リファレンスマニュアル

ＣＡＮＭｓｔ　ＲＳＬ

目　次

第１章 関数一覧

第２章 関数仕様

第１章　関数一覧

１）マスタアクセス関数

関　数	機　　能
CANMst_Open( )	マスタをオープンします。
CANMst_Close( )	マスタをクローズします。
CANMst_SetCommSetting( )	マスタの通信パラメータを設定します。
CANMst_GetCommSetting( )	マスタの通信パラメータを取得します。
CANMst_SetAcceptanceFilter( )	CAN バス上に流れるデータのフィルタリング設定値の設定。
CANMst_GetAcceptanceFilter( )	CAN バス上に流れるデータのフィルタリング設定値の読み出し。
CANMst_StartComm( )	CAN バスへ接続。
CANMst_ChkStartComm( )	CAN バスの接続を確認します。
CANMst_ResetComm( )	CAN バスから切断。
CANMst_ChkResetComm( )	CAN バスの切断を確認します。
CANMst_GetMasterStatus( )	マスタの通信状態を取得します。
CANMst_GetErrorEvent( )	CAN バス上のエラー情報を取得。
CANMst_SendMessage( )	CAN メッセージの送信。
CANMst_RecvMessage( )	CAN メッセージの受信。
CANMst_ClearBuffer( )	指定したバッファをクリアする。

１）スレーブアクセス関数

関　数	機　　能
CANSlv_DIN32Read	DI032 スレーブの IN データを読み出し。
CANSlv_DOUT32Write	DI032 スレーブの OUT データを書き込み。
CANSlv_DOUT32Read	DI032 スレーブの OUT データを読み出し。

