V016002-702

CANTOOL-A シリーズ

CANTOOL A1

[SDK-Manual]

Rev. 01.06 2022-08-31

製品名	型番	対応全体バージョン
CANTOOL A1	CTA101A-D0E0T CTA101A-D0E0	Ver01.05.00

アイテック阪急阪神 株式会社

改定履歴

Revision	日付	変更内容
01.00	2018-01-29	初版(対応全体バージョン：Ver01.02)
01.01	2018-03-26	対応全体バージョン：Ver01.03
01.02	2018-07-02	対応全体バージョン：Ver01.03.02 ・不具合対応
01.03	2018-08-01	対応全体バージョン：Ver01.03.04 ・API 追加 Rmt_SendMultiPacketData() Rmt_GetMultiPacketStatus()
01.04	2018-12-10	対応全体バージョン：Ver01.03.10 ・API 追加 Rmt_StartAppFunction2()
01.05	2020-01-20	対応全体バージョン：Ver01.04.00 ・API 追加 Rmt_SendMultiPacketData2() Rmt_ResponseMultiPacketData() Rmt_GetResponseMultiPacketDataStatus() Rmt_LoadMasterSetting2 Rmt_SendCyclicData() ・API 機能拡張 Rmt_SendMultiPacketData() Rmt_GetMultiPacketStatus()
01.06	2022-08-31	対応全体バージョン Ver01.05.00 ・API 追加 Rmt_DynamicPeriodicFrameControl() Rmt_SendCyclicData2() Rmt_SendMultiPacketData3() Rmt_ResponseMultiPacketData3() ・API 仕様変更 Rmt_SendCyclicData() 外部仕様参照

商標

Microsoft、Windows、Visual Studio、Visual Basic、Visual C#、Visual C++は、米国 Microsoft Corporation（あるいは米国マイクロソフト・コーポレーション）の米国およびその他の国における登録商標です。（Windows の正式名称は Microsoft Windows Operating System です）

その他、記載されている会社名、製品名は、各社の商標または登録商標です。

目次

1. 概要

CANTOOL A1 の SDK(Software Development Kit：ソフトウェア開発キット) は CANTOOL A1 が提供する機能です。
SDK は User Application(ユーザープログラム)から CANTOOL A1 の CAN、AD/DA、GPIO などを制御するためのソフトウェア開発キットです。
本 SDK 使用することで CANTOOL A1 が持つ CAN、LIN、AD/DA、GPIO などをユーザープログラムから制御することができます。

本書では、以降 CANTOOL A1 を"CANTOOL" と称します。
SDK は、C++版と C#版の DLL を提供しています。

本書は、「V016002-700_CANTOOL_A1-Manual」を参照し CANTOOL の基本的な利用方法をご理解されている方向けの記載となっています。

1.1 システム概要

CANTOOL および SDK のソフトウェア構成を以下に示します。

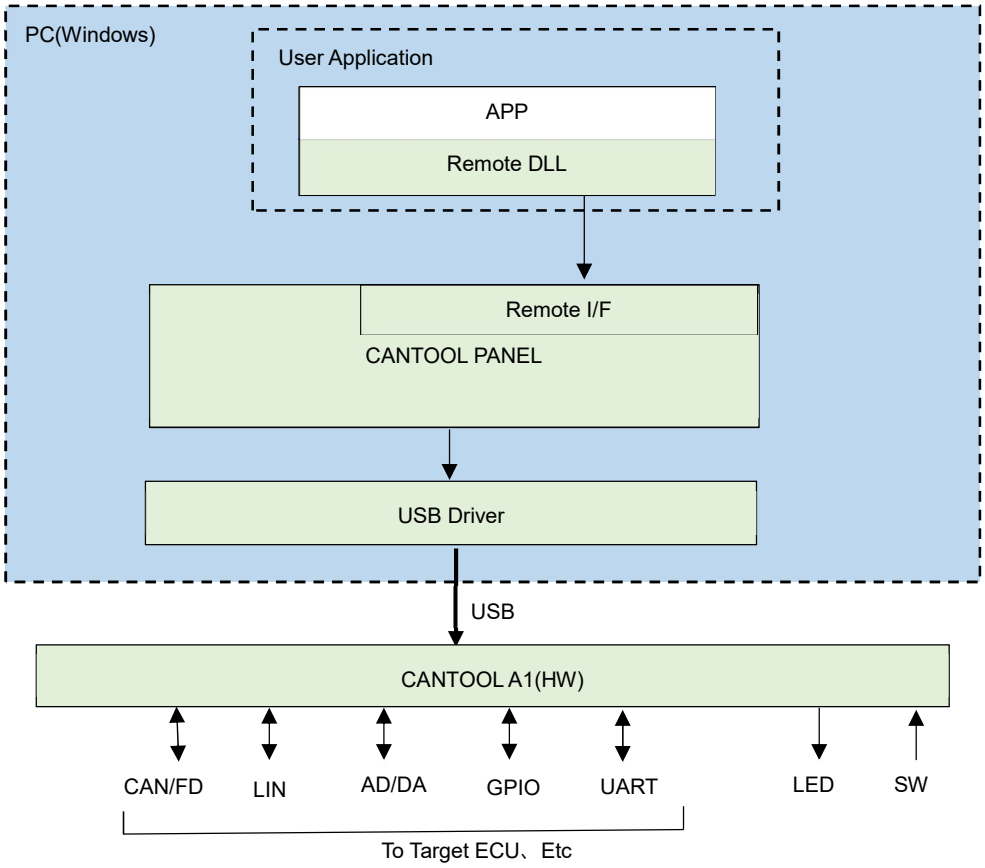


図 1-1 CANTOOL ソフトウェア構成図

表 1-1 ソフトウェア構成要素一覧

名称	説明
User Application	ユーザーにて作成されるアプリケーションです。ここでは DLL を含んだ環境を指します。 ユーザーアプリは、Visual Studio もしくは EXCEL VBA から使用されることを想定しています。
APP	ユーザーにて作成されるアプリケーション部分です。
Remote DLL	CANTOOL を制御するための API が含まれた DLL です。(SDK) VisualStudio2017 で作成されており、C++版。C#版の DLL を提供します。
CANTOOL PANEL	CANTOOL 基本ソフトウェアです。
Remote I/F	Remote Dll とのインターフェースです。CANTOOL PANEL に含まれます。
USB Driver	CANTOOL のハードウェアにアクセスするための USB ドライバです。

1.2 稼働条件

対象オペレーティングシステム

本 SDK は、以下の Microsoft Windows オペレーティングシステムに対応しています。

・Windows 8.1 32bit/64bit

・Windows 10 32bit/64bit

対象．NET Framework

本 SDK は、以下の Microsoft .NET Framework に対応しています。

・.NET Framework 4.6.2

動作確認済み開発環境

本 SDK は、以下の開発環境を使用した動作を確認済みです。

・Microsoft Visual Studio 2017

・Microsoft EXCEL 2013

1.3 付属資料

①ライブラリインターフェース仕様

V016002-702_CANTOOL_A1_SDK-ReferenceManual-API

2. SDK について

2.1 SDK の構成

SDK には、以下のものが含まれています。

なお、最新データについては、取扱説明書の「製品サポートページ」の章を参照して、入手してください。

SDK-root

|- Manual	SDK マニュアル （本書）
|- Library	ライブラリ
|- |- RemoteDLL.dll	C++/C#
|- |- RemoteDLL.lib	C++/C#用 LIB
|- |- RemoteDLLCs.dll	C#用 DLL
|- |- RemoteDLLvba.dll	EXECEL VBA 用 DLL
|- Header	ヘッダーファイル
|- |- RemoteDLL.h	C++/C#用
|- Sample	各種サンプルプログラム
|- CANTOOL A1_SDK_sample-list.pdf	各サンプルプログラムの概要を記載
|- readme_001X_sample.pdf	サンプル１ の説明
|- :	
|- 001P_sample	サンプル１（C++） ＊1
|- 001S_sample	サンプル１（C#） ＊1
|- 001E_sample	サンプル１（EXCEL） ＊1
:	

＊１：サンプルプログラム名について

XXXY_ZZZ

XXX：サンプル番号 001~

Y ：開発環境 P＝C++用、 S＝C#用、 E＝EXCEL VBA

ZZZ：サンプルプログラム名

＊Library、Heder のファイルは、実行環境に合わせてコピーして使用してください。

2.2 SDK で扱うインターフェース情報ついて

2.2.1 インターフェース構成について

CANTOOL のインターフェースには、「車載ネットワーク」、「汎用入出力」の２つがあります。

2.2.1.1. 車載ネットワーク

```
Bus
  |-Node
  |-      |-Frame
  |-      |       |-Signal
  |-      |       |-Signal
  |-      |       |-    :
  |-      |-Frame
  |-      |       |-Signal
  |-      |       |-    :
  |-      |-Frame
  |       |    :
  |-Node
  |    :
```

図 2-1 インターフェース(車載ネットワーク)構成ツリー

車載ネットワークのインターフェースは、Bus, Node, Frame, Signal の階層構造となっています。以下に各要素について示します。

表 2-1 インターフェース(車載ネットワーク)構成要素一覧

<table>
<tr><th>項目</th><th>説明</th></tr>
<tr><td>Bus</td><td>物理的なインターフェースを Bus(バス)と呼びます。
車載ネットワークには以下のバスが存在します。
<table>
<tr><th>Bus 名</th><th>説明</th></tr>
<tr><td>CAN0
CAN1
CAN2
CAN3</td><td>Control Area Network のインタフェース
CAN および、CAN FD(CAN with Flexible Data-Rate)に対応</td></tr>
<tr><td>LIN</td><td>LIN インタフェース</td></tr>
</table>
</td></tr>
<tr><td>Node</td><td>ECU 機器などの単位で Fame をグルーピングしたもの。
各 Bus 単位で作成でき、1 つの Bus 内で複数の Node を定義可能です。
なお、 1 つの Bus 内では、同一 Node 名は使用できません。(異なる Bus の場合は可能です)</td></tr>
<tr><td>Frame</td><td>CAN/LIN のフレームのこと。
送信、受信含めて定義します。
各 Bus 単位で作成でき、1 つの Bus 内で複数の Frame を定義可能です。
なお、 1 つの Bus 内では、同一 Frame 名は使用できません。(異なる Bus の場合は可能です)</td></tr>
<tr><td>Signal</td><td>CAN/LIN のフレーム内信号のこと。
各 Frame 単位で作成できます。
なお、1 つの Frame 内では、同一 Signal 名は使用できません。(異なる Frame の場合は可能です)</td></tr>
</table>

2.2.1.2. 汎用入出力

図 2-2 インターフェース(汎用入出力)構成ツリー

汎用入出力のインターフェースは、Bus, Port の階層構造となっています。以下に各要素について示します。

表 2-2 インターフェース(汎用入出力)構成要素一覧

<table>
<tr><th>項目</th><th>説明</th></tr>
<tr><td>Bus</td><td>物理的なインターフェースを Bus(バス)と呼びます。
汎用入出力には以下のバスが存在します。
<table>
<tr><th>Bus 名</th><th>説明</th></tr>
<tr><td>AD0
AD1</td><td>Analog/Digital Conversion インターフェース</td></tr>
<tr><td>DA0
DA1</td><td>Digital /Analog Conversion インターフェース</td></tr>
<tr><td>GPI</td><td>General Purpose Input インターフェース</td></tr>
<tr><td>GPO</td><td>General Purpose Output インターフェース</td></tr>
</table>
</td></tr>
<tr><td>Port</td><td>汎用入出力の Bus 内のポートのこと。
(GPI、GPO のみ)</td></tr>
</table>

2.2.2 SDK で使用するインターフェース設定情報

SDK 使用時は、「2.2.1 インターフェース構成について」をベースに、データを扱いやすくするために、各項目にユニークな ID を割り当てています。

2.2.2.1. 車載ネットワーク

車載ネットワークの各項目(Bus, Node, Frame, Signal)にそれぞれユニークな ID を割り当てています。

割り当てられた ID をそれぞれ BusID, NodeID, FrameID, SignalID と呼びます。

また、各 ID をまとめてリストにしたものをそれぞれ BusList, NodeList, FrameList, SignalList と呼びます。リスト内には当該階層以下の ID 情報を含んでいます。

ID とリストの構造の関係を以下に示します。

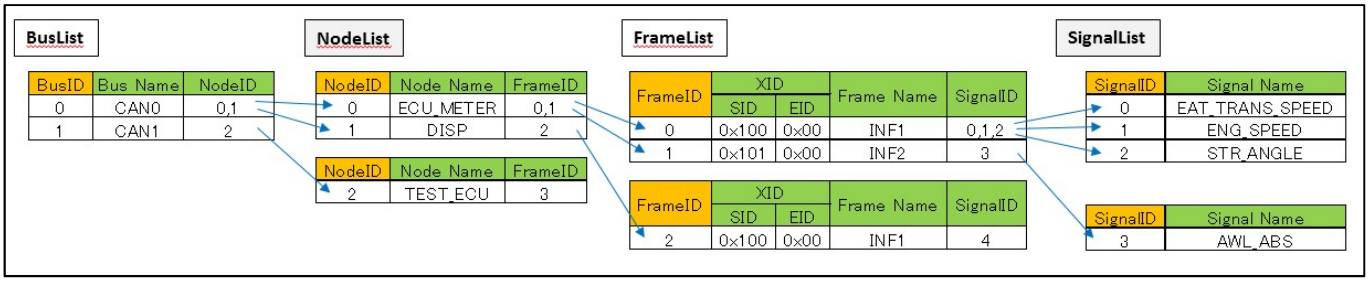


図 2-3 SDK 使用時のインターフェース(車載ネットワーク)構成

表 2-3 SDK 使用時のインターフェース(車載ネットワーク)構成情報一覧

項目			説明
BusList	BusID		Bus を一意に特定するための ID
	Bus Name		Bus 名称
	NodeId		BusID に紐づく Node 情報
NodeList	NodeID		Node を一意に特定するための ID
	Node Name		Node 名称
	FrameID		NodeID に紐づく Frame 情報
FrameList	FrameID		Frame を一意に特定するための ID
	Frame Name		Frame 名称
	XID	SID	標準 CAN ID
		EID	拡張 ID
	SignalID		FrameID に紐づく Signal 情報
SignalList	SignalID		Signal を一意に特定するための ID
	Signal Name		Signal 名称

2.2.2.2. 汎用入出力

汎用入出力のインターフェースは固定です。

以下に、汎用入出力の構成を示します。

表 2-4 SDK 使用時のインターフェース(汎用入出力)構成情報一覧

BusID	Port	説明
AD0 AD1	-	AD は 2 つの Bus で構成される
DA0 DA1	-	DA は 2 つの Bus で構成される
GPI	0~3	GPI は 1 つの Bus 内に 4 ポートある
GPO	0~3	GPO は 1 つの Bus 内に 4 ポートある

表 2-5 SDK 使用時のインターフェース(汎用入出力)構成情報一覧

項目	説明
BusID	Bus を一意に特定するための ID
Port	Bus 内の Port 番号(GPI、GPO のみ)

3. SDK 使用方法

3.1 SDK 使用準備

① PC アプリ（CANTOOL PANEL）のセットアップ

セットアップ方法については、取扱説明書の「ソフトウェアセットアップ」の章を参照してください。

② 環境変数の設定

SDK から PC アプリ(CANTOOL PANEL)に接続するために、環境変数の設定が必要です。以下の手順に沿って環境変数を設定してください。なお、本手順は、SDK を使用する PC 毎に必要です。

PC アプリ(CANTOOL-PANEL.exe が格納されている)フォルダ内にある「CANTOOL_A1_Setup.bat」を **管理者権限で実行**してください。

