V016002-703

CANTOOL-A シリーズ

CANTOOL A1

[CANTOOL Script-Manual]

Rev. 01.02 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	2018-03-26	初版(対応全体バージョン：Ver01.03)
01.01	2019-11-05	対応全体バージョン：Ver01.04 ・MultiPacket 送受信 API 追加 ・Python3.7.3 に対応(python_env_ver01.00.03)
01.02	2022-08-31	対応全体バージョン：Ver01.05 ・UART 送信 API 追加

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 Script"は CANTOOL A1 が提供する機能です。
CANTOOL A1 の CAN、AD/DA、GPIO などを Python によって制御することができます。
CANTOOL Script は、Windows 上で実行することも、CANTOOL A1 に Python スクリプトを設定してスタンドアロンで実行することもできます。

本書では、以降 CANTOOL A1 を"CANTOOL" と称します。

本書は、「V016002-700_CANTOOL_A1-Manual」「V016002-702_CANTOOL_A1_SDK-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 Script は、CANTOOL の I/O などを制御するための制御用関数ライブラリを提供します。
提供する制御用関数ライブラリを使用することにより、制御用アプリケーションと内部通信を行い Python スクリプトに従った制御を行います。

以下にシステム構成を示します。

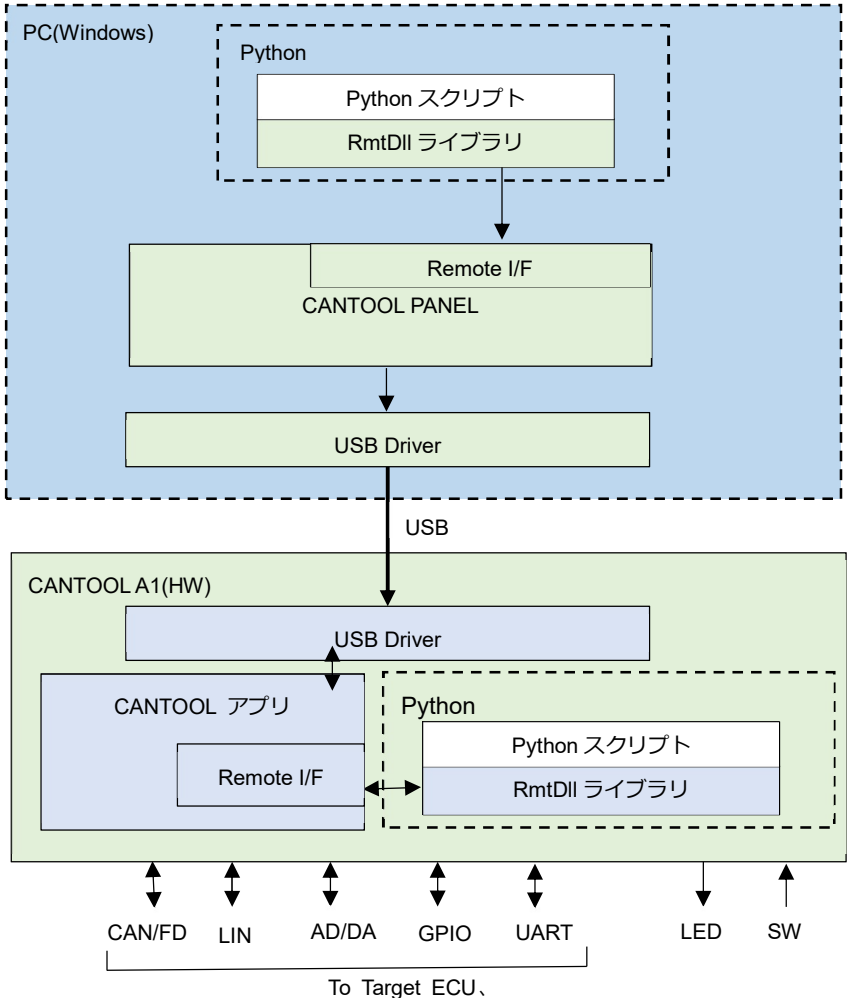
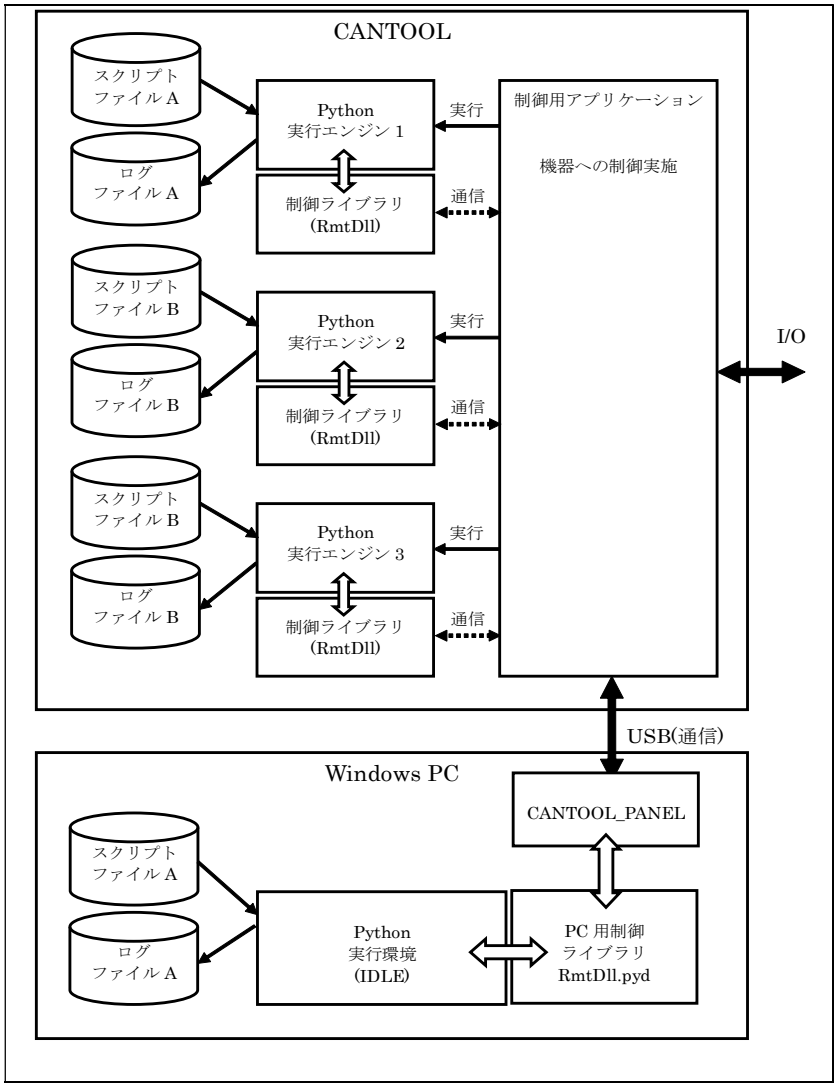


