusType GetStsCOM(COMStsType *)	任意	COM 内部ステータス公開処理
StatusType GetSndMsgStsCOM (UINT16 , CANAPIDataStsType *)	任意	送信データ ID ステータス公開処理
StatusType GetRcvMsgStsCOM (UINT16 , CANAPIDataStsType *)	任意	受信データ ID ステータス公開処理
StatusType GetRcvAllMsgStsCOM (COMRcvDataStsType **)	任意	全受信データ ID ステータス公開処理
void CntCOM(void)	1msec タイマ割り込み処理時	システムタイマ更新処理
StatusType SetSndUseReqCOM (UINT16, UINT8)	任意	送信許可/禁止要求

（１）初期化関数

Service Name:	InitCOM
Syntax:	void InitCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	COM 内部変数、送受信キュー、共有送受信バッファを初期化する。
Return:	なし

（例）

```
/*  アプリケーションのメインプログラムの初期化関数内  */
InitCOM();
```

Service Name:	InitBufferCOM
Syntax:	void InitBufferCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	共有送受信バッファを初期化する。
Return:	なし

（例）

```
/*  必要に応じて呼び出し  */
InitBufferCOM();
```

Service Name:	ResetSndCOM
Syntax:	StatusType ResetSndCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	送信途絶時に CAN コントローラ初期化する。
Return:	StatusType CAN_E_OK : 初期化成功 CAN_E_NG : 初期化失敗

（２）送受信関連

Service Name:	SndCOM
Syntax:	void SndCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	他ノードへ送信するメッセージがあれば、共有送信バッファのデータを送信メッセージとし、送信キューに格納し送信する。 また、送信途絶判定を行う。
Return:	なし

（例）

```
/*  アプリケーションのメインプログラムのメイン定周期内  */
SndCOM();
```

Service Name:	SendMessage
Syntax:	StatusType SendMessage(UINT16 , UINT8 , UINT8 far *)
Parameter (Input):	UINT16 : 送信メッセージのデータ ID UINT8 : 送信メッセージの DLC UINT8 far * : 送信メッセージの送信データのポインタ
Parameter (Output):	なし
Description:	直接送信キューに送信メッセージを格納する。
Return:	StatusType CAN_E_OK : 送信キューへの格納成功 CAN_E_NG : 送信キューへの格納失敗

（例）

```
/*  必要に応じて（ダイアグメッセージ送信処理等）  */
UINT16   DataID;
UINT8    DLC;
UINT8    ptr_msg[8];
if(SendMessage(DataID, DLC, ptr_msg) == CAN_E_OK);
{
        /*  ダイアグメッセージ正常送信終了  */
}
```

Service Name:	RcvCOM
Syntax:	Void RcvCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	他ノードからの受信メッセージがあれば、受信メッセージのデータを共有受信バッファへ格納する。また、受信途絶判定を行う。
Return:	なし

（例）

```
/*  アプリケーションのメインプログラムのメイン定周期内  */
RcvCOM();
```

Service Name:	SetSndUseReqCOM
Syntax:	StatusType SetSndUseReqCOM(UINT16, UINT8)
Parameter (Input):	UINT16：　送信データ ID UINT8：　送信許可/禁止ステータス CAN_COM_SND_OFF　送信禁止状態 CAN_COM_SND_ON　送信許可状態
Parameter (Output):	なし
Description:	データの送信許可/禁止を設定する。 InitCOM 後、送信全スロットは送信禁止状態となるので、送信する場合は当該関数にて送信許可設定する必要がある。
Return:	StatusType CAN_E_OK：　送信データ ID 該当あり CAN_E_NG：　送信データ ID 該当なし

（３）ステータス公開関連

Service Name:	GetStsCOM
Syntax:	StatusType GetStsCOM(COMStsType *)
Parameter (Input):	なし

Parameter
(Output):
	COMStsType： COM 内部ステータス格納領域
	typedef union{
	struct{
	BITFIELD_UINT8 en_sndque:1; /* 送信キュー許可 */
	BITFIELD_UINT8 en_snd:1; /* 送信許可 */
	BITFIELD_UINT8 en_rcvque:1; /* 受信キュー許可 */
	BITFIELD_UINT8 en_rcv:1; /* 受信許可 */
	BITFIELD_UINT8 sndstop:1; /* 送信途絶状態 */
	BITFIELD_UINT8 rcvstop:1; /* 受信途絶状態 */
	BITFIELD_UINT8 dummy:2;
	}bit;
	UINT8 byte;
	}COMStsType;

Description:	COM のステータス情報を取得することができる。
Return:	StatusType CAN_E_OK： 正常 CAN_E_NG： 異常

（例）

```
/*  必要に応じて（アプリケーションから呼び出し）  */
COMStsType        *Sts;
if(GetStsCOM(*Sts) == CAN_E_OK);
{
        if(Sts.bit.sndstop == CAN_TRUE)
        {
                /*  COM 送信途絶中！！  */
        }
        if(Sts.bit.rcvstop == CAN_TRUE)
        {
```

```
                    /*  COM受信途絶中！！  */
                }
            }
```