3.2 定義方法

3.2.1 DLL の参照設定

3.2.1.1. C++版

(1)Visual Studio のソリューションエクスプローラーにて、ユーザーアプリのプロジェクトを右クリック → [プロパティ] → [VC++ディレクトリ] → [ライブラリディレクトリ] を開いてください。

ライブラリディレクトリに、RemoteDLL.dll が格納されているディレクトリを指定してください。

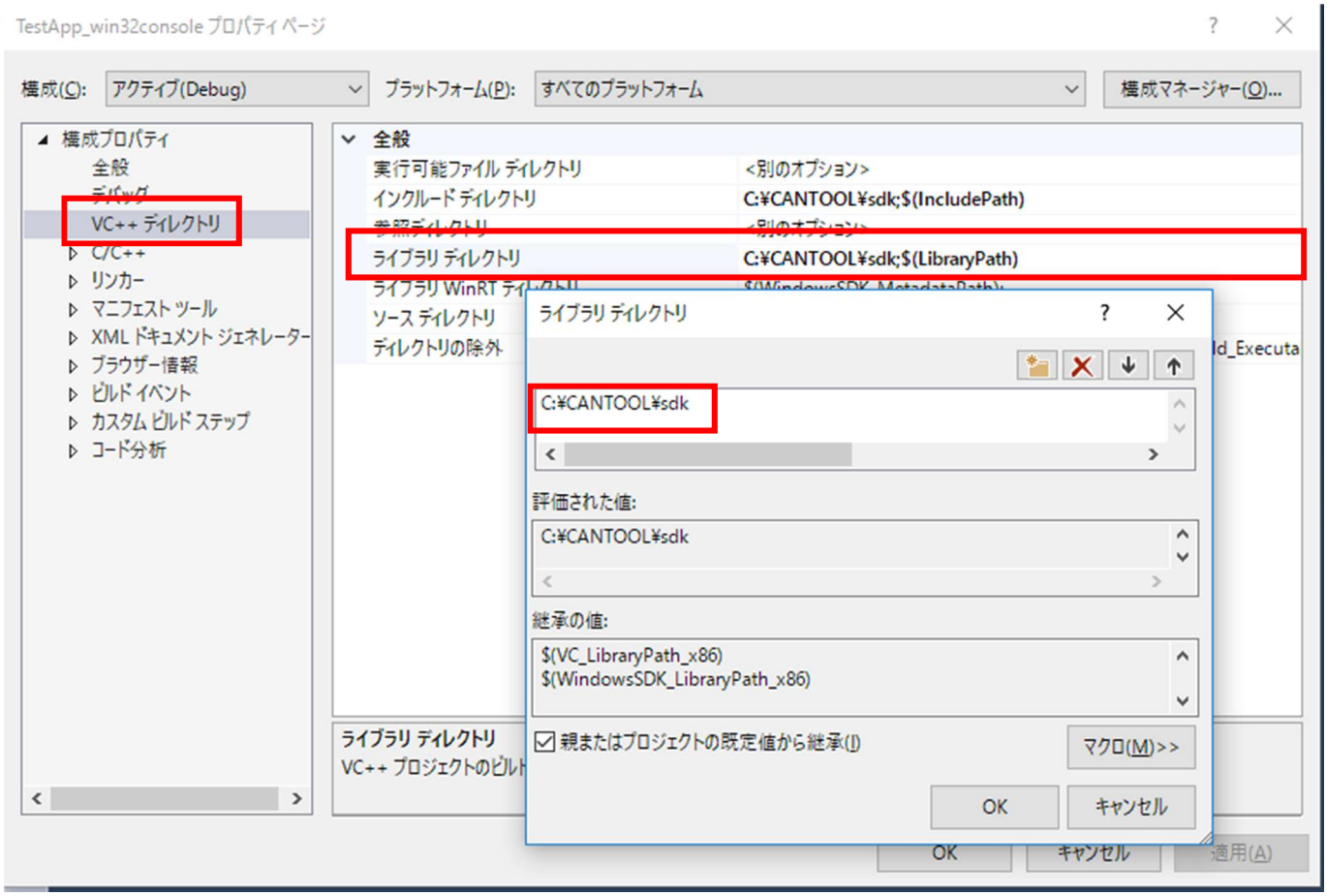
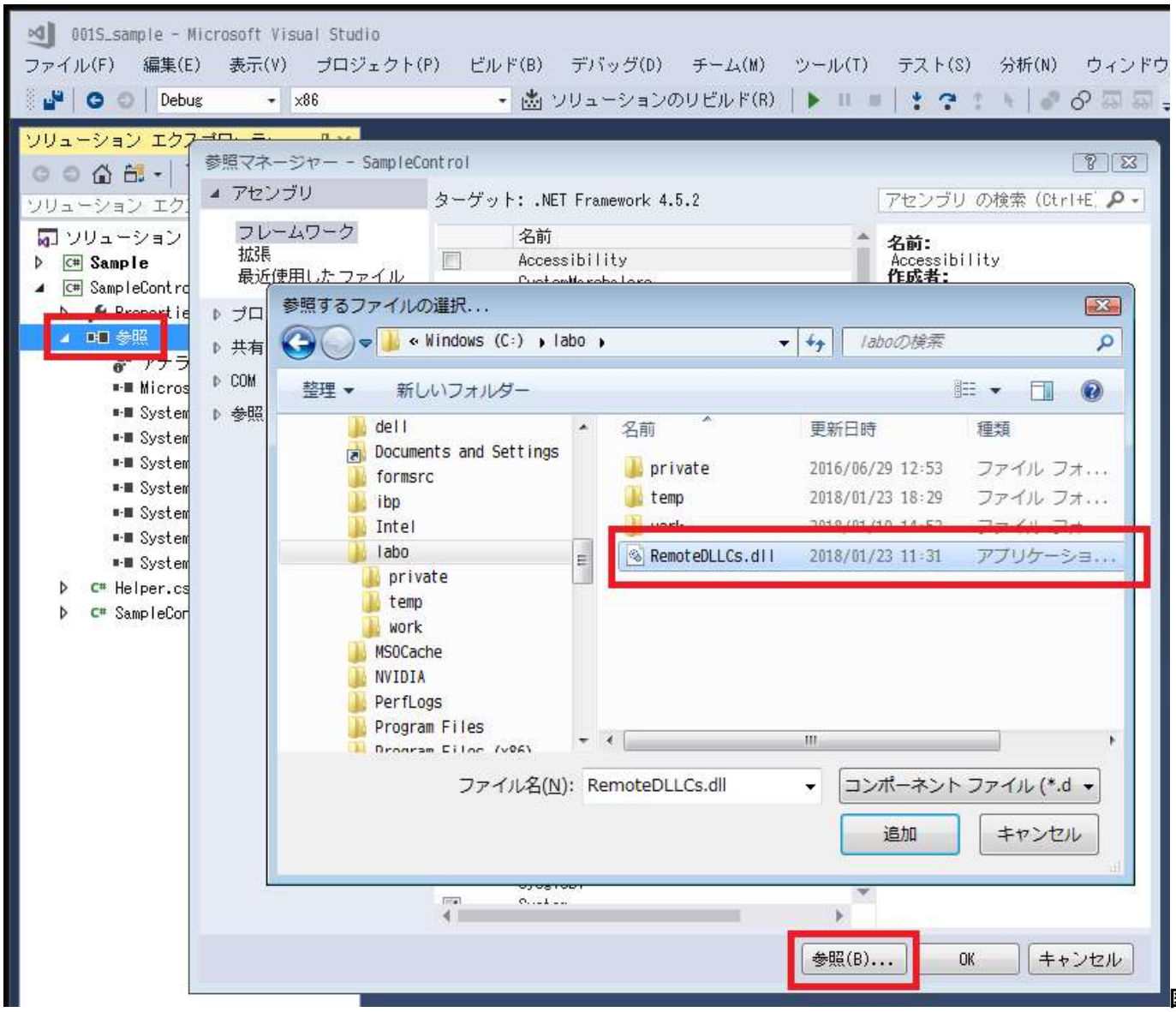


図 3-1 DLL の参照設定(C++版)

3.2.1.2. C#版

(1)Visual Studio のソリューションエクスプローラーにて、ユーザーアプリプロジェクトの「参照」を右クリック → [参照の追加] → (参照マネージャーウィンドウ右下の)[参照] を開いてください。

参照するファイルの選択にて、RemoteDLLCs.dll を設定してください。なお、RemoteDLL.dll も RemoteDLLCs.dll と同じ場所に格納してください。(RemoteDLLCs.dll から RemoteDLL.dll を参照して使用しているため)



3-2 DLL の参照設定(C#版)

3.2.1.3. EXCEL VBA 版

EXCEL　VBA からの参照は、VBA の標準モジュールに、以下のような記述を行ってください。

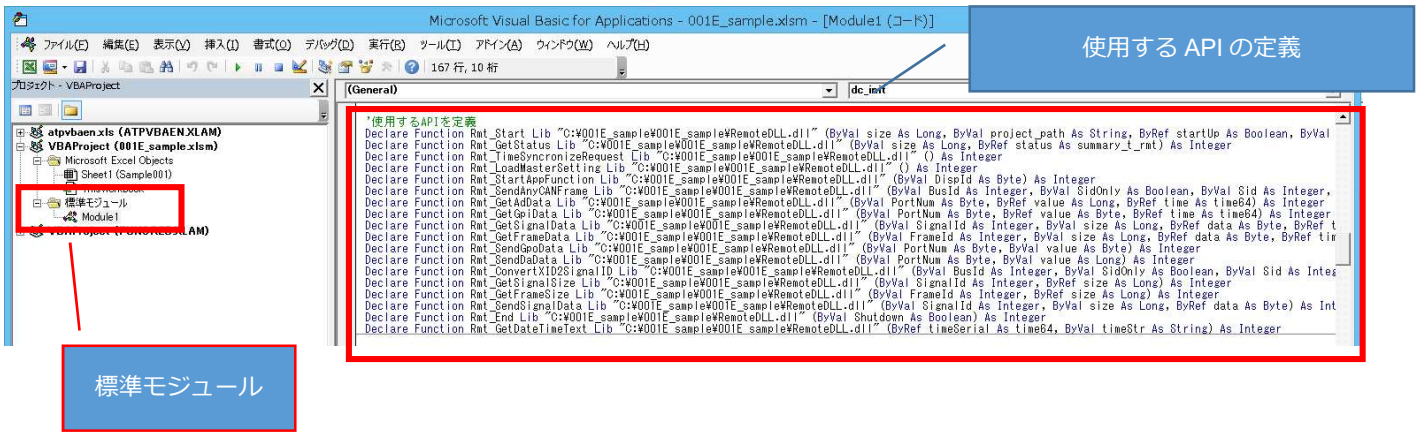


図 3-3 DLL の参照設定(EXCEL VBA 版)

```
Declare Function Rmt_Start Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" (ByVal size As Long, ByVal project_path
As String, ByRef startUp As Boolean, ByVal StartWithMainPanel As Boolean) As Integer

Declare Function Rmt_GetStatus Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" (ByVal size As Long, ByRef status As
summary_t_rmt) As Integer

Declare Function Rmt_TimeSyncronizeRequest Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" () As Integer

Declare Function Rmt_LoadMasterSetting Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" () As Integer

Declare Function Rmt_StartAppFunction Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" (ByVal DispId As Byte) As
Integer

Declare Function Rmt_SendAnyCANFrame Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" (ByVal BusId As Integer,
ByVal SidOnly As Boolean, ByVal Sid As Integer, ByVal Eid As Long, ByVal isCANFD As Boolean, ByVal bitRateSwitch As
Boolean, ByVal dataLength As Long, ByRef data As Byte) As Integer
 :
 :
```

図 3-4 使用する API の定義拡大(EXCEL VBA 版)

3.2.2　Visual Studio でのインクルード設定

3.2.2.1. C++版

Visual Studio のソリューションエクスプローラーにて、ユーザーアプリのプロジェクトを右クリック→[プロパティ] → [構成プロパティ] → [VC++ディレクトリ] を開いてください。

(1)インクルードディレクトリに、インクルードファイル(RemoteDLL.h)が含まれているフォルダ(例では C:¥CANTOOL¥sdk)を指定してください。

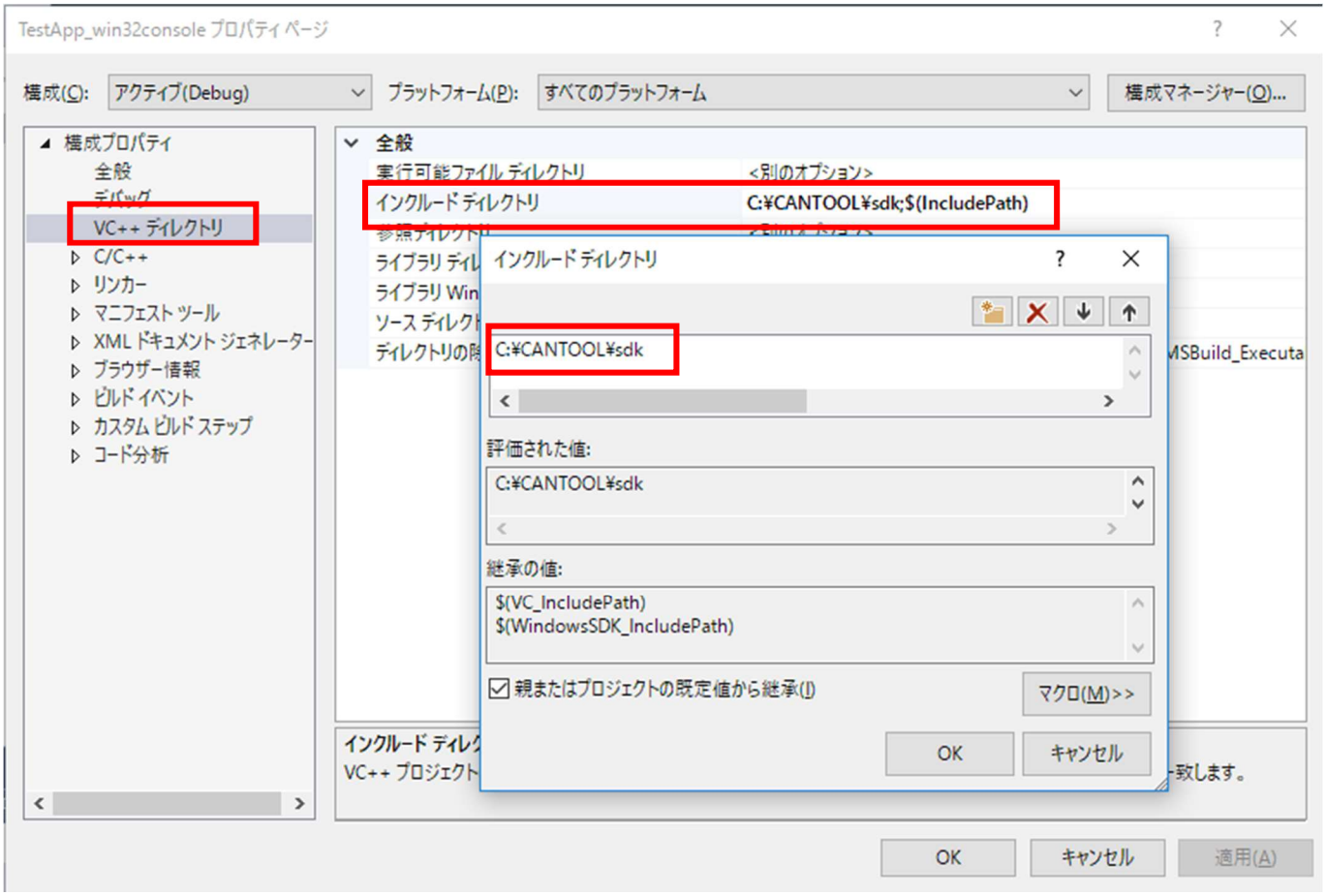


図 3-5 インクルード設定 (C++版)-1

(2)Visual Studio のソリューションエクスプローラーにて、ユーザーアプリのプロジェクトを右クリック → [プロパティ] → [構成プロパティ] → [リンカ] → [入力] を開いてください。