図 1-1 CANTOOL システム構成図

表 1-1 システム構成要素一覧

名称	説明
Python スクリプト	ユーザーにて作成される CANTOOL 制御用 Python のスクリプトコードです。
RmtDll ライブラリ	CANTOOL を制御するための API が含まれたライブラリです。Python スクリプトで Import して利用します。
CANTOOL PANEL、 CANTOOL アプリ	CANTOOL 基本ソフトウェアです。
Remote I/F	RmtDll ライブラリとのインターフェースです。
USB Driver	CANTOOL PANEL と CANTOOL アプリが通信するためのドライバです。

CANTOOL および Script のソフトウェア構成を以下に示します。



Python スクリプトを、CANTOOL のユーザスイッチに割り当て、スタンドアロンで実行させることができます。

Python 実行環境をインストールした Windows PC 上で実行することができます。IDLE などのデバッガで Python スクリプトをデバッグすることもできます。

図 1-2 CANTOOL ソフトウェア構成図

表 1-2 ソフトウェア構成要素一覧

名称	説明
スクリプトファイル	ユーザーが作成する Python スクリプトです。
Python 実行環境(IDLE)	Python をインストールすることで利用可能なソフトウェアです。
PC 用制御ライブラリ (RmtDll.pyd)	CANTOOL を制御するための API が含まれたライブラリです。 Windows 上で Python から利用されます。
CANTOOL PANEL	CANTOOL 基本ソフトウェアです。
Python 実行エンジン	CANTOOL 上で Python スクリプトを実行するソフトウェアです。
制御ライブラリ(RmtDll)	CANTOOL を制御するための API が含まれたライブラリです。 CANTOOL 上で Python 実行エンジンから利用されます。