Service Name:	GetSndMsgStsCOM
Syntax:	StatusType GetSndMsgStsCOM(UINT16 , CANAPIDataStsType *)
Parameter (Input):	UINT16： 送信データ ID

Parameter
(Output):
	CANAPIDataStsType *： 送信データ ID のステータス格納領域
	typedef union{
	struct{
	BITFIELD_UINT8 locked:1; /* 送信条件未成立状態 */
	BITFIELD_UINT8 ok:1; /* 正常送信状態 */
	BITFIELD_UINT8 backup:1; /* 送信キューフルによりバックアップ状態 */
	BITFIELD_UINT8 limit:1; /* 送信キューフルにより送信不可状態 */
	BITFIELD_UINT8 ng:1; /* 送信失敗状態 */
	BITFIELD_UINT8 cond:1; /* 送信条件成立状態 */
	BITFIELD_UINT8 tblvalid:1; /* 通信データ仕様テーブルのデータ整合性 OK */
	BITFIELD_UINT8 dummy:1;
	}bit;
	UINT8 byte;
	}CANAPIDataStsType;

Description:	各送信データ ID のステータス情報を取得することができる。
Return:	StatusType CAN_E_OK： 正常 CAN_E_NG： 異常

（例）

```
/*  必要に応じて（アプリケーションから呼び出し）  */
UINT16    DataID;
union CANAPIDataStsType      Sts;
if(GetSndMsgStsCOM(DataID, &Sts) == CAN_E_OK);
{
        if(Sts.snddata.bit.backup == CAN_TRUE)
        {
                /*  指定した送信データ ID は送信キューフルの為、待機中！！  */
        }
}
```

Service Name:	GetRcvMsgStsCOM
Syntax:	StatusType GetRcvMsgStsCOM(UINT16 , CANAPIDataStsType *)
Parameter (Input):	UINT16： 受信データ ID

Parameter (Output):	CANAPIDataStsType *： 受信データ ID のステータス格納領域
	typedef union{
	struct{
	BITFIELD_UINT8 rcv:1; /* 起動後、受信成功 */
	BITFIELD_UINT8 newdata:1; /* １周期間、受信成功受信 */
	BITFIELD_UINT8 silent:1; /* 受信途絶状態 */
	BITFIELD_UINT8 ng:1; /* 受信失敗状態 */
	BITFIELD_UINT8 pwon:1; /* 起動時状態フラグ */
	BITFIELD_UINT8 dummy:3;
	}bit;
	UINT8 byte;
	}CANAPIDataStsType;

Description:	各受信データ ID のステータス情報を取得することができる。
Return:	StatusType CAN_E_OK： 正常 CAN_E_NG： 異常

（例）

```
/*  必要に応じて（アプリケーションから呼び出し）  */
UINT16   DataID;
union CANAPIDataStsType     Sts;
if(GetRcvMsgStsCOM(DataID, &Sts) == CAN_E_OK);
{
        if(Sts.rcvdata.bit.newdata == CAN_TRUE)
        {
                /*  指定した ID は新しいデータの受信有り！！  */
        }
}
```

Service Name:	GetRcvAllMsgStsCOM
Syntax:	StatusType GetRcvAllMsgStsCOM(COMRcvDataStsType **out_sts_adr)
Parameter (Input):	なし
Parameter (Output):	CANAPIDataStsType **:　受信データ ID のステータス格納領域 typedef struct{ 　　CANAPIDataStsType sts;　　/* ステータス */ }COMRcvDataStsType;
Description:	全受信データ ID のステータス情報を取得することができる。
Return:	StatusType CAN_E_OK：　　正常 CAN_E_NG：　　異常

（４）システム関連

Service Name:	CntCOM
Syntax:	void　CntCOM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	割り込み定周期タイマ処理内でカウンタを更新し、COM 内部タイマとして使用する。
Return:	なし

（例）

```
/*　アプリケーションの割り込み定周期タイマ割り込みプログラムの関数内　*/
CntCOM();
```

５．２　NM　API 関数

インタフェース名	実行タイミング	概要	Direct	Indirect
void InitNM(void)	APP 初期化処理時	NM の初期化を行う。	○	○
void TickNM(void)	メイン周期	アプリケーション（メイン周期）にて起動する。	○	○
StatusType GoToMode(UINT8)	APP 初期化処理時 メイン周期	Sleep許可/禁止をNMに通知する。	○	○
void FncAfterSlpNM(void)	Sleep 復帰時	WakeUp 後の処理を行う。	○	○
void FncBeforeSlpNM(void)	Sleep 移行時	Sleep 前処理を行う。	○	○
StatusType GetStatus(NetStsType*)	Sleep移行許可判定時	ネットワークマネジメントのステータスを返す。	○	○
StatusType GetExtStatus(ExtNetStsType*)	任意	ネットワークマネジメントの拡張ステータスを返す	-	○
StatusType GetStsNM(NMStsType*)	任意	NM の送受信の情報を返す。	○	-
StatusType GetWkupStsNM (void)	任意	ウェイクアップ割り込みの発生/未発生を通知する。	○	○
StatusType GetForm(UINT8** , UINT8**)	任意	論理リングの構成要素を返す。	○	-
StatusType GetConfig (CfgName , CfgRefType**)	任意	ネットワークコンフィグレーションを返す。	○	○
StatusType FncGetNMNodeID (UINT8, UINT8 *)	任意	ノード ID の取得	-	○