第２章　関数仕様

２－１　ライブラリ使用方法

ライブラリを使用したアプリケーション開始のフローチャートを以下に示します。

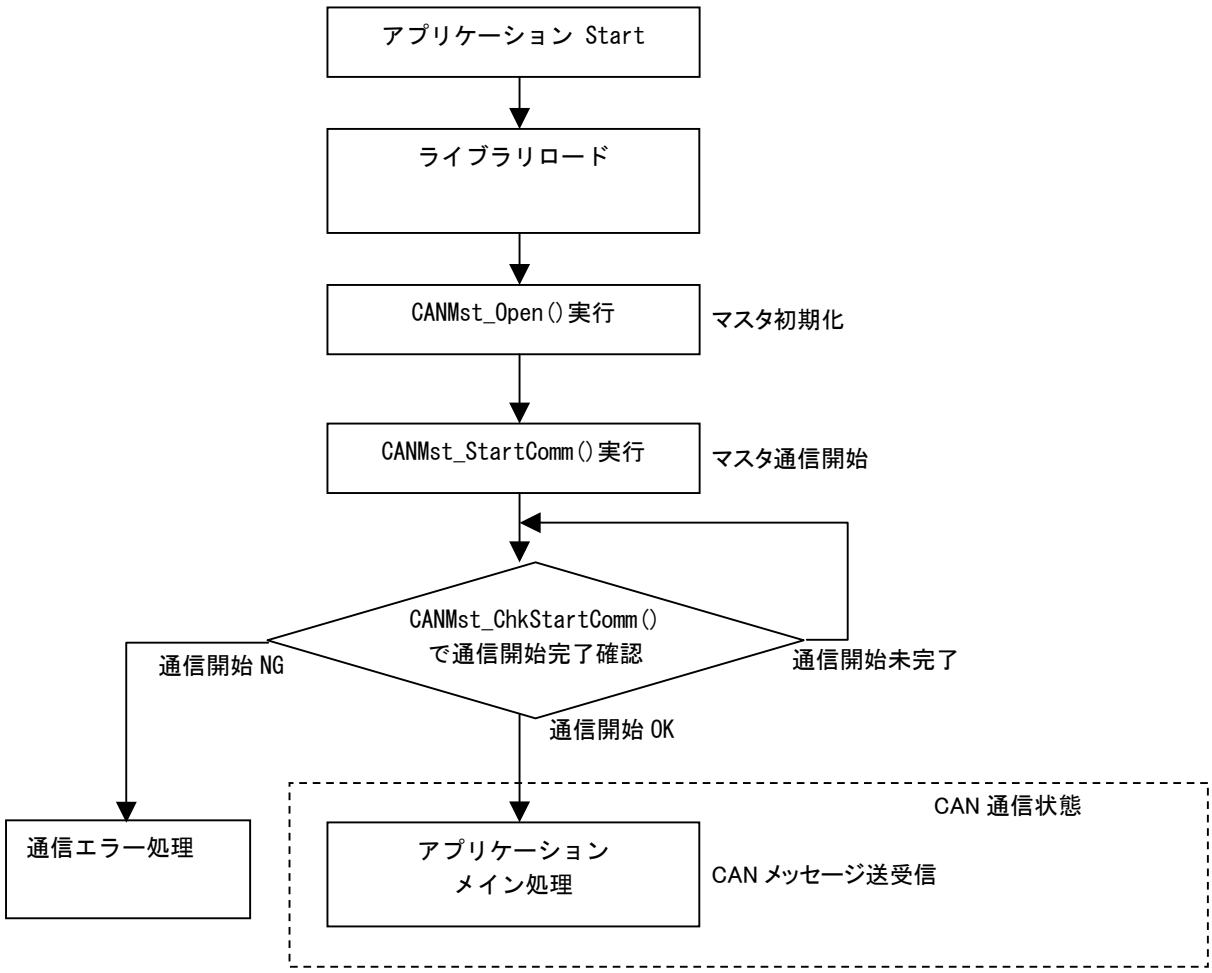

図 2-1-1. フローチャート

ライブラリロード後、マスタ初期化、マスタ通信開始を行うことで CAN バスへアクセス可能となります。通信が正常に開始されれば、CAN メッセージの送受信が可能になります。

２－２　マスタアクセス関数

CANMst_Open 関数

機能　CAN マスタをオープンします。

書式　int CANMst_Open (void);

引数　なし

戻り値

CN_ER_OK	: OK
CN_ER_NOMSTPROC	: マスタプロセスが起動されていない
CN_ER_ALREADYOPEN	: すでにオープンされている
CN_ER_CREATE	: リソース生成失敗

説明　CAN マスタをオープンします。
本関数をコール後、CAN マスタにアクセス可能となります。本ライブラリを使用して制御を行う前に、必ずコールする必要があります。

CANMst_Close 関数

機能　CAN マスタをクローズします。

書式　int CANMst_Close (void) ;

引数　なし

戻り値

CN_ER_OK	: 正常
CN_ER_NOTOPEN	: オープンされていない

説明　リソースを開放し、CAN マスタとの接続を切断します。
再度、CAN マスタへのアクセスを行う場合には CANMst_Open を実行する必要があります。

CANMst_SetCommSetting 関数

機能　CAN 通信パラメータを設定します。

書式

```
int CANMst_SetCommSetting(
    TCAN_CONFIG *Can_Config
);
```

引数　Can_Config　：通信設定を格納するポインタ

通信設定

```
typedef struct {
    unsigned char Mon;
    unsigned char AccFil;
    unsigned short SndBufSz;
    unsigned short RcvBufSz;
    unsigned short ErrBufSz;
    unsigned short ErrLimit;
    unsigned short Baudrate;
} TCAN_CONFIG;
```

Mon　：モニタモード設定　（初期値：0）
　[0：通常モード]
　[1：モニタモード]

AccFil　：アクセプタンスフィルタモード設定　（初期値：1）
　[0：Dual フィルタ]
　[1：Single フィルタ]

SndBufSz　：送信バッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【20Byte】

RcvBufSz　：受信バッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【20Byte】

ErrBufSz　：エラーバッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【3Byte】

ErrLimit　：エラーリミット数（SJA1000 の EWLR レジスタ設定値）　（初期値：96）
　[0 ～255]

Baudrate　　：通信ボーレート　　（初期値：0x1C05）

通信速度	設定値
10【Kbps】	0x6F31
20【Kbps】	0x6F18
50【Kbps】	0x1C0E
100【Kbps】	0x2F05
125【Kbps】	0x1C05
250【Kbps】	0x1C02
500【Kbps】	0x6F00
800【Kbps】	0x1B00
1【Mbps】	0x0900

上記の表以外の通信ボーレートを設定する場合は、下記の計算方法を使って設定値を計算します。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAM	TSEG2			TSEG1				SJW		BRP					

入力クロック = 24【MHz】= 24000000【Hz】
fscl = 入力クロック【Hz】/ ((BRP + 1) * 2)
ボーレート = fscl / (1 + (TSEG1 + 1) + (TSEG2 + 1))
SAM：サンプリング回数 0：1回 1：3回