追加の依存ファイルにて、RemoteDLL.lib を設定してください。

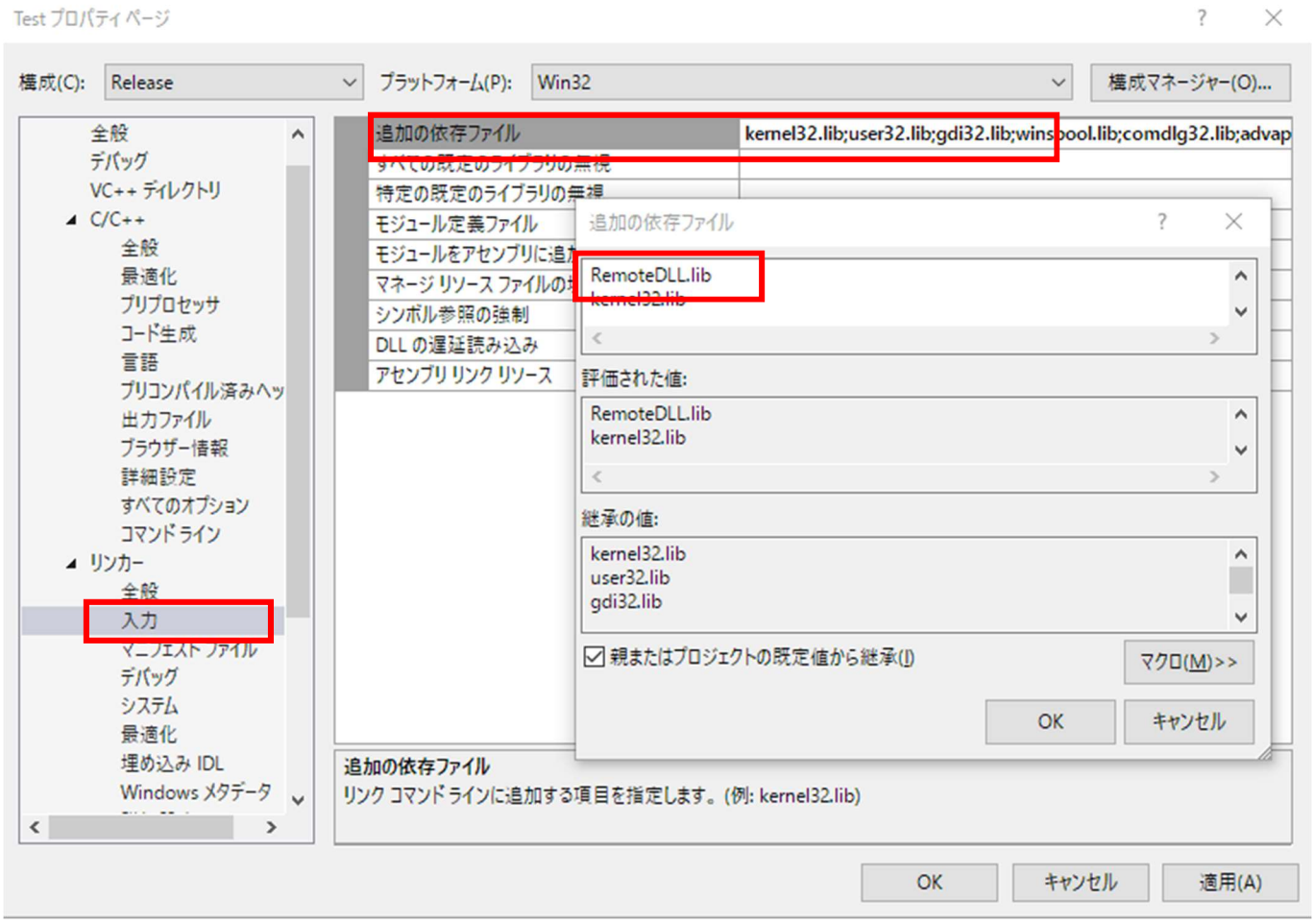


図 3-6 インクルード設定(C++版)-2

3.2.2.2. C#版

C#ではインクルード設定は必要ありません。（DLL の参照設定だけで使用可能です）

3.2.2.3. EXCEL VAB 版

EXCEL VBA ではインクルード設定は必要ありません。（DLL の参照設定だけで使用可能です）

3.3 使用手順

3.3.1　ユーザーアプリ処理フロー例

ユーザーアプリでの処理フローの例を以下に示します。

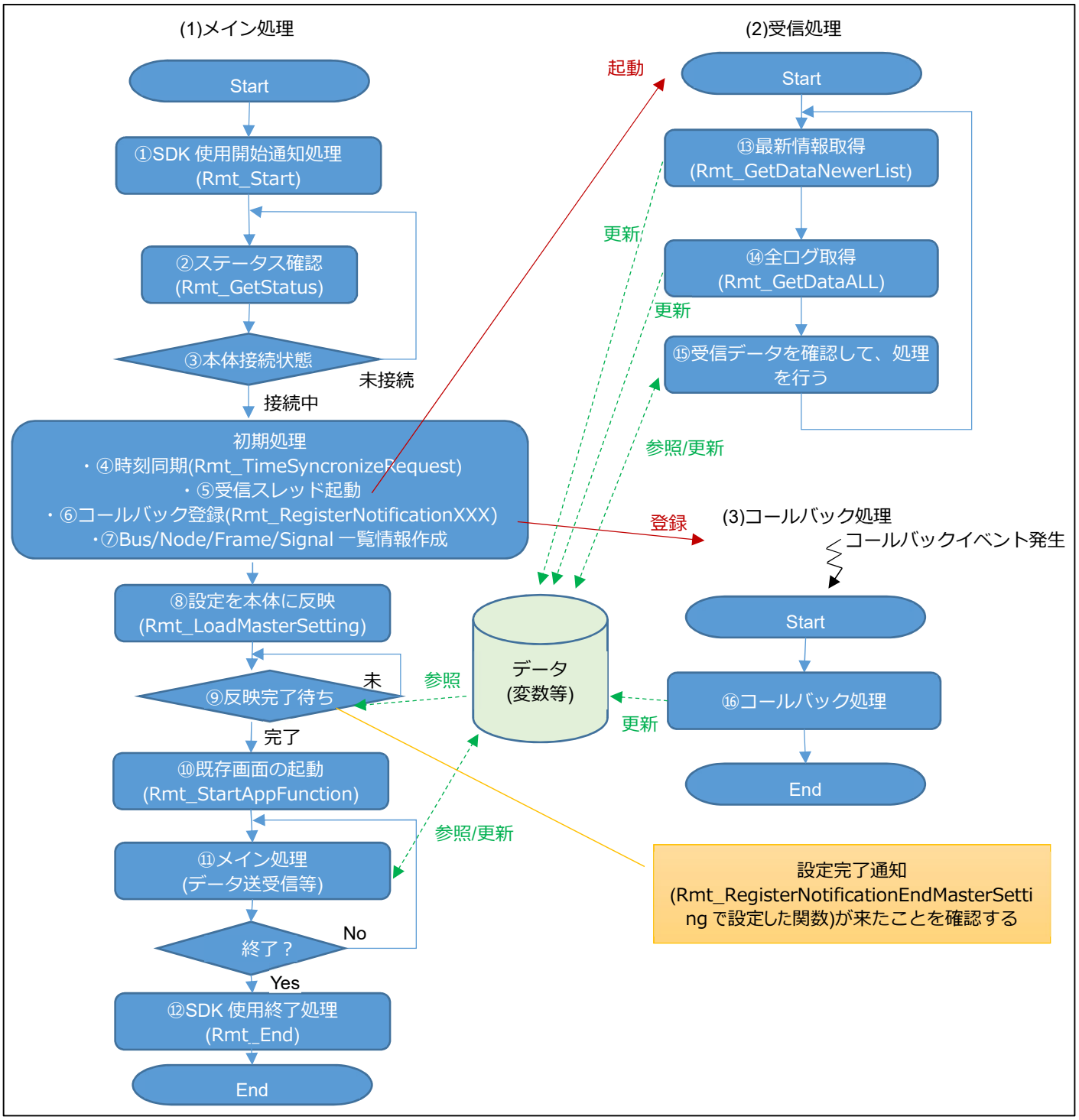


図 3-7　処理フロー例

処理フロー例では、大きく３つの処理に分割しています。

(1) メイン処理

初期化処理及び、送受信処理(受信自体は(2)受信処理で実施し取得したデータを参照する)を行う。

(2) 受信処理

受信を行い、必要なデータを格納する。受信したデータは(1)メイン処理にて参照する。

受信処理のみを独立することで、メイン処理を停滞させないようにする。

また、全ログを取得する場合は、定期的にデータを取得する必要がある(一定量で上書きされて消えるため)ためメイン処理とは別のスレッドで処理したほうがよい。

(3) コールバック処理

状態変更時などイベント発生時に必要な処理を行う。

処理フロー例の各処理についての補足説明を以下に示します。

表 3-1 処理説明

処理 No	処理内容	説明
①	SDK 使用開始通知処理 (Rmt_Start)	ユーザーアプリから SDK を開始する際に最初にコールします。なお、終了時は、⑫SDK 使用終了通知を必ず実行してください。 本処理実行時に、PC アプリ（CANTOOL PANEL）を CANTOOL のプロジェクトファイルを指定して起動します。PC アプリがすでに起動している場合には、新たに起動しません。(読み込み済みのプロジェクトファイルが使用されます) ★なお、プロジェクトファイルについては、事前に PC アプリを使用して作成しておく必要があります。
②	ステータス確認 (Rmt_GetStatus)	PC と CANTOOL 本体が接続しているかを確認するためのコマンドを実行します。 ＊CANTOOL 本体が接続されていないと実行できないコマンドがあるため、最初に接続状態を確認します。
③	本体接続状態	'③ステータス確認の結果に含まれる CANTOOL 本体との接続状態を確認します。
④	時刻同期 (Rmt_TimeSyncronizeRequest)	CANTOOL 本体に PC の時刻を設定します。
⑤	受信スレッド起動	CANTOOL 本体からの受信データは、専用受信スレッドを使用して定期的に取得してください。 特に、全ログ(時系列のログ)を取得する場合は、常時ログが発生するため取得しないと欠落することがあります。
⑥	コールバック登録 (Rmt_RegisterNotificationXXX)	状態の変更通知を受け取るためのコールバック関数を登録します。 ステータスが変更となった場合などに通知されます。
⑦	Bus/Node/Frame/Signal 一覧情報作成	プロジェクトファイルに設定されている各種情報を取得します。各 Bus,Node、Frame、Signal にはユニークな ID が振られるため、最初に一覧を作成することで以降の処理を行いやすくなります。
⑧	設定を本体に反映 (Rmt_LoadMasterSetting)	プロジェクトファイルを、CANTOOL 本体に反映します。
⑨	反映完了待ち	'⑧の処理が完了するまでに少々時間がかかりますので、完了したことを通知するコールバック関

処理 No	処理内容	説明
		数処理が呼び出されるまで待ちます。コールバック関数にて反映完了待ちのフラグを ON するなどして、本処理に結果を通知してください。
⑩	既存画面の起動(Rmt_StartAppFunction)	PC アプリ(CANTOOL PANEL)の画面を呼び出します。 ステータスモニタ画面など、既存の画面を活用することでログ確認画面をユーザーにて別途作成する必要はありません。
⑪	メイン処理(データ送受信等)	GUI を作成してデータを見える化するなど、ユーザーの処理を行います。 データ送信は、Rmt_SendXXX の API を使用することで簡単に行うことが可能です。
⑫	SDK 使用終了処理(Rmt_End)	ユーザーアプリから SDK の使用を終了することを通知します。PC アプリを終了することも可能です。(複数のユーザアプリが起動している場合には、すべてのユーザーアプリの終了を待ってから PC アプリが終了します)
⑬	最新情報取得(Rmt_GetDataNewerList)	各データの最新値を取得します。現在の値を元に処理を行う場合には本処理を使用します。
⑭	全ログ取得(Rmt_GetDataALL)	全ログを取得します。時系列に発生する事象を見ながら処理する場合には本処理を使用します。受信スレッドを効率的に実行する場合には、取得したデータは FIFO 経由でメイン処理に送り、メイン処理側でデータ処理することをお勧めします。
⑮	受信データを確認して、処理を行う	簡単な処理は、受信スレッドにて実行します。
⑯	コールバック処理	登録しておいたコールバック処理を実行します。 コールバックを登録した関数毎に、処理内容が異なります。 ・Rmt_RegisterNotificationStatusChange Rmt_GetStatus を実行して、最新のステータスを取得してください。 ・Rmt_RegisterNotificationUserSW ・Rmt_RegisterNotificationEndedRecordingLog ・Rmt_RegisterNotificationEndMasterSetting コールバック関数の引数で状態を取得できます。

3.3.2 内部情報取得

各内部情報は、以下 API を利用して、取得できます。

表 3-2 内部情報取得方法

分類	内部情報	取得方法	備考
Bus	BusList	Rmt_GetBusList	
	BusID	定義値(RMT_BUS_ID_XXX)を使用	定義については、「RemoteDLL.h」参照
Node	NodeList	Rmt_GetNodeList BusID を使用して取得する	BusID は、定義を使用
	NodeID	NodeList から取得する	
Frame	FrameList	Rmt_GetFrameList NodeID を使用して取得する	NodeID は、Rmt_GetNodeList で取得した NodeList から取得する
	FrameID	Rmt_ConvertXID2FrameID BusID, XID(SID, EID)を使用して取得する	BusID は、定義を使用
		FrameList から取得する	
Signal	SignalList	Rmt_GetSignalList FrameID を使用して取得する	FrameID は、Rmt_GetFrameList で取得した FrameList から取得する
		Rmt_ConvertSignalListFromFrame FrameID を使用して取得する	同上
	SignalID	Rmt_ConvertXID2SignalID BusID, XID(SID, EID), SignalName を使用して取得する	BusID は、定義を使用
		SignalList から取得する	

3.3.3 データ送受信方法

各バスにおける、データの送受信方法を説明します。

内部情報の取得方法については、「3.3.2 内部情報取得」をご参照ください。

3.3.3.1. CAN / LIN

3.3.3.1.1. Frame 送信

(1)FrameID を利用して送信する場合

1. FrameID を取得する。
2. Rmt_SendFrameData にて、FrameID と CAN の情報(サイズ)を指定して、データを送信する。

(2)任意のデータを送信する場合

1. Rmt_SendAnyCANFrame にて、BusID と CAN の情報(XID, サイズ)を指定して、任意の Frame データを送信する。

3.3.3.1.2. Frame 受信

(1)特定の Frame の最新値を取得する場合

1. FrameID を取得する。
2. Rmt_GetFrameData にて、FrameID と CAN の情報(サイズ)を指定して、データの最新値を取得する。

(2)受信した全 Frame の最新値を取得する場合

1. Rmt_GetDataNewerList にて、送受信した Frame データの最新値を取得する。

(3)受信したすべての Frame を取得する場合

1. Rmt_GetDataALL にて、送受信した Frame データのすべての値を取得する。

※受信したデータは一旦 PC アプリ(CANTOOL PANEL)側で保持されるため、定期的に取得処理が必要です。

3.3.3.1.3. Signal 送信

(1)SignalID を利用して送信する場合

1. SignalID を取得する。
2. Rmt_SendSignalData にて、SignalID と CAN の情報(サイズ)を指定して、データを送信する。

※Signal 送信では、指定した Signal データのみを書き換えて送信されます。指定外のデータは前回送信時の情報が使用されます。

3.3.3.1.4. Signal 受信

(1) 特定の Signal の最新値を取得する場合

1. SignalID を取得する。

2. Rmt_GetSignalData にて、SignalID と CAN の情報(サイズ)を指定して、データの最新値を取得する。

(2)Frame 受信後のデータを利用する場合

1. FrameID を取得する。
2. Rmt_ConvertSignalListFromFrame にて、FrameID とと CAN の情報(サイズ)を指定して、SignalList を取得する。
3. SignalList から、SignalName を指定して、データの最新値を取得する。