（１）初期化関数

Service Name:	InitNM
Syntax:	void　InitNM(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	NM の変数の初期化や CAN コントローラの設定を行う。
Return:	なし

(例)

```
/* メインプログラムの初期化関数内 */
InitNM();
```

（２）共通関数

Service Name:	TickNM
Syntax:	void TickNM (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	NM の各状態処理、COM の送受信許可/禁止の設定、CAN コントローラの動作/停止等を行う。
Return:	なし

(例)

```
/* メインプログラムの一定周期内 */
TickNM();
```

Service Name:	GoToMode
Syntax:	StatusType GoToMode (UINT8 in_req)
Parameter (Input):	UINT8 CAN_NM_BUS_SLEEP NM への Sleep 許可 CAN_NM_AWAKE NM への Sleep 禁止
Parameter (Output):	なし
Description:	Sleep 許可、又は Sleep 禁止を NM に通知する。
Return:	StatusType CAN_E_OK 正常終了 CAN_E_NG 設定失敗、無効な引数が渡された

(例)

```
/* アプリケーションソフト内の処理 */
if( GoToMode(CAN_NM_BUS_SLEEP) == CAN_E_OK )
{
        /* Sleep 状態移行処理 */
        ・・・・・
        ・・・・・
}
```

Service Name:	FncAfterSlpNM
Syntax:	void FncAfterSlpNM (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	WakeUp の後処理を行う。
Return:	なし

(例)

```
/* メインプログラムの CAN バスウェイクアップ割り込み処理 */
#pragma INTERRUPT can_wkupint       注)割り込み関数名はユーザ定義
void can_wkupint(void);
void can_wkupint()
{
        FncAfterSlpNM();
        ・・・・・
        ・・・・・
}

/* アプリケーションソフト内の Sleep 復帰による初期化処理 */
FncAfterSlpNM();
・・・・・
・・・・・
```

Service Name:	FncBeforeSlpNM
Syntax:	void FncBeforeSlpNM (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	Sleep 移行の前処理を行う。
Return:	なし

(例)

```
/* アプリケーションソフト内の Sleep 移行前処理 */
FncBeforeSlpNM();
        ・・・・・
 ・・・・・
```

NM ステータス関数

Service Name:	GetStatus
Syntax:	StatusType GetStatus(NetStsType *out_sts)
Parameter (Input):	なし
Parameter (Output)	NetStsType　　NM ネットワークステータス情報 typedef union{ struct{BITFIELD_UINT8 StatusRingStable:1;　/* ネットワークが安定 */ BITFIELD_UINT8 StatusBusError:1;　/* バスオフ状態 */ BITFIELD_UINT8 StatusNMActive:1;　/* NMActive 状態(無効) */ BITFIELD_UINT8 StatusLimpHome:1;　/* NMLimpHome 状態 */ BITFIELD_UINT8 StatusBusSleep:1;　/* NMBusSleep 状態 */ BITFIELD_UINT8 StatusWaitBusSleep:1;　/* NMTwbsNormal, NMTwbsLimpHome 状態 */ BITFIELD_UINT8 StatusTxRingDataNotAllowed:1;　/* RingMsg のデータ領域へのアクセス禁止 */ BITFIELD_UINT8 StatusBusSleepInd:1;　/* Sleep.ind 状態 */ }bit; UINT8 byte; }NetStsType;
Description:	NM ネットワークステータスを取得する。 DirectNM 使用時
Return:	StatusType CAN_E_OK 正常取得完了 CAN_E_NG 取得失敗

(例)

```
/* アプリケーションソフト内 */
NetStsType NetSts;
if(GetStatus(&NetSts) == CAN_E_OK)
{
        if(NetSts.bit.StateBusSleep)
        {
                /* Sleep 処理 */
                ・・・・・
```

```
        }
    }
```

Service Name:	GetStatus
Syntax:	StatusType GetStatus(NetStsType *out_sts)
Parameter (Input):	なし

Parameter (Output)	NetStsType　　NM ネットワークステータス情報
	typedef union{
	struct{
	BITFIELD_UINT8 StatusBusError:1;　　/* バスオフ状態 */
	BITFIELD_UINT8 StatusLimpHome:1;　　/* NMLimpHome 状態 */
	BITFIELD_UINT8 StatusBusSleep:1;　　/* NMBusSleep 状態 */
	BITFIELD_UINT8 StatusWaitBusSleep:1; /* NMWaitBusSleep 状態 */
	BITFIELD_UINT8 reserve:4;　　/* reserve4 */
	}bit;
	UINT8 byte;
	}NetStsType;

Description:	NM ネットワークステータスを取得する。 IndirectNM 使用時
Return:	StatusType CAN_E_OK 正常取得完了 CAN_E_NG 取得失敗