戻り値

CN_ER_OK　　：OK
CN_ER_NOTOPEN　　：オープンされていない
CN_ER_INVALIDPARAM　　：パラメータ異常
CN_ER_ALREADYCOMM　　：すでに通信されている

説明

CAN インターフェースの設定を行います。
この設定は、CAN バスが切断されているときのみ実行できます。
CAN バスが接続されている場合、CN_ER_ALREADYCOMM が返ります。

CANMst_GetCommSetting 関数

機能　CAN インターフェースの各種設定を読み出します。

書式

```
int CANMst_GetCommSetting(
    TCAN_CONFIG *Can_Config
);
```

引数　Can_Config　：通信設定を格納するポインタ

通信設定

```
typedef struct {
    unsigned char Mon;
    unsigned char AccFil;
    unsigned short SndBufSz;
    unsigned short RcvBufSz;
    unsigned short ErrBufSz;
    unsigned short ErrLimit;
    unsigned short Baudrate;
} TCAN_CONFIG;
```

Mon　：モニタモード設定　（初期値：0）
　[0：通常モード]
　[1：モニタモード]

AccFil　：アクセプタンスフィルタモード設定　（初期値：1）
　[0：Dual フィルタ]
　[1：Single フィルタ]

SndBufSz　：送信バッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【20Byte】

RcvBufSz　：受信バッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【20Byte】

ErrBufSz　：エラーバッファサイズ　（初期値：64）
　[64 ～ 65536] 単位：【3Byte】

ErrLimit　：エラーリミット数（SJA1000 の EWLR レジスタ設定値）　（初期値：96）
　[0 ～255]

Baudrate　　：通信ボーレート　　（初期値：0x1C05）

通信速度	設定値
10【Kbps】	0x6F31
20【Kbps】	0x6F18
50【Kbps】	0x1C0E
100【Kbps】	0x2F05
125【Kbps】	0x1C05
250【Kbps】	0x1C02
500【Kbps】	0x6F00
800【Kbps】	0x1B00
1【Mbps】	0x0900

上記の表以外の通信ボーレートを設定する場合は、下記の計算方法を使って設定値を計算します。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAM	TSEG2			TSEG1				SJW		BRP					

入力クロック = 24【MHz】= 24000000【Hz】
fscl = 入力クロック【Hz】/ ((BRP + 1) * 2)
ボーレート = fscl / (1 + (TSEG1 + 1) + (TSEG2 + 1))
SAM：サンプリング回数 0：1回 1：3回

戻り値

CN_ER_OK　　：正常
CN_ER_NOTOPEN　　：オープンされていない

説明

CAN インターフェースの各種設定を読み出します。

CANMst_SetAcceptanceFilter 関数

機能　CAN バス上に流れるデータのフィルタリング設定値を設定します。

書式

```
int CANMst_SetAcceptanceFilter(
    unsigned long AcceptanceCode,
    unsigned long AcceptanceMask
);
```

引数

AcceptanceCode　：　ビットフィルタ値
AcceptanceMask　：　ビットマスク値

フィルタ設定

AcceptanceCode　：　ビットフィルタ値　（初期値：0x00000000）
AcceptanceMask　：　ビットマスク値　（初期値：0xFFFFFFFF）
[0x00000000 ～ 0xFFFFFFFF]

戻り値

CN_ER_OK　：OK
CN_ER_NOTOPEN　：オープンされていない
CN_ER_ALREADYCOMM　：すでに通信されている

説明　　受信メッセージのフィルタ設定を行います。
AcceptanceMask で 0 を指定された bit がフィルタの対象となります。
AcceptanceCode の値と一致したデータのみが、受信バッファに取り込まれます。

●通常フォーマット時のフィルタ適用範囲

ID(11bit)							RTR	1バイト目データ								2バイト目データ								3バイト目データ							
28	27	26	・・・	20	19	18	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
フィルタ適用範囲																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	RTR	リザーブ			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DB1:7	DB1:6	DB1:5	DB1:4	DB1:3	DB1:2	DB1:1	DB1:0	DB2:7	DB2:6	DB2:5	DB2:4	DB2:3	DB2:2	DB2:1	DB2:0

●拡張フォーマット時のフィルタ適用範囲

ID(29bit)																RTR	1バイト目データ							
28	27	26	25	・・・								3	2	1	0	0	7	6	5	4	3	2	1	0
フィルタ適用範囲																								