1.2 稼働条件

対象オペレーティングシステム

CANTOOL Script 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

動作確認済み開発環境

CANTOOL Script は、以下の開発環境を使用した動作を確認済みです。

・Python 3.5.4

2. CANTOOL Script について

2.1 CANTOOL Script の構成

CANTOOL Script として以下のものが用意されています。

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。

Script-root	
|- Manual	CANTOOL Script マニュアル（本書）
|- Library	ライブラリ
|-　|- RmtDll.pyd	Windows 上での動作用
|- Header	ヘッダーファイル
|-　|- ctdef	スクリプト関数用定義値
|- Sample	各種サンプルプログラム
|- readme_001R_sample.pdf	サンプル 1 の説明
|- :	
|- 001R_sample	サンプル 1（Python）　＊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　R＝Python 用

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 CANTOOL Script で扱うインターフェース情報ついて

「V016002-703_CANTOOL_A1_Script-ReferenceManual-API」を参照してください。

2.3 CANTOOL Script 利用時の作業流れ

(1)Pythonスクリプトファイル作成する。

テキストエディタ等により、スクリプトファイルを作成する。

処理結果等を print 命令で、実行ログとして出力する。

(2) Pythonスクリプトファイルのデバッグを行う。

Windows PC にて、Python スクリプトファイルをデバッグ実行して動作確認を行う。

デバッグ実行は Python に標準で入っている、IDEL ツール等でステップ実行や連続実行して確認する。

(3) PythonスクリプトファイルをCANTOOLに反映させる。

CANTOOL PANEL を使用し、ユーザスイッチに割り当てるスクリプトを設定し、反映を行う。

(4)CANTOOLのユーザスイッチ操作でスクリプトファイルを実行する。

Python スクリプトファイルを割り当てたユーザスイッチを押して、Python スクリプトファイルを実行する。

(5)実行状態は、CANTOOL3のLED点灯で確認する。

CANTOOL 本体の LED の点灯状態により、実行中/正常終了/異常終了を確認する。

(6)実行結果をログファイルで確認する。

実行終了後、CANTOOL　PANEL で本体ログを取得し、実行結果をログファイルにより確認する。

3. 使用方法

3.1 CANTOOL Script 使用準備

ここでは Windows 上で Python スクリプトを作成・デバッグするための準備を行います。

3.1.1 Python のインストール

使用する Python のバージョンは「３．７．３」です。(Python 環境：Ver01.00.03 時)