3.3.3.2. AD / DA

3.3.3.2.1. AD 受信

1．Rmt_GetAdData にて、Bus 番号を指定して、データの最新値を取得する。

3.3.3.2.2. DA 送信

1．Rmt_SendDaData にて、Bus 番号を指定して、データを送信する。

3.3.3.3. GPI / GPO

3.3.3.3.1. GPI 受信

1．Rmt_GetGpiData にて、ポートを指定して、データの最新値を取得する。

3.3.3.3.2. GPO 受信

1．Rmt_SendGpoData にて、ポートを指定して、データを送信する。

3.4 注意事項

各 DLL 使用時の注意点を記載します。

3.4.1 Excel VBA 版

3.4.1.1. 構造体(ユーザー定義型)について

・構造体のメンバーは、VBA ではユーザー定義型とし、メンバーは全て BYTE 型で定義してください。

・RemoteDLL(C++版)の構造体は 4 バイトアライメントされているので、VBA 側は 4 バイトアライメントに合わせてください。

(例)

■RemoteDLL(C++版)での構造体定義

```
typedef struct {
    char        AAA[9];
    BYTE        BBB[4];
    DWORD       CC[2];
    BYTE        DDD;
    ULONGLONG   EEE;
} structName;
```

■VBA でのユーザー定義型定義

```
Public Type structName
    AAA(1 To 9) As Byte
    Dummy1(1 To 3) As Byte
    BBB(1 To 4) As Byte
    CCC(1 To 2) As Byte
    DDD As Byte
    Dummy2 As Byte
    EEE(1 To 8) As Byte
End Type
```

4 バイトアライメントのため、
ダミーデータで埋める

3.4.1.2. API コール時の引数指定について

C++での変数型と VBA での変数型の対応は、以下の通りです。

値渡しをする際は、VBA にて ByVal キーワードを指定し、参照渡しの際は ByRef キーワードを指定してください。

API(C++版)での変数型	Excel VBA での変数型	説明
BOOL	Long (TRUE の場合 1、 FALSE の場合 0 を指定)	VBA で True を設定しても、API では FALSE とみなされるため、 VBA 側では Long 型に True(1)または False(0)を設定してください。
WORD	Integer	2 バイト
DWORD	Long	4 バイト
int	Long	4 バイト
ULONGLONG	ユーザー定義型 BYTE 型配列[8] または LONG 型配列[2]	VBA では 8 バイトの整数型を扱うことができません。 BYTE 側配列または LONG 型配列にて、8 バイト分のデータを確保してください。
char*	String	文字列型
構造体	ユーザー定義型 (メンバーはすべて BYTE 型または BYTE 配列で扱う)	「3.4.1.1 構造体(ユーザー定義型)について」参照

(例)

■RemoteDLL(C++版)での定義

int Rmt_Start(**DWORD** size, **char*** project_path, **BOOL*** Startup, **BOOL** StartWithMainPanel = FALSE)

■Excel VBA での定義

Declare Function **Rmt_Start** Lib "C:¥001E_sample¥001E_sample¥RemoteDLLvba.dll" (**ByVal** size As **Long**, **ByVal** project_path As **String**, **ByRef** startUp As **Long**, **ByVal** StartWithMainPanel As **Long**) As Integer

4. ライブラリインターフェース仕様

4.1 ライブラリインターフェース一覧

SDK が提供する API の一覧を以下に示します。各 API の詳細は本冊子の末尾を参照してください。

※グレーアウトされているものは、将来対応予定です。

表 4-1 ライブラリインターフェース一覧

カテゴリ	説明	名称
スタート	SDK 使用開始通知処理	Rmt_Start
エンド	SDK 使用終了処理	Rmt_End
ステータス	各種状態を取得する。	Rmt_GetStatus
	指定の LED を制御する。	Rmt_SetLED
	状態変更時に通知される。GetStatus で取得すること。	Rmt_RegisterNotificationStatusChange
コントロール	対応する車載ネットワークバス情報を取得する。	Rmt_GetBusList
	各バスに設定されているノード情報を取得する。	Rmt_GetNodeList
	各ノードに設定されているフレーム情報を取得する。	Rmt_GetFrameList
	各フレームに設定されているシグナル情報を取得する。	Rmt_GetSignalList
	指定された CAN バスのバス設定を取得する。	Rmt_GetBusConfigCan
	LIN のバス設定を取得する。	Rmt_GetBusConfigLin
	指定された AD ポートのバス設定を取得する。	Rmt_GetBusConfigAd
	指定された DA ポートのバス設定を取得する。	Rmt_GetBusConfigDa
	指定された GPI ポートのバス設定を取得する。	Rmt_GetBusConfigGpi
	指定された GPO ポートのバス設定を取得する。	Rmt_GetBusConfigGpo
	指定された Node の設定を取得する。	Rmt_GetNodeConfig
	指定された Frame の設定を取得する。	Rmt_GetFrameConfig
	指定された Signal の設定を取得する。	Rmt_GetSignalConfig
	Frame に割り当てられている Signal のリストを取得する	Rmt_ConvertSignalListFromFrame
	BusID と XID から FrameID を取得する。	Rmt_ConvertXID2FrameID
	BusID と XID と SignalSymbol から SignalID を取得する。	Rmt_ConvertXID2SignalID
	本体の UART の設定の変更を行う。	Rmt_SetUartSetting
	本体の UART の設定を取得する。	Rmt_GetUartSetting
	指定したフレーム ID のデータサイズ(バイト)を取得する。	Rmt_GetFrameSize
	指定したシグナル ID のデータサイズ(バイト)を取得する。	Rmt_GetSignalSize
	定周期送信 Frame の有効無効を動的に切り替えする。	Rmt_DynamicPeriodicFrameControl
	ユーザースイッチ操作時に通知される。	Rmt_RegisterNotificationUserSW
	Config 変更時に通知される。	Rmt_RegisterNotificationConfigChange
	ステータスモニタ用ログをクリア時に通知される	Rmt_RegisterNotificationStatusMonitor DataClear

カテゴリ	説明	名称
データ送信	指定フレームのデータを送信する。周期送信の値に反映される。	Rmt_SendFrameData
	指定シグナルのデータを送信する。周期送信の値に反映される。	Rmt_SendSignalData
	指定ポートの DA の値を変更する。	Rmt_SendDaData
	指定ポートの GPO の値を変更する。	Rmt_SendGpoData
	UART へのデータを送信する。	Rmt_SendUartData
	任意の CAN Frame データを送信する。	Rmt_SendAnyCANFrame
	送信データを Frame サイズに分割し、指定周期で送信を行う。	Rmt_SendCyclicData
	送信データを Frame サイズに分割し、指定周期で送信を行う。(複数 Frame 制御)	Rmt_SendCyclicData2
	MultiPacket データを送信する	Rmt_SendMultiPacketData
	MultiPacket データを送信する(N_TAtype あり)	Rmt_SendMultiPacketData2
	MultiPacket データを送信する(N_TAtype、CAN Frame データ空欄部のパディング指定)	Rmt_SendMultiPacketData3
	MultiPacket への応答内容設定	Rmt_ResponseMultiPacketData
	MultiPacket への応答内容設定(N_TAtype、CAN Frame データ空欄部のパディング指定)	Rmt_ResponseMultiPacketData3
データ受信	指定フレームの最新データを取得する。	Rmt_GetFrameData
	指定シグナルの最新データを取得する。	Rmt_GetSignalData
	指定ポートの AD の値を取得する。	Rmt_GetAdData
	指定ポートの GPI の値を取得する。	Rmt_GetGpiData
	最新データを取得する。 車載ネットワーク、GPIO、AD/DA を含む。	Rmt_GetDataNewerList
	送受信したデータを全て取得する。前回取得以降のデータが取得可能。車載ネットワーク、GPIO、AD/DA を含む。	Rmt_GetDataALL
	MultiPacket 状態を取得する	Rmt_GetMultiPacketStatus
	MulitPacket 応答状態を取得する	Rmt_GetResponseMultiPacketDataStatus
	UART の受信データが通知される	Rmt_RegisterNotificationGetUartData
PC ロギング	PC でのロギングを開始する。	Rmt_StartRecordingLog
	PC でのロギングを停止する。	Rmt_EndRecordingLog
	StartRecordingLog()、EndRecordingLog()の応答として、ログ記録の実行終了を通知する。	Rmt_RegisterNotificationEndedRecordingLog
	指定した再生ファイルを実行する。	Rmt_StartPlayingFile
	再生ファイルの実行を終了する。	Rmt_EndPlayingFile
	ロギング時、ロストの発生状態を通知する。	Rmt_RegisterNotificationLostStatus
再生	Rmt_StartPlayingFile()、Rmt_EndPlayingFile()の応答として、再生ファイルの実行終了を通知する。(PC による再生)	Rmt_RegisterNotificationEndedPlayingFile
	再生ファイルの再生位置を通知する。	Rmt_RegisterNotificationPlayingFileProgress
プロジェクト	プロジェクト設定を本体に反映する	Rmt_LoadMasterSetting
	指定のインターフェース設定ファイルを、CANTOOL 本体に反映する	Rmt_LoadMasterSetting2

カテゴリ	説明	名称
	本体へのプロジェクト設定完了を通知する。 (Rmt_LoadMasterSetting の完了通知)	Rmt_RegisterNotificationEndMasterSetting
時間	PC の時刻を CANTOOL 本体に設定する	Rmt_TimeSyncronizeRequest
	Rmt_GetXXData で取得した時間(シリアル値)を 年/月/日　時:分:秒.ミリ秒.マイクロ秒.ナノ秒("YYYY/MM/DD HH:MM:SS.mmm.uuu.n")の形式に変換する。	Rmt_GetDateTimeText
アプリケーションコントロール	既存アプリの各画面を起動する。	Rmt_StartAppFunction
	既存アプリの各画面を起動する。(オプション追加)	Rmt_StartAppFunction2

4.2 ライブラリインターフェース詳細

「V016002-702_CANTOOL_A1_SDK-ReferenceManual-API」をご参照ください。

アイテック阪急阪神 株式会社

Copyright (C) 2017 ITEC HANKYU HANSHIN CO., LTD. All Rights Reserved.