(例)

```
/* アプリケーションソフト内 */
NetStsType NetSts;
if(GetStatus(&NetSts) == CAN_E_OK)
   {
        if(NetSts.bit.StateBusSleep)
        {
            /* Sleep 処理 */
            ・・・・・
        }
   }
```

Service Name:	GetExtStatus
Syntax:	StatusType GetExtStatus(ExtNetStsType *out_sts)
Parameter (Input):	なし

Parameter (Output)	ExtNetStsType NM 拡張ネットワークステータス情報
	typedef union{
	struct{
	BITFIELD_UINT8 StateExtd:2; /* 拡張ステータス */
	BITFIELD_UINT8 reserve:6; /* reserve4 */
	}bit;
	UINT8 byte;
	}ExtNetStsType;

Description:	NM 拡張ネットワークステータスを取得する。
Return:	StatusType CAN_E_OK 正常取得完了 CAN_E_NG 取得失敗

(例)

```
/* アプリケーションソフト内 */
ExtNetStsType ExtNetSts;
if(GetExtStatus(&ExtNetSts) == CAN_E_OK)
{
   if(ExtNetSts.bit.StateExtd == CAN_NM_EXTNETSTS_NOT_POSSIBLE)
   {
        ・・・・・
   }
}
```

Service Name:	GetStsNM
Syntax:	StatusType GetStsNM(NMStsType *out_ptr_sts)
Parameter (Input):	なし
Parameter (Output):	NMStsType NMの通信状態 typedef struct{ UINT8 sndstop:1; /* [1:送信不能状態, 0:送信可能状態] */ UINT8 rcvstop:1; /* [1:受信不能状態, 0:受信可能状態] */ UINT8 dummy:6; }NMStatusType;
Description:	NMの送受信の状態を取得する。 DirectNM使用時
Return:	StatusType CAN_E_OK正常取得完了 CAN_E_NG取得失敗

(例)

```
/* アプリケーションソフト内 */
NMStsType       NMSts;
if(GetStsNM(&NMSts) == CAN_E_OK)
{
        if(NMSts.sndstop)
        {
                /* 送信不能検出処理 */
                ・・・・・
        }
}
```

Service Name:	GetWkupStsNM
Syntax:	void GetWkupStsNM (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	ウェイクアップ割り込みの発生/未発生を通知する。
Return:	StatusType CAN_E_OK ウェイクアップ割り込み発生 CAN_E_NG ウェイクアップ割り込み未発生

(例)

```
/* メインプログラムの WakeUp 判定処理に記述 */
while( (GetWkupStsNM() == CAN_E_NG) /* ウェイクアップ割り込み発生判定 */
&& (wakeup_flag == CAN_OFF) )       /* ユーザの WakeUp フラグ判定 */
{
    ・・・・・
    ・・・・・
}
```

Service Name:	GetForm
Syntax:	StatusType GetForm(UINT8 **out_ptr_adr , UINT8 ** out_ptr_num)
Parameter (Input):	なし
Parameter (Output):	UINT8 *out_ptr_adr 論理リング構成テーブル先頭アドレス（論理リング構成要素ノード番号を格納） UINT8 *out_ptr_num 接続ノード数
Description:	論理リング構成テーブルの先頭アドレスと論理リング構成ノード数を取得することができる。 NM は最大接続ノード数分の大きさのテーブルを持っており、テーブルを読み出すことで接続されているノード ID を知ることができる。 DirectNM 使用時
Return:	StatusType CAN_E_OK 格納完了 CAN_E_NG 論理リングネットワークが不安定

(例)

```
/* ダイアグ処理内 */
UINT8   **RingAdr;
UINT8   **NodeNum;
if(GetForm (RingAdr, NodeNum) == CAN_E_OK)
{
        ・・・・・
}
```

Service Name:	GetConfig
Syntax:	StatusType GetConfig(CfgKndName in_cfg_knd , CfgRefType **out_ptr_adr)
Parameter (Input):	CfgKndName CAN_Normal ネットワーク参加ノード情報の取得 CAN_LimpHome LimpHomeMsg 送信ノード情報の取得
Parameter (Output):	UINT8 **out_ptr_adr ネットワークコンフィグレーションテーブル先頭アドレス
Description:	ネットワークコンフィグレーションテーブルの先頭アドレスを取得することができる。ネットワークコンフィグレーションテーブルとはネットワーク上のノードの ID を登録するテーブルである。 テーブルには AliveMsg または RingMsg を送信した(ネットワークに参加している)ノード ID を登録するプレゼントコンフィグレーションテーブルと LimpHomeMsg を送信したノード ID を登録するリンプホームコンフィグレーションテーブルの 2 種類があり、NM はコンフィグレーションテーブルとして 8byte の領域を持つ。それぞれのビットが各ノードの ID と対応しており、セットされているビットを検索することで接続されているノード ID を知ることができる。 DirectNM 時
Return:	StatusType CAN_E_OK 格納完了 CAN_E_NG 格納失敗

(例)

```
/* ダイアグ処理内 */
UINT8    *ConfAdr;
if(GetConfig(CAN_Normal, &ConfAdr) == CAN_E_OK)
{
        ・・・・・
}
```