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR	リザーブ	

CANMst_ GetAcceptanceFilter 関数

機能　CAN バス上に流れるデータのフィルタリング設定値を読み出します。

書式

```
int CANMst_GetAcceptanceFilter(
    unsigned long *AcceptanceCode,
    unsigned long *AcceptanceMask
)
```

引数

AcceptanceCode	：	ビットフィルタ値を格納するポインタ
AcceptanceMask	：	ビットマスク値を格納するポインタ

戻り値

CN_ER_OK	：正常
CN_ER_NOTOPEN	：オープンされていない

説明　フィルタ設定値の詳細については、「CANMst_SetAcceptanceFilter」を参照してください。

CANMst_StartComm 関数

機能　CAN バス接続を開始します。

書式　int CANMst_StartComm (void);

引数　なし

戻り値

CN_ER_OK	：正常
CN_ER_NOTOPEN	：オープンされていない
CN_ER_NOMSTPROC	：マスタプロセスが動作していない
CN_ER_NOTRESET	：リセット処理が完了していない
CN_ER_ALREADYCOMM	：すでに通信されている
CN_ER_ERROR	：マスタプロセスへのアクセスに失敗した

説明　CAN バスの通信設定を確定し、通信開始します。
送信バッファ、受信バッファ、エラーバッファのサイズはこの関数実行時に決まります。
通信中は、通信設定およびフィルタ設定は実行することができません。
接続完了は、「CANMst_ChkStartComm」関数にて確認してください。

CANMst_ChkStartComm 関数

機能　CAN バス接続開始の確認を行います。

書式　int CANMst_ChkStartComm (void);

引数　なし

戻り値

CN_ER_OK	：正常
CN_ER_NOTOPEN	：オープンされていない
CN_ER_NOTCOMM	：通信開始未完了
CN_ER_COMMINITERR	：通信開始失敗

説明　「CANMst_StartComm」の接続状況を確認します。
「CN_ER_NOTCOMM」が返答される間は、通信接続処理中です。
「CN_ER_OK」が返答されれば、正常に通信接続できています。
戻り値で「CN_ER_COMMINITERR」が返ってきた場合は、「CANMst_GetMasterStatus」を使ってエラー情報を取得してください。

CANMst_ResetComm 関数

機能　CAN バス接続を切断します。

書式　int CANMst_ResetComm (void);

引数　なし

戻り値

CN_ER_OK	：正常
CN_ER_NOTOPEN	：オープンされていない
CN_ER_NOTCOMM	：未接続

説明　CAN バスを切断します。
通信が正常に停止されたかどうかは、「CANMst_ChkResetComm」関数で確認してください。

CANMst_ChkResetComm 関数

機能　CAN バス切断の確認を行います。

書式　int CANMst_ChkResetComm (void);

引数　なし

戻り値

CN_ER_OK	：正常
CN_ER_NOTOPEN	：オープンされていない
CN_ER_NOTRESET	：通信停止処理中

説明　「CANMst_ResetComm」の接続状況を確認します。
CN_ER_NOTRESET が返答される間は、通信切断処理中です。
正常に通信が切断された場合は、「CN_ER_OK」が返ります。

CANMst_GetMasterStatus 関数

機能　CAN バスのステータスを取得します。

書式

```
int CANMst_GetMasterStatus(
    TCAN_STATUS *canstat
);
```

引数　canstat　：マスタステータスを格納するポインタ

ステータス

```
typedef struct {
    unsigned short BusStat;
    unsigned short ErrCode;
    unsigned short TxBufCnt;
    unsigned short RxBufCnt;
    unsigned short ErrBufCnt;
    unsigned char TxErrCnt;
    unsigned char RxErrCnt;
} TCAN_STATUS;
```

BusStat　：　CAN バスステータス

BusStat	内容
CANMSTPROC_STS_INIT	0: マスタプロセス初期状態
CANMSTPROC_STS_WAIT	1: マスタプロセス通信待機状態
CANMSTPROC_STS_ACT	2: マスタプロセス通信処理中
CANMSTPROC_STS_CONNECT	3: マスタプロセス通信状態
CANMSTPROC_STS_RACT	4: マスタプロセス通信リセット処理中