ライブラリインターフェース仕様

アイテック阪急阪神 株式会社

(※)各エラーや定義、構造体については、RemoteDLLのヘッダーファイル(RemoteDLL.h)　を参照してください。

Rev.　01.06
最終更新日：　2022/08/31

種別	定義	概要	引数	設定可能範囲	取得内容(戻り値)	説明
Connection	Rmt_Start	ユーザーアプリをCANTOOL-PANELに接続する。 CANTOOL-PANEL未起動時には、CANTOOL-PANELを起動する	①DWORD size [I] プロジェクトファイルパスのサイズ(バイト) ②char* project_path [I] プロジェクトファイルパス(フルパス指定) ③BOOL* Startup [O] CANTOOL-PANEL起動状態 ④(省略可)BOOL StartWithMainPanel [I] CANTOOL-PANELメインパネル表示/非表示	①最大255。 ②・フルパス、UTF-8で指定する。フォルダ区切り文字は「¥¥」。 ・プロジェクトファイルパスのサイズが0で、プロジェクトファイルの選択画面が表示された状態でCANTOOL-PANELが起動する。 ③TURE：起動済み,FALSE:未起動 ④・TRUE:メインパネル表示あり／FALSE：メインパネル表示なし(Default:FALSE)	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・本APIによりCANTOOL-PANELにユーザーアプリを接続する。接続することでSDKのAPIが利用できるようになる。複数のユーザーアプリ(最大5個)を接続することが可能。 ・引数③に実行時点のCANTOOL-PANELの起動状態(起動済み/未起動)を格納して返す。 ・CANTOOL-PANEL未起動時は、「CANTOOL_PANEL_PATH」のシステム変数にて指定されているCANTOOL-PANELを起動し、PANEL起動成功が返る。 ・CANTOOL-PANEL起動済時は、起動処理は行わない。 ・CANTOOL-PANEL起動成功時は、引数①、②で指定したプロジェクトファイルが読み込まれる。起動済時は、すでに読み込んでいるプロジェクトファイルを使用する。 ・引数②で指定したパスが読み込めなければ処理失敗となる。 ・引数④により、CANTOOL-PANEL表示/非表示を指定する。省略時は非表示で起動する。PANEL起動済時は指定は無効となり、その時のCANTOOL-PANEL表示状態を維持する。 ・本APIコール後、CANTOOL-PANELの起動処理が実施されるため、次のAPIコールはインターバルを設けて実施すること。(推奨インターバル：5秒)
	Rmt_End	ユーザーアプリをCANTOOL-PANELから切断する。	①(省略可)BOOL　Shutdown[I] CANTOOL-CANTOOL-PANEL終了 する/しない	①TRUE：終了する FALSE：終了しない (Default:FALSE)	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・本APIによりCANTOOL-PANELからユーザーアプリを切断する。切断することでCANTOOL-PANELがユーザーアプリの情報を破棄する。 ・引数①に「TRUE」を設定すると、CANTOOL-PANEL自身が終了する。ただし、他に接続しているユーザーアプリがある場合は、終了しない。CANTOOL-PANEL起動中に一度でも「TRUE」を指定すると、全ユーザーアプリが切断した際に終了する。 ・引数①に「FALSE」を設定すると、全ユーザーアプリが切断してもCANTOOL-PANELは終了しない。 ・プロジェクト設定ファイルを読み直す場合は、CANTOOL-PANELを終了させる必要がある。 ・Rmt_End()を呼び出さずにユーザーアプリが終了した場合、一定時間経過後にCANTOOL-PANELから切断処理を行う。 ・CANTOOL PANEL側で未保存のデータがある場合、確認ダイアログが表示され、CANTOOL-PANELは終了しない。 ・本APIコール後、CANTOOL-PANELの終了処理が実施されるため、次のAPIコールはインターバルを設けて実施すること。(推奨インターバル：5秒)
Status	Rmt_GetStatus	各種状態を取得する	①DWORD size　[I] 各種状態格納サイズ(バイト) ②summary_t_rmt* status [O] ステータス(各種状態)	①各種状態格納サイズは、 「sizeof(summary_t_rmt)」を固定でセットすること。 ②summary_t_rmt構造体のメモリを確保する。	RET_OK:正常 負数:エラー	・summary_t_rmt構造体の情報を取得する。 構造体は、[summary_t_rmt構造体]定義を参照。 ・状態変化については Rmt_RegisterNotificationStatusChange()にコールバック関数を登録することで、状態変更時に登録したコールバック関数が通知される。
	Rmt_SetLED	指定のLEDを制御する	①int port [I] LED番号 ②int value [I] 色 ③int ledSts [I] 動作(点灯、点滅、消灯)	① LED_PORT_USER1 LED_PORT_USER2 ② LED_COLOR_OFF LED_COLOR_RED LED_COLOR_GREEN LED_COLOR_ORANGE ③ LED_STATUS_OFF LED_STATUS_ON LED_STATUS_BRINK	RET_OK:正常,負数:エラー エラーは[エラー一覧]シートを参照。	・指定したLED番号の色と動作を制御する。 ・色と動作のどちらかを「消灯」とした場合、LEDは消灯状態となる。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
Control	Rmt_GetBusList	対応する車載ネットワークBus情報を取得する	①DWORD size　[I] バス情報格納サイズ(バイト) ②bus_list_t_rmt* BusList　[O] バス情報	①各種状態格納サイズは、 「sizeof(bus_list_max_t_rmt)」を固定でセットすること。 ②bus_list_max_t_rmt構造体のメモリを確保する。	int　BusList件数(Node種別件数分) エラーは[エラー一覧]シートを参照。	・BusListの領域は、bus_list_max_t_rmt構造体サイズ分のメモリを確保すること。 ・Busリストからは、bus_list_t_rmt構造体の情報が取得できる。 ・戻り値のBusList件数分、BusListに情報が格納される。 構造体は、[bus_list_t_rmt構造体]定義を参照。
	Rmt_GetNodeList	各Busに設定されているNode情報を取得する	①WORD BusId　[I] Bus ID ②DWORD size　[I] Node情報格納サイズ ③node_list_t_rmt* NodeList　[O] Node情報	① RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 RMT_BUS_ID_LIN ②Node情報格納サイズは、 「sizeof(node_list_max_t_rmt)」を固定でセットすること。 ③node_list_t_rmt構造体のメモリを確保する。	int　NodeList件数 エラーは[エラー一覧]シートを参照。	・指定バスに登録されているノードのリストを返す。 ・node_list_max_t_rmt構造体のメモリを確保すること。 ・ノードリストからは、node_list_t_rmt構造体の情報が取得できる。 ・戻り値のNodeList件数分、NodeListに情報が格納される。 構造体は、[node_list_t_rmt構造体]定義を参照。
	Rmt_GetFrameList	各Nodeに設定されているFrame情報を取得する	①WORD NodeId　[I] Node ID ②DWORD size　[I] Frame情報格納サイズ(バイト) ③frame_list_t_rmt* FrameList　[O] Frame情報	[①がNODEID_UNDEFINEDの場合] ②Frame情報格納サイズは、 「sizeof(undefined_frame_list_max_t_rmt)」を固定でセットすること。 ③undefined_frame_list_max_t_rmt構造体のメモリを確保する。 [①がNODEID_UNDEFINED以外の場合] ②Frame情報格納サイズは、 「sizeof(defined_frame_list_max_t_rmt)」を固定でセットすること。 ③defined_frame_list_max_t_rmt構造体のメモリを確保する。	int　FrameList件数 エラーは[エラー一覧]シートを参照。	[①がNODEID_UNDEFINEDの場合] ・指定ノードに登録されているフレームのリストを返す。 ・undefined_frame_list_max_t_rmt構造体のメモリを確保すること。 ・フレームリストからは、frame_list_t_rmt構造体の情報が取得できる。 ・戻り値のFrameList件数分、FrameListに情報が格納される。 [①がNODEID_UNDEFINED以外の場合] ・指定ノードに登録されているフレームのリストを返す。 ・defined_frame_list_max_t_rmt構造体のメモリを確保すること。 ・フレームリストからは、frame_list_t_rmt構造体の情報が取得できる。 ・戻り値のFrameList件数分、FrameListに情報が格納される。 構造体は、[frame_list_t_rmt構造体]定義を参照。
	Rmt_GetSignalList	各Frameに設定されているSignal情報を取得する	①WORD FrameId [I] Frame ID ②DWORD size　[I] Signal情報格納サイズ(バイト) ③signal_list_t_rmt* SignalList　[O] Signal情報	①0～4999 内部管理IDのためRmt_GetFrameListで取得した値を使うこと。 ②Signal情報格納サイズは、 「sizeof(signal_list_max_t_rmt)」を固定でセットすること。 ③signal_list_max_t_rmt構造体のメモリを確保する。	int　SignalList件数 エラーは[エラー一覧]シートを参照。	・指定フレームに登録されているシグナルのリストを返す。 ・signal_list_max_t_rmt構造体のメモリを確保すること。 ・シグナルリストからは、signal_list_t_rmt構造体の情報が取得できる。 ・戻り値のSignalList件数分、SignalListに情報が格納される。 構造体は、[signal_list_t_rmt構造体]定義を参照。
	Rmt_GetBusConfigCan	CANの設定を取得する	①WORD BusId [I] バスID ②DWORD size [I] CANのBus Config格納サイズ(バイト) ③cfg_bus_can_t_rmt* BusConfigCan [O] CANのBus Config	① RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ②CANのBus Config格納サイズは、 「sizeof(cfg_bus_can_t_rmt)」を固定でセットすること。 ③cfg_bus_can_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したバスID(0：CAN0、1：CAN1、2：CAN2、3：CAN3)の設定の取得を行う。 ・CANのBus Configは、[cfg_bus_can_t_rmt構造体]定義を参照。
	Rmt_GetBusConfigAd	ADの設定を取得する	①WORD BusId [I] バスID ②DWORD size [I] ADのBus Config格納サイズ(バイト) ③cfg_bus_adc_t_rmt* BusConfigAd [O] ADのBus Config	① RMT_BUS_ID_AD0 RMT_BUS_ID_AD1 ②ADのBus Config格納サイズは、 「sizeof(cfg_bus_adc_t_rmt)」を固定でセットすること。 ③cfg_bus_adc_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したバスID(5：AD0、6：AD1)の設定の取得を行う。 ・ADのBus Configは、[cfg_bus_adc_t_rmt構造体]定義を参照。

Rmt_GetBusConfigDa	DAの設定を取得する	①WORD BusId [I] バスID ②DWORD size [I] DAのBus Config格納サイズ(バイト) ③cfg_bus_dac_t_rmt* BusConfigDa [O] DAのBus Config	① RMT_BUS_ID_DA0 RMT_BUS_ID_DA1 ②DAのBus Config格納サイズは、「sizeof(cfg_bus_dac_t_rmt)」を固定でセットすること。 ③cfg_bus_dac_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したバスID(7：DA0、8：DA1)の設定の取得を行う。 ・DAのBus Configは、[cfg_bus_dac_t_rmt構造体]定義を参照。
Rmt_GetBusConfigGpi	GPIの設定を取得する	①WORD BusId [I] バスID ②DWORD size [I] GPIのBus Config格納サイズ(バイト) ③cfg_bus_gpi_t_rmt* BusConfigGpi [O] GPIのBus Config	① RMT_BUS_ID_GPI ②GPIのBus Config格納サイズは、「sizeof(cfg_bus_gpi_t_rmt)」を固定でセットすること。 ③cfg_bus_gpi_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したバスID(9：GPI)の設定の取得を行う。 ・GPIのBus Configは、[cfg_bus_gpi_t_rmt構造体]定義を参照。
Rmt_GetBusConfigGpo	GPOの設定を取得する	①WORD BusId [I] バスID ②DWORD size [I] GPOのBus Config格納サイズ(バイト) ③cfg_bus_gpo_t_rmt* BusConfigGpo [O] GPOのBus Config	① RMT_BUS_ID_GPO ②GPOのBus Config格納サイズは、「sizeof(cfg_bus_gpo_t_rmt)」を固定でセットすること。 ③cfg_bus_gpo_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したバスID(10：GPO)の設定の取得を行う。 ・GPOのBus Configは、[cfg_bus_gpo_t_rmt構造体]定義を参照。
Rmt_GetNodeConfig	Nodeの名前や、関連するFrameの設定を取得する	①WORD NodeId [I] ノードID ②DWORD size [I] ノード Config格納サイズ(バイト) ③cfg_node_t_rmt* NodeConfig [O] ノード Config	②ノード Config格納サイズは、「sizeof(cfg_node_t_rmt)」を固定でセットすること。 ③cfg_node_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したノードIDの設定の取得を行う。 ・ノード Configは、[cfg_node_t_rmt構造体]定義を参照。
Rmt_GetFrameConfig	Frameの設定を取得する	①WORD FrameId [I] フレームID ②DWORD size [I] フレーム Config格納サイズ(バイト) ③cfg_frame_t_rmt* FrameConfig [O] フレーム Config	①0～4999 内部管理IDのためRmt_GetFrameList or Rmt_ConvertXID2FrameIDで取得した値を使うこと。 ②フレーム Config格納サイズは、「sizeof(cfg_frame_t_rmt)」を固定でセットすること。 ③cfg_frame_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したフレームIDの設定の取得を行う。 ・フレーム Configは、[cfg_frame_t_rmt構造体]定義を参照。
Rmt_GetSignalConfig	Signalの設定を取得する	①WORD SignalId [I] シグナルID ②DWORD size [I] シグナルConfig格納サイズ(バイト) ③cfg_signal_t_rmt* SignalConfig [O] シグナルConfig	①0～29999 内部管理IDのためRmt_GetSignalList or Rmt_ConvertXID2SignalIDで取得した値を使うこと。 ②シグナルConfig格納サイズは、「sizeof(cfg_signal_t_rmt)」を固定でセットすること。 ③cfg_signal_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー	・指定したシグナルIDの設定の取得を行う。 ・シグナル Configは、[cfg_signal_t_rmt構造体]定義を参照。
Rmt_ConvertSignalListFromFrame	Frameに割り当てられているSignalのリストを取得する	①WORD　FrameId　[I] Frame ID ②DWORD FrameSize [I] 1Frameデータのサイズ(バイト) ③BYTE*　FrameData [I] 1Frameのデータ ④DWORD SignalSize [I] SignalDataの格納サイズ(バイト) ⑤cfg_signal_data_t_rmt* SignalData [O] Signalデータ	①0～4999 内部管理IDのためRmt_GetFrameList or Rmt_ConvertXID2FrameIDで取得した値を使うこと。 ④SignalDataサイズは、「sizeof(cfg_signal_data_max_t_rmt)」を固定でセットすること。 ⑤cfg_signal_data_t_rmt構造体のメモリを確保する。	int　SignalData件数 エラーは[エラー一覧]シートを参照。	・SignalDataの領域は、cfg_signal_data_max_t_rmt構造体サイズ分のメモリを確保すること。 ・指定されたフレームのデータを登録されているシグナルデータに変換して返す。 ・シグナルデータは、cfg_signal_data_t_rmt構造体の情報が取得できる。 ・戻り値のSignalData件数分、SignalDataに情報が格納される。 構造体は、[cfg_signal_data_t_rmt構造体]定義を参照。
Rmt_ConvertXID2FrameID	BusIdとXIDからFrameIDを取得する	①WORD BusId [I] バスID ②BOOL SidOnly [I] 拡張ID使用有無 ③WORD Sid [I] 標準フォーマットのベースID 11ビット ④DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤WORD* FrameId [O] フレームID(フレームを一意に識別するID番号)	① RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 RMT_BUS_ID_LIN ② TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ③ CAN：0x00～0x7FF LIN：0x00～0x3F ④CAN:0x00～0x3FFFF ⑤0～4999	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたバス情報に登録されたSid(標準フォーマットのベースID 11ビット)とEid(拡張フォーマットの拡張ID 18ビット)から、FrameId(フレームを一意に識別するID番号)を取得する。 ・標準フォーマットのベースID 11ビットのみを指定する場合は、SidOnlyをTRUEで指定すること。 ・標準フォーマットのベースIDと拡張IDの29ビットを指定する場合は、SidOnlyをFALSEで指定すること。 ・本I/Fで取得したFrameIdを、ConvertSignalListFromFrame、SendFrameData、GetFrameDataなどの引数に使用する。
Rmt_ConvertXID2SignalID	BusIdとXIDとSignalSymbolからSignalIDを取得する	①WORD BusId [I] バスID ②BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ③WORD Sid [I] 標準フォーマットのベースID 11ビット ④DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤char* SignalSymbol [I] シグナル名 ⑥WORD* SignalId [O] シグナルID(シグナルを一意に識別するID番号)	① RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 RMT_BUS_ID_LIN ② TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ③ CAN：0x00～0x7FF LIN：0x00～0x3F ④CAN:0x00～0x3FFFF ⑤1～63文字 ⑥0～29999	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたバス情報に登録されたSid(標準フォーマットのベースID 11ビット)とEid(拡張フォーマットの拡張ID 18ビット)とシグナル名称から、SignalId(シグナルを一意に識別するID番号)を取得する。 ・標準フォーマットのベースID 11ビットのみを指定する場合は、SidOnlyをTRUEで指定すること。 ・標準フォーマットのベースIDと拡張IDの29ビットを指定する場合は、SidOnlyをFALSEで指定すること。 ・本I/Fで取得したSignalIdを、SendSignalData、GetSignalDataなどの引数に使用する。
Rmt_SetUartSetting	本体のUARTの設定の変更を行う	①BOOL enable[I] UART有効/無効 ②BYTE baudrate [I] ボーレート ③BYTE datalength [I] データ長 ④BYTE parity [I] パリティ ⑤BYTE stopbit [I] ストップビット	①FALSE:無効 TRUE:有効 ② UART_SETTING_BR_4800 UART_SETTING_BR_9600 UART_SETTING_BR_19200 UART_SETTING_BR_38400 UART_SETTING_BR_115200 UART_SETTING_BR_230400 ③ UART_SETTING_LEN8 UART_SETTING_LEN7 ④ UART_SETTING_PAR_NONE UART_SETTING_PAR_ODD UART_SETTING_PAR_EVEN ⑤ UART_SETTING_STPB1 UART_SETTING_STPB2	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定された値をUARTの設定に反映する。 ・Rmt_RegisterNotificationGetUartDataにてコールバック登録しておくと、UART有効設定から無効設定するまでの間、受信したデータがコールバックに通知される。
Rmt_GetUartSetting	本体のUARTの設定を取得する。	①BOOL *enable [O] 有効/無効 ②BYTE *baudrate [O] ボーレート ③BYTE *datalength [O] データ長 ④BYTE *parity [O] パリティ ⑤BYTE *stopbit [O] ストップビット ⑥BYTE *FreeMsgCnt [O] 空きメッセージ数	⑥UARTMSG_FREE_MAX	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・現在のUARTの設定内容を返信する。 ・UART有効時(SetUartSettingでenableに有効設定している場合)のみ使用可能。 ・FreeMsgCntは連続実施可能なメッセージ数を示す。 Rmt_SendUartDataを連続して実施する場合は、本変数で可能な件数をチェックし、空きができるまで待機して実施すること。 (FreeMsgCntの最大件数はUARTMSG_FREE_MAX)
Rmt_GetFrameSize	指定したフレームIDのデータサイズ(バイト)を取得する。	①WORD FrameId [I] フレームID ②DWORD *size [O] フレームのデータサイズ(バイト)	①0～4999 内部管理IDのためRmt_GetFrameList or Rmt_ConvertXID2FrameIDで取得した値を使うこと。		・指定したフレームIDのデータサイズ(バイト)を取得する。
Rmt_GetSignalSize	指定したシグナルIDのデータサイズ(バイト)を取得する。	①WORD SignalId [I] シグナルID ②DWORD *size [O] シグナルのデータサイズ(バイト)	①0～29999 内部管理IDのためRmt_GetSignalList or Rmt_ConvertXID2SignalIDで取得した値を使うこと。		・指定したシグナルIDのデータサイズ(バイト)を取得する。
Rmt_DynamicPeriodicFrameControl	定周期送信Frameの有効無効を動的に切り替えする。	①BOOL on_off [I] Flame有効の有無 ②WORD BusId [I] バスID ③BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ④WORD Sid [I] 標準フォーマットのベースID 11ビット ⑤DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット	①TRUE：有効/FALSE：無効 ②RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ③TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ④0x00～0x7FF ⑤0x00～0x3FFFF	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	本APIの制御対象Frameは事前にCANTOOL_PANELのFrame/Signal設定で設定し、CANTOOL AIに反映すること。 引数①をTRUEにした場合、事前に反映した設定どおり定周期送信される。FALSEに設定した場合、対象のFrameの定周期送信が停止される。 制御対象Frameの情報を引数②～⑤で指定すること。 ・SidとEidに範囲外の値が設定された場合の動作は保障外。