Service Name:	GetConfig
Syntax:	StatusType　GetConfig(CfgKndName in_cfg_knd , CfgRefType **out_ptr_adr)
Parameter (Input):	CfgKndName CAN_NM_CONFIG　　ネットワーク参加ノード情報の取得 CAN_NM_EXT_CONFIG　ネットワーク参加ノード拡張情報の取得
Parameter (Output):	UINT8 **out_ptr_adr　　ネットワークコンフィグレーション テーブル先頭アドレス
Description:	ネットワークコンフィグレーションテーブルの先頭アドレスを取得することができる。ネットワークコンフィグレーションテーブルとはネットワーク上のノードの ID を登録するテーブルである。 テーブルにはネットワークに参加しているノードを登録しており、NM はコンフィグレーションテーブルとして 8byte の領域を持つ。それぞれのビットが各モニタしているノードと対応しており、セットされているビットを検索することで接続されているノード ID を知ることができる。 拡張コンフィグレーション情報は OneMonitoring 時のみ取得できる。 IndirectNM 時
Return:	StatusType CAN_E_OK 格納完了 CAN_E_NG 格納失敗

（例）

```
/* ダイアグ処理内 */
UINT8    *ConfAdr;
if(GetConfig(CAN_NM_CONFIG,&ConfAdr) == CAN_E_OK)
{
        ・・・・・
}
```

Service Name:	FncGetNMNodeID
Syntax:	StatusType FncGetNMNodeID(UINT8 index, UINT8 *out_nodeid)
Parameter (Input):	UINT8 index　　参照対象箇所の指定
Parameter (Output):	UINT8 *out_nodeid　　ノード ID 格納用変数のアドレス
Description:	指定されたインデックスが監視ノード最大数の範囲内であればその箇所のノード ID を返す。
Return:	StatusType CAN_E_OK 格納完了 CAN_E_NG 格納失敗

(例)

```
UINT8    index = 3;
UINT8    m_nodeid;
if(FncGetNMNodeID (index, &m_nodeid) == CAN_E_OK)
{
        /* 自ノード？ */
        if(m_nodeid == MY_NODE_ID)
        {
                ・・・・・
        }
}
```

５．３ DRIVER　API 関数

ドライバは COM/NM より呼び出されるサービスで、ポーリングでも割り込みの両方から使用することが可能である。サービスとしては以下の関数が存在する。

インタフェース名	実行タイミング	概要
FncSetCANIntLVLDRV(void)	ECU 初期化時	CAN 割込みレベル設定処理
FncCANErrIntDRV (void)	CAN データ受信時	CAN エラー割り込み処理
FncCANSndIntDRV (void)	CAN データ送信時	CAN 送信完了割り込み処理
FncCANRcvIntDRV (void)	バスオフ発生時	CAN 受信完了割り込み処理
FncChgModCAN(UINT8)	任意	CAN 動作モード変更関数
FncCANInitDRV (void)	任意	CAN コントローラ初期化処理
FncCANResetDRV (void)	任意	CAN コントローラリセット処理
FncCANRunDRV (void)	任意	CAN コントローラ動作処理
FncCANSleepDRV (void)	任意	CAN コントローラスリープ処理
FncWrCOMMsgDRV (UINT32, UINT8, UINT8 *)	任意	COM 送信データ受け渡し関数
FncEnCANRcvInt(void)	任意	CAN 受信割込み有効関数
FncDisCANRcvInt(void)	任意	CAN 受信割込み無効関数
FncWrCOMMsgDRV (UINT32, UINT8, UINT8 *)	任意	COM 送信データ受け渡し関数
FncGetCOMSndStsDRV(UINT8 *)	任意	COM ステータス受け渡し関数
FncAbtCOMSndMsgDRV(void)	任意	COM 送信データアボート関数
FncWrNMMsgDRV (UINT32, UINT8, UINT8 *)	任意	NM 送信データ受け渡し関数
FncGetNMSndStsDRV(UINT8 *)	任意	NM ステータス受け渡し関数
FncAbtNMSndMsgDRV(void)	NMLimpHome 状態遷移時	NM 送信データアボート関数

インタフェース名	実行タイミング	概要
FncAbtAllSndMsgDRV(void)	NMTwbsLimpHome 状態遷移時	全送信データアボート関数
FncEnCANSndInt(void)	任意	CAN 送信割込み有効関数
FncDisCANSndInt(void)	任意	CAN 送信割込み無効関数
FncEnCANWkupInt(void)	任意	CAN ウェイクアップ割込み有効関数
FncDisCANWkupInt(void)	任意	CAN ウェイクアップ割込み無効関数
FncEnCANErrInt(void)	任意	CAN エラー割込み有効関数
FncDisCANErrInt(void)	任意	CAN エラー割込み無効関数
FncGetBusoffStsDRV (UINT8 *)	任意	CAN バスエラー状態通知処理 (DRV モジュール内部の状態)
FncClrBusOffNoticeDRV(void)	任意	CAN DRV 内部バスオフ情報クリア関数