(1) Python のインストールモジュールをダウンロードする。

以下のダウンロード用 URL へブラウザでアクセスする。

URL: https://www.python.org/downloads/

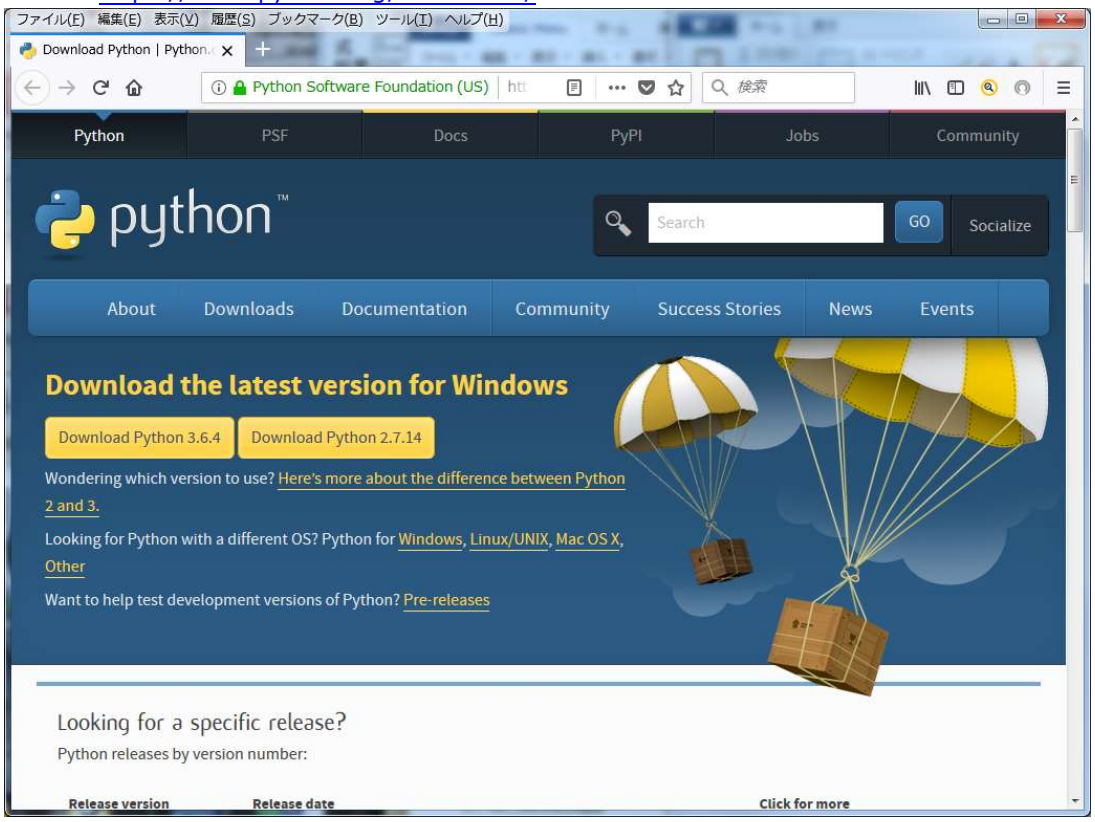


図 3-1 Python3.5.4 ダウンロード(1/4)

「Downloads」の選択から、「Windows」を選択してバージョン一覧を表示する。

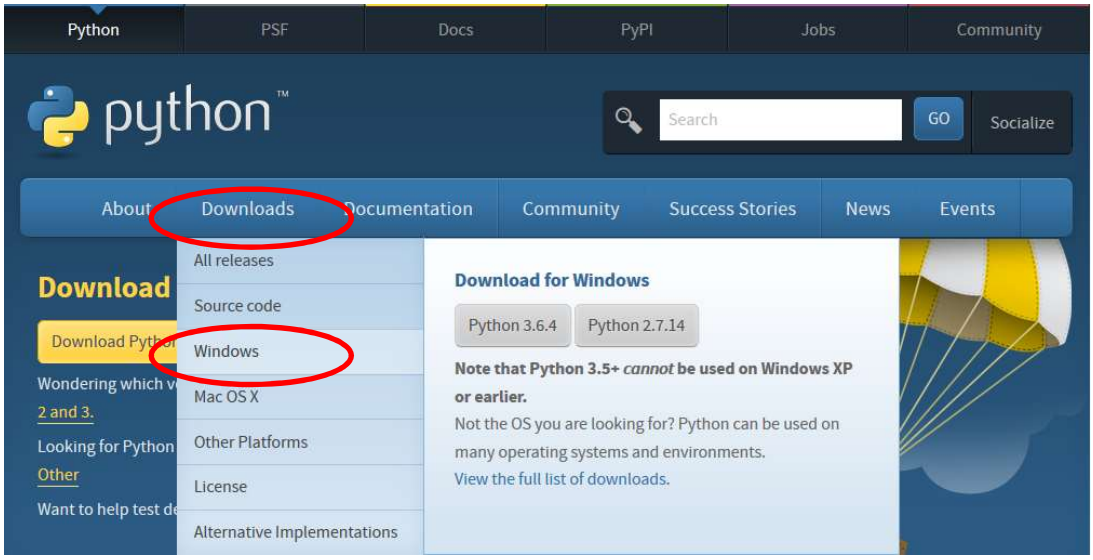
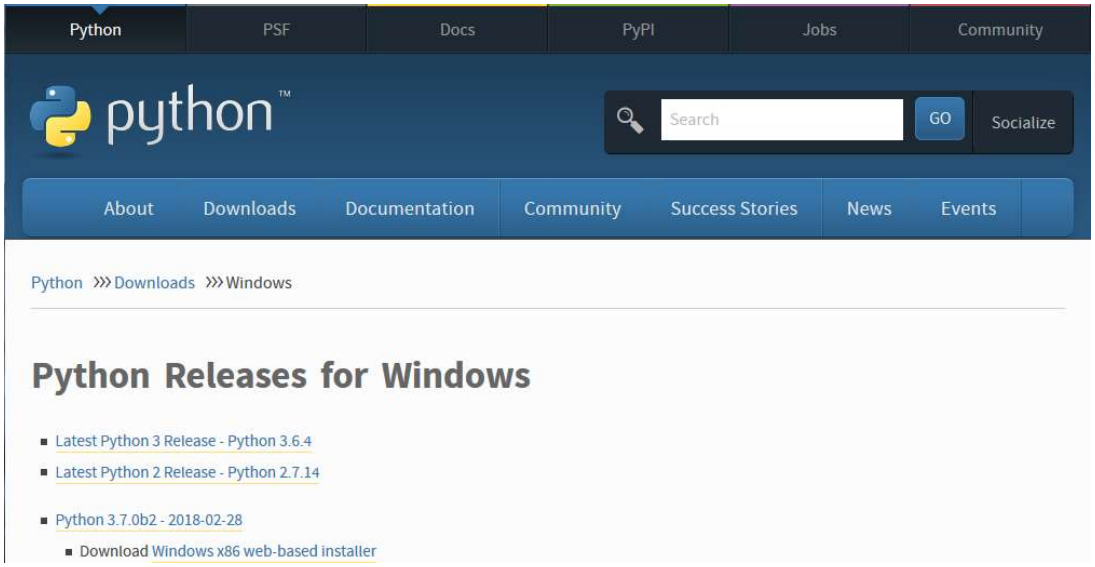