DataSend	Rmt_SendFrameData	指定Frameのデータを送信する(周期送信の値も更新される)	①WORD FrameId [I] フレームID ②DWORD size [I] 送信データサイズ(バイト) ③BYTE* data [I] 送信データ	①0～4999 内部管理IDのためRmt_GetFrameList or Rmt_ConvertXID2FrameIDで取得した値を使うこと。 ②0～255 ③②のサイズ分のメモリを確保する。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたFrameIDでデータを送信する。周期送信の値に反映される。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_SendSignalData	指定Signalのデータを送信する(周期送信の値も更新される)	①WORD SignalId [I] シグナルID ②DWORD size [I] 送信データサイズ(バイト) ③BYTE* data [I] 送信データ	①0～29999 内部管理IDのためRmt_GetSignalList or Rmt_ConvertXID2SignalIDで取得した値を使うこと。 ②0～255 ③②のサイズ分のメモリを確保する。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたSignalIDでデータを送信する。周期送信の値に反映される。 ・送信データへのデータ格納について 下詰めでStart位置に上から送信データに格納される (例)Width：9の場合 上 data[0] bit7～bit1:未使用 data[0] bit0→sig data0(Start位置) data[1] bit7→sig data1 data[1] bit6→sig data2 data[1] bit5→sig data3 data[1] bit4→sig data4 data[1] bit3→sig data5 data[1] bit2→sig data6 data[1] bit1→sig data7 下 data[1] bit0→sig data8 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_SendDaData	指定ポートのDAの値を変更する	①BYTE PortNum [I] DAポート番号 ②DWORD value [I] 値	① DA_CHANNEL_0 DA_CHANNEL_1 ②値 = (0.0000～5.0000(V)) * 10000	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたポートのDAの値を変更する。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_SendGpoData	指定ポートのGPOの値を変更する	①BYTE PortNum [I] GPOポート番号 ②BYTE value [I] 値	① GPIO_CHANNEL_0 GPIO_CHANNEL_1 GPIO_CHANNEL_2 GPIO_CHANNEL_3 ② GPIO_BIT_VALUE_LOW GPIO_BIT_VALUE_HIGH	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたポートのGPOの値を変更する。 ・値は、GPIO_BIT_VALUE_LOW,GPIO_BIT_VALUE_HIGHを設定すること。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_SendUartData	UARTへのデータを送信する	①DWORD size [I] 送信データサイズ(バイト) ②BYTE* data [I] 送信データ	①最大データ長：UARTDATA_SEND_SIZE_MAX ②①のサイズ分のメモリを確保	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・送信データをUARTに送信する。 ・送信データの最大データ長は UARTDATA_SEND_SIZE_MAX(単位：Bytes) ・UART有効時(SetUartSettingでenableに有効設定している場合)のみ使用可能。 ・連続送信(本APIの連続実施)を行う場合は、Rmt_GetUartSettingにてFreeMsgCntを取得し格納されてい件数分までを連続実施可能とする。 (FreeMsgCntの最大件数はUARTMSG_FREE_MAX) 不足している場合は、空きができるまで待機して実施すること。
	Rmt_SendAnyCANFrame	任意のCAN Frameデータを送信する	①WORD BusId [I] バスID ②BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ③WORD Sid [I] 標準フォーマットのベースID 11ビット ④DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑥BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑦DWORD DataLength [I] 送信データサイズ(バイト) ⑧BYTE* Data [I] 送信データ	① RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 RMT_BUS_ID_LIN ② TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ③0x00～0x7FF ④0x00～0x3FFFF ⑤ TRUE：CAN-FD FALSE：CAN ⑥ TRUE：有効 FALSE：無効 引数⑤がCAN-FDの場合のみ有効 ⑦ 1～64Byte ⑧ 「⑦」で指定してデータ領域を設定	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたバスにFrameデータを送信する。 単発送信とし周期送信のFrameであっても周期送信データには反映されない。 (周期送信データを更新する場合は、Rmt_SendFrameData/Rmt_SendSignalDataを使用してください) ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_SendCyclicData	送信データをFrameサイズに分割し、指定周期で送信を行う。	①BOOL enable CycleData送信有効/無効 ②WORD cycle 周期時間(ms) ③WORD BusId [I] バスID ④BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑤WORD Sid [I] 標準フォーマットのベースID 11ビット ⑥DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑦BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑧BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑨DWORD DataLength [I] 送信データサイズ(バイト) ⑩BYTE* Data [I] 送信データ	① TRUE：CycleData送信を行う。 FALSE：CycleData送信を停止する。 ② 0～10000(ms) ③ RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ④ TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑤0x00～0x7FF ⑥0x00～0x3FFFF ⑦ TRUE：CAN-FD FALSE：CAN ⑧ TRUE：有効 FALSE：無効 引数⑦がCAN-FDの場合のみ有効 ⑨ 0～1024Byte ⑩ 「⑨」で指定したデータ領域を設定	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたバス、CAN ID等の設定及び、その周期送信の有効/無効を制御する。 ・引数の情報はCANTOOL本体に保存され、次回のCANTOOL本体起動時も設定に従った動作を行う。 ・設定したデータはFrameサイズに分割され、設定した周期で順番に送信される。 ・データの最終まで送信すると、再びデータの先頭から送信を行う。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。 ・本APIはRmt_SendCyclicData2()のnumber=0を使用する。 ・分割してデータ送信している途中で設定をしなおした場合、更新したデータの途中から継続してフレーム送信される。(何分割目か考慮せず上書きする)
	Rmt_SendCyclicData2	送信データをFrameサイズに分割し、指定周期で送信を行う。	①BYTE number CycleData番号 ②BOOL enable CycleData送信有効/無効 ③WORD cycle 周期時間(ms) ④WORD BusId [I] バスID ⑤BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑥WORD Sid [I] 標準フォーマットのベースID 11ビット ⑦DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑧BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑨BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑩DWORD DataLength [I] 送信データサイズ(バイト) ⑪BYTE* Data [I] 送信データ	①0～39 0xFFを設定した場合は登録されている送信がすべて停止する(enable=FALSE) ② TRUE：CycleData送信を行う。 FALSE：CycleData送信を停止する。 ③ 0～10000(ms) ④ RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑤ TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑥0x00～0x7FF ⑦0x00～0x3FFFF ⑧ TRUE：CAN-FD FALSE：CAN ⑨ TRUE：有効 FALSE：無効 引数⑧がCAN-FDの場合のみ有効 ⑩ 0～1024Byte ⑪ 「⑩」で指定したデータ領域を設定	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたCycleData番号毎に、指定されたバス、CAN ID等の設定及び、その周期送信の有効/無効を制御する。 ・引数の情報はCANTOOL本体に保存され、次回のCANTOOL本体起動時も設定に従った動作を行う。 ・設定したデータはFrameサイズに分割され、設定した周期で順番に送信される。 ・データの最終まで送信すると、再びデータの先頭から送信を行う。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。 ・分割してデータ送信している途中で設定をしなおした場合、更新したデータの途中から継続してフレーム送信される。(何分割目か考慮せず上書きする)

Rmt_SendMultiPacketData	MultiPacketデータを送信する	①BYTE fs [I] FS値 ②BYTE bs [I] BS値 ③BYTE STmin [I] 最低送信間隔 ④WORD timeout [I] タイムアウト時間 ⑤WORD BusId [I] バスID ⑥BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦WORD Sid [I] 標準フォーマットのベースID 11ビット ⑧DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪WORD RecvBusId [I] バスID ⑫BOOL RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬WORD RecvSid [I] 標準フォーマットのベースID 11ビット ⑭DWORD RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮BOOL RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯BOOL RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰DWORD DataLength [I] 送信データサイズ(バイト) ⑱BYTE* Data [I] 送信データ	①FS値　MULTI_PACKET_FS_CONTINUOUS_RECEIVE：連続受信可、MULTI_PACKET_FS_WAIT:待機、MULTI_PACKET_FS_OVERFLOW:Overflow ②0～FF　0：全て受信可 ③0～FF(ms) ④0～10000(ms) ⑤RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑥TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑦0x00～0x7FF ⑧0x00～0x3FFFF ⑨TRUE：CAN-FD / FALSE：CAN ⑩TRUE：有効 / FALSE：無効 引数⑨がCAN-FDの場合のみ有効 ⑪RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑫TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑬0x00～0x7FF ⑭0x00～0x3FFFF ⑮TRUE：CAN-FD / FALSE：CAN ⑯TRUE：有効 / FALSE：無効 引数⑮がCAN-FDの場合のみ有効 ⑰1～4095Byte ⑱「⑰」で指定したデータ領域を設定	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・本APIで送受信ポートの情報を設定する。設定しなければRmt_GetMultiPacketStatus()での情報取得はできない。 ・4095Byteより大きい送受信データサイズでの動作は保証しない。 ・マルチパケットのシーケンス完了前に本APIをコールした場合の動作は保証しない。 ・引数①の値ごとの挙動は以下となる。 0:正常ケースであり、APIの動作を継続する。 1:待機状態であることをFCにて送信する。その1秒後に「連続受信可」でFCを送信する。以降は通常のAPIの動作を継続する。 2:相受信に失敗したことをFCにて送信する。本来、TP上はマルチパケット送受信処理は終了するが、APIとしてはデータを受信できる状態でタイムアウトまで待つ。 ・引数①②③は対象へFCパケットで通知する値であり、FC情報は相手から受け取った情報に従う。 ・複数コネクションには対応せず、本APIコール毎に情報は上書きされる。 ・⑤～⑩の情報で⑰～⑱のデータをTP送信し、その後⑪～⑯の情報に一致するデータを待ちTP受信する。 ●想定利用方法 ・Rmt_SendMultiPacketData()で送受信情報を設定し、データを送信する。 ・Rmt_GetMultiPacketStatus()で100ms程度の間隔でポーリングして受信したデータを取得する。 ・Rmt_GetMultiPacketStatus()で送受信の失敗やターゲットの応答タイムアウトを取得する。
Rmt_SendMultiPacketData2	MultiPacketデータを送信する	①BYTE fs [I] FS値 ②BYTE bs [I] BS値 ③BYTE STmin [I] 最低送信間隔 ④WORD timeout [I] タイムアウト時間 ⑤WORD BusId [I] バスID ⑥BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦WORD Sid [I] 標準フォーマットのベースID 11ビット ⑧DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪WORD RecvBusId [I] バスID ⑫BOOL RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬WORD RecvSid [I] 標準フォーマットのベースID 11ビット ⑭DWORD RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮BOOL RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯BOOL RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰DWORD DataLength [I] 送信データサイズ(バイト) ⑱BYTE* Data [I] 送信データ ⑲WORD AddressFormatType [I] ⑳BYTE N_TA	①FS値 MULTI_PACKET_FS_CONTINUOUS_RECEIVE：連続受信可、MULTI_PACKET_FS_WAIT:待機、MULTI_PACKET_FS_OVERFLOW:Overflow ②0～FF MULTI_PACKET_BS_ALL_RECEIVE_POSSIBLE：全て受信可 ③0～FF(ms) ④0～10000(ms) ⑤RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑥TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑦0x00～0x7FF ⑧0x00～0x3FFFF ⑨TRUE：CAN-FD / FALSE：CAN ⑩TRUE：有効 / FALSE：無効 引数⑨がCAN-FDの場合のみ有効 ⑪RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑫TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑬0x00～0x7FF ⑭0x00～0x3FFFF ⑮TRUE：CAN-FD / FALSE：CAN ⑯TRUE：有効 / FALSE：無効 引数⑮がCAN-FDの場合のみ有効 ⑰1～4095Byte ⑱「⑰」で指定したデータ領域を設定 ⑲MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	Rmt_SendMultiPacketData()にAddressFormatTypeとN_TAの引数を追加したもの。 ・本APIで送受信ポートの情報を設定する。設定しなければRmt_GetMultiPacketStatus()での情報取得はできない。 ・4095Byteより大きい送受信データサイズでの動作は保証しない。 ・マルチパケットのシーケンス完了前に本APIをコールした場合の動作は保証しない。 ・引数①の値ごとの挙動は以下となる。 0:正常ケースであり、APIの動作を継続する。 1:待機状態であることをFCにて送信する。その1秒後に「連続受信可」でFCを送信する。以降は通常のAPIの動作を継続する。 2:相受信に失敗したことをFCにて送信する。本来、TP上はマルチパケット送受信処理は終了するが、APIとしてはデータを受信できる状態でタイムアウトまで待つ。 ・引数①②③は対象へFCパケットで通知する値であり、FC情報は相手から受け取った情報に従う。 ・複数コネクションには対応せず、本APIコール毎に情報は上書きされる。 ・⑤～⑩の情報で⑰～⑳のデータをTP送信し、その後⑪～⑯の情報に一致するデータを待ちTP受信する。 ●想定利用方法 ・Rmt_SendMultiPacketData()で送受信情報を設定し、データを送信する。 ・Rmt_GetMultiPacketStatus()で100ms程度の間隔でポーリングして受信したデータを取得する。 ・Rmt_GetMultiPacketStatus()で送受信の失敗やターゲットの応答タイムアウトを取得する。
Rmt_SendMultiPacketData3	MultiPacketデータを送信する	①BYTE fs [I] FS値 ②BYTE bs [I] BS値 ③BYTE STmin [I] 最低送信間隔 ④WORD timeout [I] タイムアウト時間 ⑤WORD BusId [I] バスID ⑥BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦WORD Sid [I] 標準フォーマットのベースID 11ビット ⑧DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪WORD RecvBusId [I] バスID ⑫BOOL RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬WORD RecvSid [I] 標準フォーマットのベースID 11ビット ⑭DWORD RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮BOOL RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯BOOL RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰DWORD DataLength [I] 送信データサイズ(バイト) ⑱BYTE* Data [I] 送信データ ⑲WORD AddressFormatType [I] Addressing format ⑳BYTE N_TA [I] Network Target Address ㉑BYTE PADING [I] CAN Frameデータの空欄(データを詰めた余り)部分に格納する値	①FS値 MULTI_PACKET_FS_CONTINUOUS_RECEIVE：連続受信可、MULTI_PACKET_FS_WAIT:待機、MULTI_PACKET_FS_OVERFLOW:Overflow ②0～FF MULTI_PACKET_BS_ALL_RECEIVE_POSSIBLE：全て受信可 ③0～FF(ms) ④0～10000(ms) ⑤RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑥TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑦0x00～0x7FF ⑧0x00～0x3FFFF ⑨TRUE：CAN-FD / FALSE：CAN ⑩TRUE：有効 / FALSE：無効 引数⑨がCAN-FDの場合のみ有効 ⑪RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑫TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑬0x00～0x7FF ⑭0x00～0x3FFFF ⑮TRUE：CAN-FD / FALSE：CAN ⑯TRUE：有効 / FALSE：無効 引数⑮がCAN-FDの場合のみ有効 ⑰1～4095Byte。0の場合は応答データの送信を行わない。他の動作は同じ。 ⑱「⑰」で指定したデータ領域を設定 ⑲MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF ㉑0x00～0xFF	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	Rmt_SendMultiPacketData2()に㉑を追加したもの。 ・本APIで送受信ポートの情報を設定する。設定しなければRmt_GetMultiPacketStatus()での情報取得はできない。 ・4095Byteより大きい送受信データサイズでの動作は保証しない。 ・マルチパケットのシーケンス完了前に本APIをコールした場合の動作は保証しない。 ・引数①の値ごとの挙動は以下となる。 0:正常ケースであり、APIの動作を継続する。 1:待機状態であることをFCにて送信する。その1秒後に「連続受信可」でFCを送信する。以降は通常のAPIの動作を継続する。 2:相受信に失敗したことをFCにて送信する。本来、TP上はマルチパケット送受信処理は終了するが、APIとしてはデータを受信できる状態でタイムアウトまで待つ。 ・引数①②③は対象へFCパケットで通知する値であり、FC情報は相手から受け取った情報に従う。 ・複数コネクションには対応せず、本APIコール毎に情報は上書きされる。 ・⑤～⑩の情報で⑰～㉑のデータをTP送信し、その後⑪～⑯の情報に一致するデータを待ちTP受信する。 ●想定利用方法 ・Rmt_SendMultiPacketData()で送受信情報を設定し、データを送信する。 ・Rmt_GetMultiPacketStatus()で100ms程度の間隔でポーリングして受信したデータを取得する。 ・Rmt_GetMultiPacketStatus()で送受信の失敗やターゲットの応答タイムアウトを取得する。
Rmt_ResponseMultiPacketData	MultiPacketへの応答内容設定	①BYTE fs [I] FS値 ②BYTE bs [I] BS値 ③BYTE STmin [I] 最低送信間隔 ④WORD timeout [I] タイムアウト時間 ⑤WORD BusId [I] バスID ⑥BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦WORD Sid [I] 標準フォーマットのベースID 11ビット ⑧DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪WORD RecvBusId [I] バスID ⑫BOOL RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬WORD RecvSid [I] 標準フォーマットのベースID 11ビット ⑭DWORD RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮BOOL RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯BOOL RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰DWORD DataLength [I] 送信データサイズ(バイト) ⑱BYTE* Data [I] 送信データ ⑲WORD AddressFormatType [I] xxx ⑳BYTE N_TA [I] xxx	①FS値 MULTI_PACKET_FS_CONTINUOUS_RECEIVE：連続受信可、MULTI_PACKET_FS_WAIT:待機、MULTI_PACKET_FS_OVERFLOW:Overflow ②0～FF MULTI_PACKET_BS_ALL_RECEIVE_POSSIBLE：全て受信可 ③0～FF(ms) ④0～10000(ms) ⑤RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑥TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑦0x00～0x7FF ⑧0x00～0x3FFFF ⑨TRUE：CAN-FD / FALSE：CAN ⑩TRUE：有効 / FALSE：無効 引数⑨がCAN-FDの場合のみ有効 ⑪RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑫TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑬0x00～0x7FF ⑭0x00～0x3FFFF ⑮TRUE：CAN-FD / FALSE：CAN ⑯TRUE：有効 / FALSE：無効 引数⑮がCAN-FDの場合のみ有効 ⑰1～4095Byte。0の場合は応答データの送信を行わない。他の動作は同じ。 ⑱「⑰」で指定したデータ領域を設定 ⑲MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・本APIでマルチパケットへの応答内容を設定する。 ・4095Byteより大きい送受信データサイズでの動作は保証しない。 ・マルチパケットのシーケンス完了前に本APIをコールした場合の動作は保証しない。 ・引数①の値ごとの挙動は以下となる。 0:正常ケースであり、APIの動作を継続する。 1:待機状態であることをFCにて送信する。その1秒後に「連続受信可」でFCを送信する。以降は通常のAPIの動作を継続する。 2:相受信に失敗したことをFCにて送信する。本来、TP上はマルチパケット送受信処理は終了するが、APIとしてはデータを受信できる状態でタイムアウトまで待つ。 ・引数①②③は対象へFCパケットで通知する値であり、FC情報は相手から受け取った情報に従う。 ・複数コネクションには対応せず、本APIコール毎に情報は上書きされる。 ・⑪～⑯の情報に一致するデータをTP受信し、その後⑤～⑩の情報で⑰～⑳のデータをTP送信する。 ●想定利用方法 ・Rmt_SendMultiPacketData()で送受信情報を設定する。 ・ターゲットの機器から受信情報従ったデータを受信データを保持し、設定された送信データを送信する。 ・Rmt_GetResponseMultiPacketDataStatus()で100ms程度の間隔でポーリングして受信したデータを取得する。 ・Rmt_GetResponseMultiPacketDataStatus()で送受信の失敗やターゲットの応答タイムアウトを取得する。