ErrCode　：　マスタエラーコード

ErrCode	内容
CANMSTPROC_ERR_OK	0: エラー無し
CANMSTPROC_ERR_NOTWAIT	1: マスタプロセス通信待機状態でない
CANMSTPROC_ERR_INIT	2: マスタプロセス通信初期化失敗
CANMSTPROC_ERR_NOSEND	3: 送信失敗
CANMSTPROC_ERR_NOSENDMSG	4: 送信メッセージがない
CANMSTPROC_ERR_COMMFAIL	5: 通信異常

TxBufCnt　：　送信バッファにある未送信 CAN メッセージ数

RxBufCnt　：　受信バッファにある受信済み CAN メッセージ数

ErrBufCnt　：　エラーバッファにあるエラー情報数

TxErrCnt　：　送信エラーカウンタ値

RxErrCnt　：　受信エラーカウンタ値

戻り値　CN_ER_OK　：正常
CN_ER_NOTOPEN　：オープンされていない

説明　CAN バスのステータスを取得します。

CANMst_GetErrorEvent 関数

機能　エラーバッファのエラー情報を取得します。

書式

```
int CANMst_GetErrorEvent(
    TCAN_ERRBUF *errbuf,
    unsigned short errbufcnt,
    unsigned short *errgetcnt
);
```

引数

errbuf　：　エラー情報を格納するバッファのポインタ
errbufcnt　：　エラー情報を格納するバッファのサイズ
errgetcnt　：　取得したエラー情報の数を格納するポインタ

エラーバッファ

```
typedef struct {
    unsigned char IntErr;
    unsigned char ErrCode;
    unsigned char ArbLstCap;
} TCAN_ERRBUF;
```

IntErr　：　エラー割り込み発生要因

IntErr	内容
CAN_ERROR_BUSOFF_INT	0x80: CAN バス上のエラーを検出しました。
CAN_ARBITR_LOST_INT	0x40: CAN バス上でアービトレーション・ロストを検出しました。
CAN_ERROR_PASSIVE_INT	0x20: エラーパッシブ状態となりました。
CAN_OVERRUN_INT	0x08: オーバーランエラーが発生し、受信メッセージが失われました。
CAN_ERROR_INT	0x04: 送信か受信のエラーカウンタがリミット値を越えました。

ErrCode　：　エラーコード（SJA1000 ECC レジスタの値）
詳細は「２－４　エラーコード一覧」を参照してください。

ArbLstCap　：　アービトレーション・ロスト(SJA1000 ALC レジスタの値)
詳細は「２－４　エラーコード一覧」を参照してください。

戻り値

CN_ER_OK　：正常
CN_ER_NOTOPEN　：オープンされていない

説明

エラーバッファに格納されているエラー情報を取得します。
エラー情報は TCAN_ERRBUF 毎に格納されています。
TCAN_ERRBUF を格納できる数を errbufcnt に指定します。
格納した数が errgetcnt に格納されます。

CANMst_SendMessage 関数

機能　CAN メッセージの送信を行います。

書式

```
int CANMst_SendMessage(
    TCAN_MESSAGE *sndmsg,
    unsigned short sndcnt
);
```

引数

*sndmsg　：　送信メッセージを格納するポインタ
sndcnt　：　送信メッセージ数

送受信バッファ

```
typedef struct {
    unsigned long Flags;
    unsigned long Id;
    unsigned long Length;
    unsigned char Data[8];
} TCAN_MESSAGE;
```

Flags　：　送信メッセージに設定するフラグ
　　受信メッセージに設定されたフラグ

Flags	内容
MSG_FLAGS_RTR	0x01：0：通常フレーム 1：リモートフレーム
MSG_FLAGS_EXT	0x02：0：ID 11bit　1：ID 29bit

Id　：　送信メッセージに付加する ID、受信したメッセージの ID

Length　：　送受信メッセージの長さ（ 0 ～ 8 ）

Data　：　送受信メッセージデータ

戻り値

CN_ER_OK　：正常
CN_ER_NOTOPEN　：オープンされていない
CN_ER_NOTCOMM　：通信開始されていない

説明

CAN メッセージの送信を行います。
指定された数だけ、送信バッファに格納します。
格納されたメッセージは順次送信されます。

CANMst_RecvMessage 関数

機能　CAN メッセージの受信を行います。

書式

```
int CANMst_RecvMessage(
    TCAN_MESSAGE *rcvmsg,
    unsigned short rcvbufcnt,
    unsigned short *getrcvcnt
);
```

引数

*rcvmsg　：　受信メッセージを格納するポインタ
rcvbufcnt　：　受信バッファカウント
*getrcvcnt　：　取得した受信メッセージ数格納ポインタ