（１）CAN 送信処理関数

Service Name:	FncSetCANIntLVLDRV
Syntax:	StatusType FncSetCANIntLVLDRV()
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN 割込みレベルの設定を行う。
Return:	StatusType　CAN_E_OK　成功

(例)

```
/* アプリケーションの初期化処理の前に呼び出す*/
FncSetCANIntLVLDRV ();
AppInit();
InitCOM();
InitNM();
```

（２）CAN エラー割り込み処理関数

Service Name:	FncCANErrIntDRV
Syntax:	void FncCANErrIntDRV (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	ドライバでの CAN エラー割り込み処理を実施する。
Return:	なし

(例)

```
/* アプリケーションソフトの CAN エラー割り込み処理内 */
#pragma INTERRUPT CANErrInterrupt       注)割り込み関数名はユーザ定義
void CANErrInterrupt (void);
void CANErrInterrupt (void)
{
        FncCANErrIntDRV ();
                :       /* ユーザ割り込み処理を記述 */
                :
}
```

（３）CAN 送信完了割り込み処理関数

Service Name:	FncCANSndIntDRV
Syntax:	void FncCANSndIntDRV (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	ドライバでの CAN メッセージ送信完了処理を実施する。
Return:	なし

(例)

```
/* アプリケーションソフトの送信完了割り込み処理内 */
#pragma INTERRUPT CANSndInterrupt       注)割り込み関数名はユーザ定義
```

```
void CANSndInterrupt (void);
void CANSndInterrupt (void)
{
        FncCANSndIntDRV ();
                :       /* ユーザ割り込み処理を記述 */
                :
}
```

（４）CAN 受信完了割り込み処理関数

Service Name:	FncCANRcvIntDRV
Syntax:	void FncCANRcvIntDRV (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	ドライバでの CAN メッセージ受信完了処理を実施する。
Return:	なし

(例)

```
/* アプリケーションソフトの Sleep 後処理内 */
#pragma INTERRUPT CANRcvInterrupt  注)割り込み関数名はユーザ定義
void CANRcvInterrupt (void);
void CANRcvInterrupt (void)
{
        FncCANRcvIntDRV ();
                :       /* ユーザ割り込み処理を記述 */
                :
}
```

（５）CAN 動作モード変更関数

Service Name:	FncChgModCAN
Syntax:	StatusType FncChgModCAN(UINT8)
Parameter (Input):	UINT8： モード切替要求 CAN_DRV_CAN_RUN　CAN 動作モード CAN_DRV_CAN_RESET　CAN Reset モード CAN_DRV_CAN_SLEEP　CAN スリープモード
Parameter (Output):	なし
Description:	CAN コントローラの動作モードを変更する。 動作モード移行時、CAN コントローラの初期化を行う。 動作モードから再度 CAN コントローラを初期化する場合は一度 Reset モードに移行させてから動作モードに移行して下さい。 スリープモードを使用しない場合は、スリープモード未使用（CAN_DRV_SLEEPMODE を CAN_FALSE）にして下さい。 また、FncChgModCAN()は、CAN 関連の割り込み有効無効設定は行っていませんので、本関数を使用する際は、アプリケーション側で CAN 関連の割り込み有無の設定を行ってください。
Return:	StatusType　CAN_E_OK　正常 StatusType　CAN_E_NG　異常 StatusType　CAN_E_PARAM　引数異常

※CAN ドライバモジュールの初期化処理において、FncCANInitDRV()なしに、FncChgModCAN(CAN_DRV_CAN_RUN)を行っても問題ありません。(FncChgModCAN()処理内で、CAN コントローラ初期化を行っているため)

（６）CAN コントローラ初期化関数

Service Name:	FncCANInitDRV
Syntax:	StatusType　FncCANInitDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN コントローラの初期化、リセットモードへ移行を行う。
Return:	StatusType　CAN_E_OK　CAN コントローラ初期化成功 StatusType　CAN_E_NG　CAN コントローラ初期化失敗

（７）CAN コントローラリセット関数

Service Name:	FncCANResetDRV
Syntax:	StatusType　FncCANResetDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	リセットモードへ移行、及び CAN 関連割り込みの設定を行う。
Return:	StatusType　CAN_E_OK　CAN コントローラリセット成功 StatusType　CAN_E_NG　CAN コントローラリセット失敗

（８）CAN コントローラ動作関数

Service Name:	FncCANRunDRV
Syntax:	StatusType FncCANRunDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	動作モードへ移行、及び CAN 関連割り込みの設定を行う。
Return:	StatusType CAN_E_OK CAN コントローラリセット成功 StatusType CAN_E_NG CAN コントローラリセット失敗

（９）CAN コントローラスリープ関数

Service Name:	FncCANSleepDRV
Syntax:	StatusType FncCANSleepDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	スリープモードへ移行、及び CAN 関連割り込みの設定を行う。 スリープモード使用時のみ。
Return:	StatusType CAN_E_OK CAN コントローラリセット成功 StatusType CAN_E_NG CAN コントローラリセット失敗