	Rmt_ResponseMultiPacketData3	MultiPacketへの応答内容設定	①BYTE fs [I] FS値 ②BYTE bs [I] BS値 ③BYTE STmin [I] 最低送信間隔 ④WORD timeout [I] タイムアウト時間 ⑤WORD BusId [I] バスID ⑥BOOL SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦WORD Sid [I] 標準フォーマットのベースID 11ビット ⑧DWORD Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨BOOL isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩BOOL BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪WORD RecvBusId [I] バスID ⑫BOOL RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬WORD RecvSid [I] 標準フォーマットのベースID 11ビット ⑭DWORD RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮BOOL RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯BOOL RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰DWORD DataLength [I] 送信データサイズ(バイト) ⑱BYTE* Data [I] 送信データ ⑲WORD AddressFormatType [I] Addressing format ⑳BYTE N_TA [I] Network Target Address ㉑BYTE PADING [I] CAN Frameデータの空欄(データを詰めた余り)部分に格納する値	①FS値 MULTI_PACKET_FS_CONTINUOUS_RECEIVE：連続受信可、MULTI_PACKET_FS_WAIT:待機、 MULTI_PACKET_FS_OVERFLOW:Overflow ②0～FF MULTI_PACKET_BS_ALL_RECEIVE_POSSIBLE：全て受信可 ③0～FF(ms) ④0～10000(ms) ⑤RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑥TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑦0x00～0x7FF ⑧0x00～0x3FFFF ⑨TRUE：CAN-FD / FALSE：CAN ⑩TRUE：有効 / FALSE：無効 引数⑨がCAN-FDの場合のみ有効 ⑪RMT_BUS_ID_CAN0 RMT_BUS_ID_CAN1 RMT_BUS_ID_CAN2 RMT_BUS_ID_CAN3 ⑫TRUE：ベースIDのみ FALSE：ベースID＋拡張ID ⑬0x00～0x7FF ⑭0x00～0x3FFFF ⑮TRUE：CAN-FD / FALSE：CAN ⑯TRUE：有効 / FALSE：無効 引数⑮がCAN-FDの場合のみ有効 ⑰1～4095Byte。0の場合は応答データの送信を行わない。他の動作は同じ。 ⑱「⑰」で指定したデータ領域を設定 ⑲MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF ㉑0x00～0xFF	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	Rmt_ResponseMultiPacketData()に㉑を追加したもの。 ・本APIでマルチパケットへの応答内容を設定する。 ・4095Byteより大きい送受信データサイズでの動作は保証しない。 ・マルチパケットのシーケンス完了前に本APIをコールした場合の動作は保証しない。 ・引数①の値ごとの挙動は以下となる。 0:正常ケースであり、APIの動作を継続する。 1:待機状態であることをFCにて送信する。その1秒後に「連続受信可」でFCを送信する。以降は通常のAPIの動作を継続する。 2:相受信に失敗したことをFCにて送信する。本来、TP上はマルチパケット送受信処理は終了するが、APIとしてはデータを受信できる状態でタイムアウトまで待つ。 ・引数①②③は対象へFCパケットで通知する値であり、FC情報は相手から受け取った情報に従う。 ・複数コネクションには対応せず、本APIコール毎に情報は上書きされる。 ・⑪～⑯の情報に一致するデータをTP受信し、その後⑤～⑩の情報で⑰～㉑のデータをTP送信する。 ●想定利用方法 ・Rmt_SendMultiPacketData()で送受信情報を設定する。 ・ターゲットの機器から受信情報に従った受信データを保持し、設定された送信データを送信する。 ・Rmt_GetResponseMultiPacketDataStatus()で100ms程度の間隔でポーリングして受信したデータを取得する。 ・Rmt_GetResponseMultiPacketDataStatus()で送受信の失敗やターゲットの応答タイムアウトを取得する。
DataRecieve	Rmt_GetFrameData	指定Frameの最新データを取得する	①WORD FrameId [I] フレームID ②DWORD size [I] 受信フレームデータサイズ(バイト) ③BYTE* data [O] 受信フレームデータ ④ULONGLONG* Time [O] 最終更新時タイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間))	①0～4999 内部管理IDのためRmt_GetFrameList or Rmt_ConvertXID2FrameIDで取得した値を使うこと。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたFrameIDの最新データを取得する。 ・一度も受信していない場合は、戻り値がRET_NOTFOUND、受信フレームデータはNULL、最終更新時タイムスタンプは0となる。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_GetSignalData	指定Signalの最新データを取得する	①WORD SignalId [I] シグナルID ②DWORD size [I] 受信シグナルデータサイズ(バイト) ③BYTE* data [O] 受信シグナルデータ ④ULONGLONG* Time [O] 最終更新時タイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間))	①0～29999 内部管理IDのためRmt_GetSignalList or Rmt_ConvertXID2SignalIDで取得した値を使うこと。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたSignalIDの最新データを取得する。 ・一度も受信していない場合は、戻り値がRET_NOTFOUND、受信フレームデータはNULL、最終更新時タイムスタンプは0となる。 ・data格納について 下詰めで上からStart位置よりdataに格納される (例)Width：9の場合 上 data[0] bit7～bit1:未使用 data[0] bit0←sig data0(Start位置) data[1] bit7←sig data1 data[1] bit6←sig data2 data[1] bit5←sig data3 data[1] bit4←sig data4 data[1] bit3←sig data5 data[1] bit2←sig data6 data[1] bit1←sig data7 下 data[1] bit0←sig data8 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_GetAdData	指定ポートのADの値を取得する	①BYTE PortNum [I] ADポート番号 ②DWORD* value [O] 値 ③ULONGLONG* Time [O] 最終更新時タイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間))	① AD_CHANNEL_0 AD_CHANNEL_1 ②値 = (0.0000～50.0000(V)) * 10000	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたADポートの最新データを取得する。 ・値は、AD電圧値0V～50Vを1万倍した値が取得できる。 ・一度も受信していない場合は、戻り値がRET_NOTFOUND、値はNULL、最終更新時タイムスタンプは0となる。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_GetGpiData	指定ポートのGPIの値を取得する	①BYTE PortNum [I] GPIポート番号 ②BYTE* value [O] 値 ③ULONGLONG* Time [O] 最終更新時タイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間))	① GPIO_CHANNEL_0 GPIO_CHANNEL_1 GPIO_CHANNEL_2 GPIO_CHANNEL_3 ② GPIO_BIT_VALUE_LOW GPIO_BIT_VALUE_HIGH	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定されたGPIポートの最新データを取得する。 ・値は、GPIO_BIT_VALUE_LOW,GPIO_BIT_VALUE_HIGHが取得できる。 ・一度も受信していない場合は、戻り値がRET_NOTFOUND、値はNULL、最終更新時タイムスタンプは0となる。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_GetDataNewerList	CANTOOL本体が送受信したデータの最新情報のみを取得する。	①DWORD size [I] 最大取得サイズ(バイト) ②log_record_dt_t_rmt* data [O] 最新データ(log_record_dt_t_rmt構造体の配列) ③DWORD* ActualOut [O] 取得件数	① 「sizeof(log_record_dt_newer_list_max_t_rmt)」を固定でセットすること。 ②log_record_dt_newer_list_max_t_rmt構造体のメモリを確保すること。 ③引数の取得件数分、引数②にデータが格納される。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・CANTOOL本体が送受信したデータの最新情報のみを取得する。 ※受信したデータは随時上書きされるため、値の変化を確認する用途には向かない。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_GetDataALL	CANTOOL本体が送受信した全データを取得する。	①BOOL StartEnd [I] 開始/終了 ②DWORD size [I] 最大取得サイズ(バイト) ③log_record_dt_t_rmt* data [O] 全データ(log_record_dt_t_rmt構造体の配列) ④DWORD* ActualOut [O] 取得件数	①TRUE：蓄積停止 FALSE：蓄積開始 ②全データ格納サイズは、「sizeof(log_record_dt_get_data_all_max_t_rmt)」を固定でセットすること。 ③log_record_dt_get_data_all_max_t_rmt構造体のメモリを確保する。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・CANTOOL本体が送受信した全データを取得する。 本APIを定期的にコールし、CANTOOL本体で蓄積している送受信した全データを取得することを想定している。 ※インターフェース設定に左右されるが、50ms以内の周期で取得することを推奨する。 ・引数①にFALSEを設定すると、送受信データの蓄積を開始する。 送受信データ取得中は、引数①はFALSEを設定する。 ※初回はログの蓄積を開始するトリガであるため、引数の取得件数は0となる。2回目以降の呼び出しで全データが取得できる。 ・引数①にTRUEを設定すると、送受信データの蓄積を停止し、データを破棄する。 StartEndにTRUEを設定せず本APIコールを停止すると、パケットのロストが発生する。(CANTOOL-PANELに表示される) ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。 ・引数④の件数分、引数③にデータが格納される。
	Rmt_GetDateTimeText	受信したデータのタイムスタンプを文字列で取得する	①ULONGLONG* [I] timeSerial ②char* [O] timeChar	①タイムスタンプ ②時間文字列(形式：YYYY/MM/DD HH:MM:SS.mmm.uuu.n) 2000/1/1～2500/1/1間のタイムシリアル値を設定する。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・GetFrameData等、DataRecieveのAPIで取得したタイムスタンプをシリアル値から文字列に変換する。 ・ExcelVBAでは、ULONGLONG型(8バイト整数データ)が扱えない為、本APIを使用してタイムスタンプを取得する。
	Rmt_GetMultiPacketStatus	MultiPacket状態を取得する	①BYTE* DataState [O] ②DWORD* DataLength [O] 受信データサイズ(バイト) ③BYTE* Data [O] 受信データ	① 「MULTI_PACKET_STATE_～」の定義を参照 0:データがある 1:データがない 10:送信エラー(タイムアウト) 11:送信エラー(内部エラー) 12:送信エラー(シーケンス異常) 20:受信エラー(タイムアウト) 21:受信エラー(内部エラー) 22:受信エラー(シーケンス異常) 30:FC情報 ② 1～4095Byte データがない場合は0 ③ 4095Byteのメモリを確保すること。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・マルチパケット送受信の状態を取得することができる。 ・先にRmt_SendMultiPacketData()で送受信ポートの設定を行っておく必要がある。 ・送受信データサイズが4095Byte以上となる環境での動作は保証しない。 ・状態は1バッファのみ保持する。Get前に新しい状態に変化すると情報を上書きする。 ・本APIをコールすると情報を破棄し、①が「1」となる。 ・①が「データがある」と「FC情報」の場合のみ②と③は有効。 「30」の時は受信したFCのデータが格納される。 1Byte目：FS値、2Byte目：BS値、3Byte目：STmin値 ・FCデータのFS値ごとの挙動は以下となる。 0:正常な応答であり、APIの動作を継続する。 1:相手の処理待ち状態であり、処理待ちが解除され次第APIの動作を継続する。 2:相手側で受信に失敗した状態であり、TP上はマルチパケット送受信処理は終了する。ただしAPIとしてはデータを受信できる状態でタイムアウトまで待つ。