送受信バッファ

```
typedef struct {
    unsigned long Flags;
    unsigned long Id;
    unsigned long Length;
    unsigned char Data[8];
} TCAN_MESSAGE;
```

Flags　：　送信メッセージに設定するフラグ
　　受信メッセージに設定されたフラグ

Flags	内容
MSG_FLAGS_RTR	0x01：0：通常フレーム 1：リモートフレーム
MSG_FLAGS_EXT	0x02：0：ID 11bit　1：ID 29bit

Id　：　送信メッセージに付加する ID、受信したメッセージの ID

Length　：　送受信メッセージの長さ　[0 ～ 8]

Data　：　送受信メッセージデータ

戻り値

CN_ER_OK　：正常
CN_ER_NOTOPEN　：オープンされていない
CN_ER_NOTCOMM　：通信開始されていない

説明

受信バッファに格納されている受信メッセージを取得します。
受信メッセージは TCAN_MESSAGE 毎に格納されています。
TCAN_MESSAGE を格納できる数を rcvbufcnt に指定します。
格納した数が getrcvcnt に格納されます。
受信メッセージが無い場合は getrcvcnt には 0 が格納されます。

CANMst_ClearBuffer 関数

機能　指定したバッファまたはカウント値をクリアします。

書式　int CANMst_ClearBuffer (unsigned char bufsel);

引数　bufsel　：クリアするバッファまたはカウント値を選択

bufsel	内容
CLR_SNDBUF	0x01：送信バッファクリア
CLR_RCVBUF	0x02：受信バッファクリア
CLR_SNDERRCNT	0x04：送信エラーカウンタクリア
CLR_RCVERRCNT	0x08：受信エラーカウンタクリア
CLR_ERRBUF	0x10：エラーバッファクリア

戻り値
CN_ER_OK　：正常
CN_ER_NOTOPEN　：オープンされていない
CN_ER_NOTRESET　：通信停止処理中

説明　指定したバッファまたはカウント値をクリアします。

２－３　スレーブアクセス関数

CANSlv_DIN32Read 関数

機能　DIO32 モジュールの IN32 点の状態を読み出します。

書式
```
int CANSlv_DIN32Read(
    unsigned long id,
    unsigned long *indata
)
```

引数

id	：	スレーブ ID [1 ～ 15]
indata	：	IN データ格納ポインタ

戻り値

CN_ER_OK	：OK
CN_ER_NOTOPEN	：マスタプロセスが起動されていない
CN_ER_NOTCOMM	：通信していない

説明　指定された DIO32 モジュールの IN32 点の状態を読み出します。

CANSlv_DOUT32Write 関数

機能　DIO32 モジュールの OUT32 点の状態を変更します。

書式

```
int CANSlv_DOUT32Write(
    unsigned long id,
    unsigned long outdata
)
```

引数

id	：	スレーブ ID [1 ～ 15]
outdata	：	出力する OUT データ

戻り値

CN_ER_OK	：OK
CN_ER_NOTOPEN	：マスタプロセスが起動されていない
CN_ER_NOTCOMM	：通信していない

説明　指定された DIO32 モジュールの OUT32 点の状態を変更します。

CANSlv_DOUT32Read 関数

機能　DIO32 モジュールの OUT32 点の状態を読み出します。

書式

```
int CANSlv_DOUT32Read(
    unsigned long id,
    unsigned long *outdata
)
```

引数

id	:	スレーブ ID [1 ～ 15]
outdata	:	OUT データ格納ポインタ

戻り値

CN_ER_OK	: OK
CN_ER_NOTOPEN	: マスタプロセスが起動されていない
CN_ER_NOTCOMM	: 通信していない

説明　指定された DIO32 モジュールの OUT32 点の状態を読み出します。

２－４　エラーコード一覧

関数戻り値エラーコード一覧

表 2-4-1. 関数戻り値エラーコード一覧

エラーコード定義名	エラーコード	内容
CN_ER_OK	0x0000	正常です。
CN_ER_ALREADYOPEN	0x0001	すでにオープンしています。
CN_ER_NOMSTPROC	0x0002	CAN マスタプロセスが起動していません。
CN_ER_INVALIDPARAM	0x0003	無効な引数です。
CN_ER_NOTOPEN	0x0004	オープンしていません。
CN_ER_ALREADYCOMM	0x0005	すでに通信開始されています。
CN_ER_NOTCOMM	0x0006	通信していません。
CN_ER_NOTRESET	0x0007	リセットされていません。
CN_ER_COMMINITERR	0x0009	通信初期化エラーです。
CN_ER_ERROR	0x0101	内部エラーです。
CN_ER_CREATE	0x0102	各種デバイス生成失敗。
CN_ER_TIMEOUT	0x0103	タイムアウトエラーです。
CN_ER_LOADDEVICE	0x0201	デバイスドライバロードエラーです。