（１０）COM 受信データ受け渡し関数

Service Name:	FncRdCOMMsgDRV
Syntax:	StatusType FncRdCOMMsgDRV(UINT32 *, UINT8 *, UINT8 *)
Parameter (Input):	なし
Parameter (Output):	UINT32 * : CAN-ID 格納変数へのアドレス UINT8 * : DLC 格納変数へのアドレス UINT8 * : メッセージ格納バッファ先頭へのアドレス
Description:	COM へ受信したデータを引渡す。 メッセージ格納バッファは 8 バイト以上確保すること。 COM ポーリング受信使用時のみ。
Return:	StatusType CAN_E_OK 引渡し完了 StatusType CAN_E_NG 受信メッセージなし

（１１）CAN 受信割込み有効関数

Service Name:	FncEnCANRcvInt
Syntax:	StatusType FncEnCANRcvInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	受信完了割り込みを有効にする。 受信割込み使用時のみ。
Return:	StatusType CAN_E_OK 成功

（１２）CAN 受信割込み無効関数

Service Name:	FncDisCANRcvInt
Syntax:	StatusType FncDisCANRcvInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	受信完了割り込みを無効にする。
Return:	StatusType　CAN_E_OK　　成功

（１３）COM 送信データ受け渡し関数

Service Name:	FncWrCOMMsgDRV
Syntax:	StatusType FncWrCOMMsgDRV(UINT32, UINT8, UINT8 *)
Parameter (Input):	UINT32　：　CAN-ID UINT8　：　DLC UINT8 ＊　：　メッセージ格納バッファ先頭へのアドレス
Parameter (Output):	なし
Description:	COM から送信データを受け渡す。 メッセージ格納バッファは 8 バイト以上確保すること。 COM 送信使用時のみ。
Return:	StatusType　CAN_E_OK　　成功 StatusType　CAN_E_NG　　失敗

（１４）COM ステータス受け渡し関数

Service Name:	FncGetCOMSndStsDRV
Syntax:	void FncGetCOMSndStsDRV(UINT8 *)
Parameter (Input):	なし
Parameter (Output):	UINT8 * : 送信状態 CAN_E_DRV_EMPTY 未送信状態 CAN_E_DRV_WAIT 送信中 CAN_E_DRV_COMP 送信完了
Description:	COM の送信状態を取得する。 COM 送信使用時のみ。
Return:	なし

（１５）COM 送信データアボート関数

Service Name:	FncAbtCOMSndMsgDRV
Syntax:	StatusType FncAbtCOMSndMsgDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	COM の送信フレームを停止する。 COM 送信使用時のみ。
Return:	StatusType CAN_E_OK 成功 StatusType CAN_E_NG 失敗

（１６）NM 送信データ受け渡し関数

Service Name:	FncWrNMMsgDRV
Syntax:	StatusType FncWrNMMsgDRV(UINT32, UINT8, UINT8 *)
Parameter (Input):	UINT32 : CAN-ID UINT8 : DLC UINT8 * : メッセージ格納バッファ先頭へのアドレス
Parameter (Output):	なし
Description:	NM から送信データを受け渡す。 メッセージ格納バッファは 8 バイト以上確保すること。 NM 送信使用時のみ。
Return:	StatusType CAN_E_OK 成功 StatusType CAN_E_NG 失敗

（１７）NM ステータス受け渡し関数

Service Name:	FncGetNMSndStsDRV
Syntax:	void FncGetNMSndStsDRV(UINT8 *)
Parameter (Input):	なし
Parameter (Output):	UINT8 * : 送信状態 CAN_E_DRV_EMPTY 未送信状態 CAN_E_DRV_WAIT 送信中 CAN_E_DRV_COMP 送信完了
Description:	NM の送信状態を取得する。 NM 送信使用時のみ。
Return:	なし

（１８）NM 送信データアボート関数

Service Name:	FncAbtNMSndMsgDRV
Syntax:	StatusType FncAbtNMSndMsgDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	NM の送信フレームを停止する。 NM 送信使用時のみ。
Return:	StatusType　CAN_E_OK　成功 StatusType　CAN_E_NG　失敗

(例)

```
/* LimpHome 状態移行時 */
FncAbtNMSndMsgDRV();
```

（１９）全送信データアボート関数

Service Name:	FncAbtAllSndMsgDRV
Syntax:	StatusType FncAbtAllSndMsgDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	COM,NM の送信フレームを停止する。 COM または NM 送信使用時のみ。
Return:	StatusType　CAN_E_OK　成功 StatusType　CAN_E_NG　失敗

(例)

```
/* NMTwbsLimpHome 状態移行時 */
FncAbtAllSndMsgDRV();
```

（２０）CAN 送信割込み有効関数

Service Name:	FncEnCANSndInt
Syntax:	StatusType FncEnCANSndInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	送信完了割り込みを有効にする。
Return:	StatusType　CAN_E_OK　　成功

（２１）CAN 送信割込み無効関数

Service Name:	FncDisCANSndInt
Syntax:	StatusType FncDisCANSndInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	送信完了割り込みを無効にする。
Return:	StatusType　CAN_E_OK　　成功