	Rmt_GetResponseMultiPacketDataStatus	MultiPacketの応答結果を取得する	①BYTE* DataState [O] ②DWORD* DataLength [O] 受信データサイズ(バイト) ③BYTE* Data [O] 受信データ	① 「MULTI_PACKET_STATE_～」の定義を参照 0:データがある 1:データがない 10:送信エラー(タイムアウト) 11:送信エラー(内部エラー) 12:送信エラー(シーケンス異常) 20:受信エラー(タイムアウト) 21:受信エラー(内部エラー) 22:受信エラー(シーケンス異常) 30:FC情報 ② 1～4095Byte データがない場合は0 ③ 4095Byteのメモリを確保すること。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・マルチパケットの応答結果状態を取得することができる。 ・先にRmt_ResponseMultiPacketData()で送受信ポートの設定を行っておく必要がある。 ・受信データサイズが4095Byte以上となる環境での動作は保証しない。 ・状態は1バッファのみ保持する。Get前に新しい状態に変化すると情報を上書きする。 ・本APIをコールすると情報を破棄し、①が「1」となる。 ・①が「データがある」と「FC情報」の場合のみ②と③は有効。 「30」の時は受信したFCのデータが格納される。 1Byte目：FS値、2Byte目：BS値、3Byte目：STmin値 ・FCデータのFS値ごとの挙動は以下となる。 0:正常な応答であり、APIの動作を継続する。 1:相手の処理待ち状態であり、処理待ちが解除され次第APIの動作を継続する。 2:相手側で受信に失敗した状態であり、TP上はマルチパケット送受信処理は終了する。ただしAPIとしてはデータを受信できる状態でタイムアウトまで待つ。
Logging	Rmt_StartRecordingLog	ログ記録を開始する(PCロギング)	なし	なし	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・ログ記録を開始する。 ・プロジェクト設定ファイルに記載されているPCのパスのフォルダに記録される。 ・ログ分割サイズ毎に分割し保存される。 ・開始時や記録中に、ファイルの書き込み失敗やメディアフルで、ログ記録を終了した場合は、Rmt_RegisterNotificationEndedRecordingLogで登録した関数がコールバックされる。 ・すでにロギングが開始されている時に、本APIをコールした場合は正常を返し、実行中のロギングを継続する。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_EndRecordingLog	ログ記録を終了する(PCロギング)	なし	なし	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・ログ記録を終了する。 ・終了後、Rmt_RegisterNotificationEndedRecordingLogで登録した関数がコールバックされる。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_StartPlayingFile()	指定した再生ファイルを実行する (PCでの再生)	①char* PlayFile [I] ファイルパス	①・フルパス、UTF-8で指定する。フォルダ区切り文字は「¥¥」。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定した再生ファイルの再生を開始する。 ・再生中の周期送信有無はログファイルの「Cyclic Send」の内容に従い動作する。 ・再生位置は指定ファイルの先頭から実施される。 ・再生中は、NotificationPlayingFileProgressで再生しているログのタイムスタンプが通知される。 ・Rmt_EndPlayingFileをコールする前に再生が終了した場合は、NotificationEndedPlayingFileが通知される。 ・ファイルパスは、UTF-8で指定すること。
	Rmt_EndPlayingFile()	再生ファイルの実行を終了する (PCでの再生)	なし	なし	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・再生ファイルの再生を終了する。 ・終了後、NotificationEndedPlayingFileが通知される。
	Rmt_LoadMasterSetting	プロジェクトファイルの設定を、CANTOOL本体に反映する	なし	なし	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・プロジェクトファイルに設定されている、インターフェース設定、ログ設定、フィルタ設定、再生データを本体に反映する。 ・非同期で処理するため結果は、NotificationEndMasterSettingで通知される。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_LoadMasterSetting2	指定のインターフェース設定ファイルを、CANTOOL本体に反映する	①DWORD size ②char* PlayFile [I] ファイルパス	①0～255Byte ②・フルパス、UTF-8で指定する。フォルダ区切り文字は「¥¥」。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・プロジェクトファイルに設定されている、ログ設定、フィルタ設定、再生データと、引数で指定した、インターフェース設定を本体に反映する。 ・CANTOOL_PANELとしては一時的な反映であり、GUI上もプロジェクトファイル上も反映されない。CANTOOL A1としては反映されたインターフェース設定ファイルを本体に記録し、再起動後も反映後の状態で動作する。 ・非同期で処理するため結果は、NotificationEndMasterSettingで通知される。 ・指定したパスが読み込めなければ処理失敗となる。 ・sizeとCdbPathの整合性が合わない場合の挙動は保証しない。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_TimeSyncronizeRequest	CANTOOL本体にPCの時刻を設定する。	なし	なし	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・CANTOOL本体にPCの時間を設定する。(モニタやログのタイムスタンプがPCとずれることを防ぐため設定する) ・ユーザーアプリから設定を行わず、PCとCANTOOL本体と5秒以上の時間のずれが発生した場合は、CANTOOL-PANELにより時刻設定ダイアログが表示される。ダイアログ表示中にユーザーアプリから時刻設定を行っても問題ない。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
APP Control	Rmt_StartAppFunction	CANTOOL PANELの各画面を起動する。	①BYTE DispId [I] 画面ID	①「ユーザーアプリから起動する画面ID」シートを参照。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定された画面IDに対応したCANTOOL-PANELの各画面を起動する。 ・指定できる画面IDは、「ユーザーアプリから起動する画面ID」シートを参照。
	Rmt_StartAppFunction2	CANTOOL PANELの各画面を起動する。	①BYTE DispId [I] 画面ID ②BOOL IsShow[I] 表示、非表示 ③char* XmlFile [I] ファイルパス	①「ユーザーアプリから起動する画面ID」シートを参照。 ②TRUE：表示 FALSE：非表示 ③フルパス、UTF-8で指定する。フォルダ区切り文字は「¥¥」。指定しない場合はNULL。	RET_OK:正常、負数:エラー エラーは[エラー一覧]シートを参照。	・指定された画面IDに対応したCANTOOL-PANELの各画面を起動する。 ・指定できる画面IDは、「ユーザーアプリから起動する画面ID」シートを参照。
Status	Rmt_RegisterNotificationStatusChange	状態変更時に呼び出されるコールバック関数を登録する	①TNotificationStatusChange NotificationStatusChange [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationStatusChangeで設定した関数	状態変更時に通知される	なし		RET_OK:正常	・Rmt_RegisterNotificationStatusChange()でコールバック関数を登録すること。 ・GetStatus()で取得できる各種状態が変化した際に通知される。
Control	Rmt_RegisterNotificationUserSW	ユーザーSW操作時に呼び出されるコールバック関数を登録する	①TNotificationUserSW NotificationUserSW [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationUserSWで設定した関数	ユーザーSW操作時に通知される。	①BYTE UserSW [I] ユーザースイッチ番号 ②BYTE SwStatus [I] 状態	① USERSW_NUM_LOGGING USERSW_NUM_PLAY1 USERSW_NUM_PLAY2 USERSW_NUM_PLAY3 ② BUTTON_STATUS_RELEASE BUTTON_STATUS_PUSH_SHORT BUTTON_STATUS_PUSH_LONG	RET_OK:正常	・Rmt_RegisterNotificationUserSW()でコールバック関数を登録すること。 ・ユーザーSW操作時に、ユーザースイッチ番号と状態が通知される。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_RegisterNotificationConfigChange	Config変更時に呼び出されるコールバック関数を登録する	①TNotificationConfigChange NotificationConfigChange [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationConfigChangeで設定した関数	Config変更時に通知される。	なし		RET_OK:正常	・Rmt_RegisterNotificationConfigChange()でコールバック関数を登録すること。 ・Config変更時に通知されるため、必要に応じて情報を読み直すこと。
	Rmt_RegisterNotificationStatusMonitorDataClear	ステータスモニタ用ログクリア時に呼び出されるコールバック関数を登録する。	①TNotificationStatusMonitorDataClear NotificationStatusMonitorDataClear [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationStatusMonitorDataClearで設定した関数	ステータスモニタ用ログをクリア時に通知される。	なし		RET_OK:正常	・Rmt_RegisterNotificationStatusMonitorDataClear()でコールバック関数を登録すること。 ・ステータスモニタ用ログがクリアされた時に通知される。(複数のユーザープログラム使用時に、他のユーザープログラムにて設定を変更したことが分かる)

DataRecieve	Rmt_RegisterNotificationGetUartData	UARTのデータを受信した際に呼び出されるコールバック関数を登録する。	①TNotificationGetUartData NotificationGetUartData [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_NotificationGetUartDataで設定した関数	UARTの受信データが通知される。	①BYTE status [I] 状態 ②DWORD size [I] データ長 ③BYTE* data [I] UARTからの取得データ	① GET_UARTDATA_STATE_COMPLETE GET_UARTDATA_STATE_FAILED ② UARTDATA_GET_SIZE_MAX	RET_OK:正常	・Rmt_RegisterNotificationGetUartDataでコールバック関数を登録すること。 ・UART有効の間(SetUartSettingでenableに有効設定して無効に設定するまでの間)、UARTから受信したデータを通知する。 ・最大でUARTDATA_GET_SIZE_MAX(単位:byte)取得される。
Logging	Rmt_RegisterNotificationEndedRecordingLog	ログ記録の状態変化時に呼び出されるコールバック関数を登録する。	①TNotificationEndedRecordingLog NotificationEndedRecordingLog [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationEndedRecordingLogで設定した関数	StartRecordingLog()、EndRecordingLog()の応答として、ログ記録の実行終了を通知する。	①BYTE Status [I] 状態	① ENDEDRECORDINGLOG_STATE_COMPLETE ENDEDRECORDINGLOG_STATE_MEDIA_FULL ENDEDRECORDINGLOG_STATE_FAILED	RET_OK:正常	・Rmt_RegisterNotificationEndedRecordingLog()でコールバック関数を登録すること。 ・StartRecordingLog()、EndRecordingLog()の応答として、ログ記録の実行終了時に、終了時の状態が通知される。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。
	Rmt_RegisterNotificationLostStatus	ロギング時のロスト発生時に呼び出されるコールバック関数を登録する。	①TNotificationLostStatus NotificationLostStatus [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationLostStatusで設定した関数	ロギング時、ロストの発生状態を通知する。	①BOOL Lost [I] ロスト発生状態 ②ULONGLONG OccurrenceTime [I] 発生時のタイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間)) ③ULONGLONG ClearedTime [I] 解消時のタイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間)) ④BYTE OccurrencePosition [I] 発生箇所(00:FPGA、02:本体アプリ、10:PCアプリ) ⑤BYTE BusKind [I] ユニット ⑥BYTE channel [I] チャンネル	④ LOST_POINT_FPGA LOST_POINT_CANTOOL LOST_POINT_PC ⑤ UNIT_CAN UNIT_LIN UNIT_AD UNIT_DA UNIT_GPI UNIT_GPO UNIT_CAN_FD ⑥⑤の値によって以下の範囲とななる。 UNIT_CAN,UNIT_CAN_FDの場合 CAN_CHANNEL_0 CAN_CHANNEL_1 CAN_CHANNEL_2 CAN_CHANNEL_3 UNIT_ADの場合 AD_CHANNEL_0 AD_CHANNEL_1 UNIT_DAの場合 DA_CHANNEL_0 DA_CHANNEL_1 UNIT_GPI,UNIT_GPOの場合 GPIO_CHANNEL_0 UNIT_LINの場合 channel未使用(0固定)	RET_OK:正常	・Rmt_RegisterNotificationLostStatus()でコールバック関数を登録すること。 ・ロギング時にロストの発生状態を通知する。
Play	Rmt_RegisterNotificationEndedPlayingFile	再生状態の完了時に呼び出されるコールバック関数を登録する。	①TNotificationEndedPlayingFile NotificationEndedPlayingFile [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationEndedPlayingFileで設定した関数	Rmt_StartPlayingFile()、Rmt_EndPlayingFile()の応答として、再生ファイルの実行終了を通知する。 (PCによる再生)	①BYTE Status [I] 状態	① ENDEDPLAYINGFILE_STATE_COMPLETE ENDEDPLAYINGFILE_STATE_FAILED	RET_OK:正常	・Rmt_RegisterNotificationEndedPlayingFile()でコールバック関数を登録すること。 ・Rmt_StartPlayingFile()、Rmt_EndPlayingFile()の応答として、再生ファイルの再生終了時に、終了時の状態が通知される。
	Rmt_RegisterNotificationPlayingFileProgress	再生開始時に呼び出されるコールバック関数を登録する	①TNotificationPlayingFileProgress NotificationPlayingFileProgress [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる
	Rmt_RegisterNotificationPlayingFileProgressで設定した関数	再生ファイルの再生位置を通知する。	①ULONGLONG timestamp [I] タイムスタンプ((単位：100ns 1601-01-01 00:00:00を基準とした経過時間))		RET_OK:正常	・Rmt_RegisterNotificationPlayingFileProgress()でコールバック関数を登録すること。 ・開始時に再生位置のタイムスタンプが通知される。 ・タイムスタンプ値は、再生ファイル内の時間です。
Setting	Rmt_RegisterNotificationEndMasterSetting	本体へのプロジェクト設定完了時に呼び出されるコールバック関数を登録する。	①TNotificationEndMasterSetting NotificationEndMasterSetting [I] コールバックする関数のポインタ		RET_OK:正常	・呼び出されるコールバック関数を登録する。 ・パラメータ①にCALLBACK_CLEARを指定することでコールバック登録情報がクリアされる。
	Rmt_RegisterNotificationEndMasterSettingで設定した関数	本体へのプロジェクト設定完了を通知する。	①BYTE MasterSettingKind [I] MasterSetting種別 ②BYTE Status [I] 状態	① ENDMASTERSETTING_KIND_LOAD ENDMASTERSETTING_KIND_SEND ENDMASTERSETTING_KIND_GET ② ENDMASTERSETTING_STATE_COMPLETE ENDMASTERSETTING_STATE_FAILED ENDMASTERSETTING_STATE_NO_FILE ENDMASTERSETTING_STATE_NO_SD	RET_OK:正常	・Rmt_RegisterNotificationEndMasterSetting()でコールバック関数を登録すること。 ・本体へのプロジェクト設定完了時に、MasterSetting種別と状態が通知される。 ・summary_t_rmt構造体の「USB通信状態(本体-PC間)」が「USB_STATUS_DISCONNECT」の場合は利用不可。

●ユーザーアプリから起動する画面ID

画面ID	画面名		補足	備考
DISP_ID_STATUS	ステータスモニタ	Status Monitor		
DISP_ID_SEQ	シーケンスモニタ	Sequence Monitor		
DISP_ID_LOG	ログ操作画面	Log		
DISP_ID_SYSLOG	システムログ画面	System Log		
DISP_ID_PROJECT	プロジェクト設定	Setting		
DISP_ID_EMU	エミュレーションシグナルデータ変更画面	Change Value(All)		
DISP_ID_SIGNAL	シグナルデータ変更(カスタム)起動画面	Change Value(Custom)	エミュレーションパネル開くか新規を選択する画面が起動するので、選択する	複数機能可能。ただし、選択画面が表示されている場合には新たに画面は開かれないため、選択完了後に画面を呼び出してください。
DISP_ID_PLAY	再生データ操作画面	Playback		
DISP_ID_SUPPOR	ユーザーサポート画面	Support		

●エラー一覧

エラーコード	意味	失敗の原因	対処法
RET_PARAM_ERR	パラメータ不正	引数の指定内容が不正	・引数内容が仕様の範囲かを確認する
RET_DISCONNECT	CANTOOL本体未接続	CANTOOL本体がPC(CANTOOL PANEL)と接続されていない	・本体を接続する ・本体の電源が入っていることを確認する
RET_LENGTH_ERR	データ長不正	データ長の指定が対応するデータと異なる	・データ長とデータを一致するよう修正する
RET_ERROR	異常終了	状態異常、システム異常、ファイル生成失敗、指定するデータがない等	・PCのリソースやパスの指定が正しいか確認する ・関連するAPIが同時にコールされていないか確認する ・指定するデータと設定に相違がないかを確認する
RET_INVALID	データ不正	文字列が変換不能、変換後のファイルパスが異常など	・文字列の指定内容が正しいか確認する
RET_NOTFOUND	データなし	指定したデータやファイルが無い	・データやファイルの指定が正しいか確認する
RET_OUTOFMEMORY	領域確保失敗	PC上で処理のためのメモリが確保できなかった	・PCを再起動してから実施する ・PCのメモリを増設する