SJA1000 Error Code Capture Register (ECC)

表 2-4-2. ECCレジスタ内容

BIT	名称	内容
D6-D7	ERRC	エラー種別 00：ビットエラー 01：フォームエラー 10：スタッフエラー 11：その他のエラー
D5	DIR	エラー発生方向 0：送信エラー 1：受信エラー
D0-D4	SEG	エラー発生箇所 00011：SOF（スタートオブフレーム） 00010：ID.28～ID.21 の間 00110：ID.20～ID.18 の間 00100：SRTR（標準フォーマット時は RTR） 00101：IDE 00111：ID.17～ID.13 の間 01111：ID.12～ID.5 の間 01110：ID.4～ID.0 の間 01100：RTR 01101：R1（予約ビット） 01001：R0（予約ビット） 01011：DLC（データ長コード） 01010：データフィールド 01000：CRC シーケンス 11000：CRC デリミタ 11001：ACK スロット 11011：ACK デリミタ 11010：EOF（エンドオブフレーム） 10010：インターミッション 10001：アクティブエラーフラグ 10110：パッシブエラーフラグ 10011：重ね合わせエラーフラグ 10111：エラーデリミタ 11100：オーバーロードフラグ

SJA1000 Arbitration Lost Capture Register (ALC)

表 2-4-3. ALCレジスタ内容

BIT	名称	内容
D5-D7	Reserved	リザーブ
D0-D4	SEG	アービトレーション・ロスト発生位置 00000：ID の 1 ビット目 (※1) 00001：ID の 2 ビット目 00010：ID の 3 ビット目 00011：ID の 4 ビット目 00100：ID の 5 ビット目 00101：ID の 6 ビット目 00110：ID の 7 ビット目 00111：ID の 8 ビット目 01000：ID の 9 ビット目 01001：ID の 10 ビット目 01010：ID の 11 ビット目 01011：SRTR ビット (※2) 01100：IDE ビット 01101：ID の 12 ビット目 (※3) 01110：ID の 13 ビット目 (※3) 01111：ID の 14 ビット目 (※3) 10000：ID の 15 ビット目 (※3) 10001：ID の 16 ビット目 (※3) 10010：ID の 17 ビット目 (※3) 10011：ID の 18 ビット目 (※3) 10100：ID の 19 ビット目 (※3) 10101：ID の 20 ビット目 (※3) 10110：ID の 21 ビット目 (※3) 10111：ID の 22 ビット目 (※3) 11000：ID の 23 ビット目 (※3) 11001：ID の 24 ビット目 (※3) 11010：ID の 25 ビット目 (※3) 11011：ID の 26 ビット目 (※3) 11100：ID の 27 ビット目 (※3) 11101：ID の 28 ビット目 (※3) 11110：ID の 29 ビット目 (※3) 11111：RTR ビット (※3)

※1. ID の 1 ビット目は ID.28、2 ビット目は ID.27、…、となります。
※2. 標準フォーマットの場合、RTR ビット。
※3. 拡張フォーマットの場合のみ有効。

このリファレンスマニュアルについて

（1）本書の内容の一部または全部を当社からの事前の承諾を得ることなく、無断で複写、複製、掲載することは固くお断りします。
（2）本書の内容に関しては、製品改良のためお断りなく、仕様などを変更することがありますのでご了承下さい。
（3）本書の内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気付きのことがございましたらお手数ですが巻末記載の弊社もしくは、営業所までご連絡下さい。その際、巻末記載の書籍番号も併せてお知らせ下さい。

７６ＤＬＨ００５８Ｂ　２０１４年　１０月　第２版
７６ＤＬＨ００５８Ａ　２０１２年　１０月　初版

株式会社アルゴシステム

本社
〒５８７－００２１　大阪府堺市美原区小平尾６５６番地
ＴＥＬ（０７２）３６２－５０６７
ＦＡＸ（０７２）３６２－４８５６

ホームページ　http://www.algosystem.co.jp/