（２２）CAN ウェイクアップ割込み有効関数

Service Name:	FncEnCANWkupInt
Syntax:	StatusType FncEnCANWkupInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN ウェイクアップ割り込みを有効にする。 スリープモード使用時のみ
Return:	StatusType　CAN_E_OK　　成功

（２３）CAN ウェイクアップ割込み無効関数

Service Name:	FncDisCANWkupInt
Syntax:	StatusType FncDisCANWkupInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN ウェイクアップ割り込みを無効にする。
Return:	StatusType　CAN_E_OK　　成功

（２４）CAN エラー割込み有効関数

Service Name:	FncEnCANErrInt
Syntax:	StatusType FncEnCANErrInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN エラー割り込みを有効にする。
Return:	StatusType　CAN_E_OK　　成功

（２５）CAN エラー割込み無効関数

Service Name:	FncDisCANErrInt
Syntax:	StatusType FncDisCANErrInt(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	CAN エラー割り込みを無効にする。
Return:	StatusType　CAN_E_OK　　成功

（２６）CAN バスエラー状態通知処理関数

Service Name:	FncGetBusoffStsDRV
Syntax:	void FncGetBusoffStsDRV (UINT8 *)
Parameter (Input):	なし
Parameter (Output):	UINT8 * ：バスオフ状態 CAN_E_DRV_BUSOFF　バスオフ検出 CAN_E_DRV_NORMAL　バスオフ未検出
Description:	ドライバ内部のバスエラー状態を通知する。
Return:	なし

（２７）CAN DRV 内部バスオフ情報クリア関数

Service Name:	FncClrBusOffNoticeDRV
Syntax:	StatusType FncClrBusOffNoticeDRV (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	ドライバ内部のバスエラーフラグをクリアする。
Return:	StatusType CAN_E_OK　成功

５．４ 外部 API 関数

Driver 単体で使用するときのみ設定が必要な関数で、ユーザはこれに合わせ関数を作成する必要がある。

作成する関数は can_drv_def.h の宣言に依存する。

インタフェース名	実行タイミング	概要
CallNMRcvMsgDRV (UINT32, UINT8, UINT8 *)	NM スロットの受信割込み時	NM スロットの受信割込みコールバック処理はユーザ定義
CallCOMRcvMsgDRV (UINT32, UINT8, UINT8 *)	COM スロットの受信割込み時	COM スロットの受信割込みコールバック処理はユーザ定義
CallNMSndMsgDRV(void)	NM スロットの送信割込み時	NM スロットの送信割込みコールバック処理はユーザ定義
CallCOMSndMsgDRV(void)	COM スロットの送信割込み時	COM スロットの送信割込みコールバック処理はユーザ定義
CallErrIntDRV(void)	バスオフ検出時	バスオフ発生時コールバック

インタフェース名	定数	値
CallNMRcvMsgDRV(UINT32, UINT8, UINT8 *)	CAN_DRV_NM_RCV	CAN_TRUE
CallCOMRcvMsgDRV(UINT32, UINT8, UINT8 *)	CAN_DRV_COM_RCV	CAN_TRUE
CallNMSndMsgDRV(void)	CAN_DRV_NM_SND	CAN_TRUE
CallCOMSndMsgDRV(void)	CAN_DRV_COM_SND	CAN_TRUE
CallErrIntDRV(void)	CAN_DRV_ERR_CALL	CAN_TRUE

“can_drv_def.h”での定数の定義

（１）受信完了割込みコールバック

Service Name:	CallNMRcvMsgDRV
Syntax:	CallNMRcvMsgDRV(UINT32, UINT8, UINT8 *)
Parameter (Input):	UINT32　: CAN-ID UINT8　: DLC UINT8*　: メッセージ格納バッファ先頭へのアドレス
Parameter (Output):	なし
Description:	NM スロット受信割込みコールバック 処理内容はユーザ作成コード内容に依存する。
Return:	なし

Service Name:	CallCOMRcvMsgDRV
Syntax:	CallCOMRcvMsgDRV(UINT32, UINT8, UINT8 *)
Parameter (Input):	UINT32　: CAN-ID UINT8　: DLC UINT8*　: メッセージ格納バッファ先頭へのアドレス
Parameter (Output):	なし
Description:	COM スロット受信割込みコールバック 処理内容はユーザ作成コード内容に依存する。
Return:	なし

（２）送信完了コールバック

Service Name:	CallNMSndMsgDRV
Syntax:	CallNMSndMsgDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	NM スロット送信割込みコールバック 処理内容はユーザ作成コード内容に依存する。
Return:	なし

Service Name:	CallCOMSndMsgDRV
Syntax:	CallCOMSndMsgDRV(void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	COM スロット送信割込みコールバック 処理内容はユーザ作成コード内容に依存する。
Return:	なし

（３）エラー通知コールバック

Service Name:	CallErrIntDRV
Syntax:	CallErrIntDRV (void)
Parameter (Input):	なし
Parameter (Output):	なし
Description:	バスオフ検出割込みコールバック 処理内容はユーザ作成コード内容に依存する。
Return:	なし

改訂履歴

Version	変更内容	日付	作成者
Ver.1.00	・　新規作成	2007/10/22	サニー技研