図 3-2 Python3.5.4 ダウンロード(2/4)

「Python 3.5.4 - 2017-08-08」の、「Download　Windows x86 executable installer」(32ビット版)を選択する。



（中略）

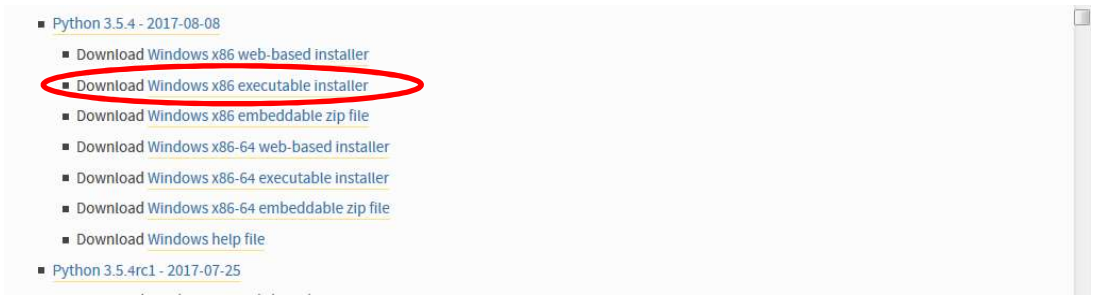


図 3-3 Python3.5.4 ダウンロード(3/4)

「ファイルを保存」を選択にしてファイルに保存する。

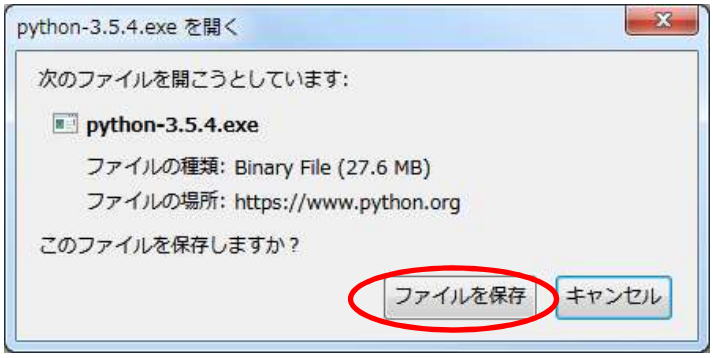


図 3-4 Python3.5.4 ダウンロード(4/4)

(2)Pythonをインストールする。

ダウンロードした「python-3.5.4.exe」を開き、「実行」を選択してインストールを実行する。

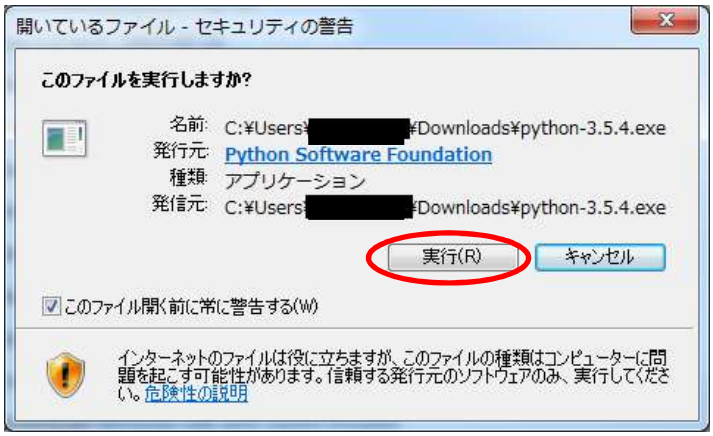


図 3-5 Python3.5.4 インストール(1/6)

「Add Python 3.5 to PATH」をチェックして、「Install Now」を選択してインストールを開始する。

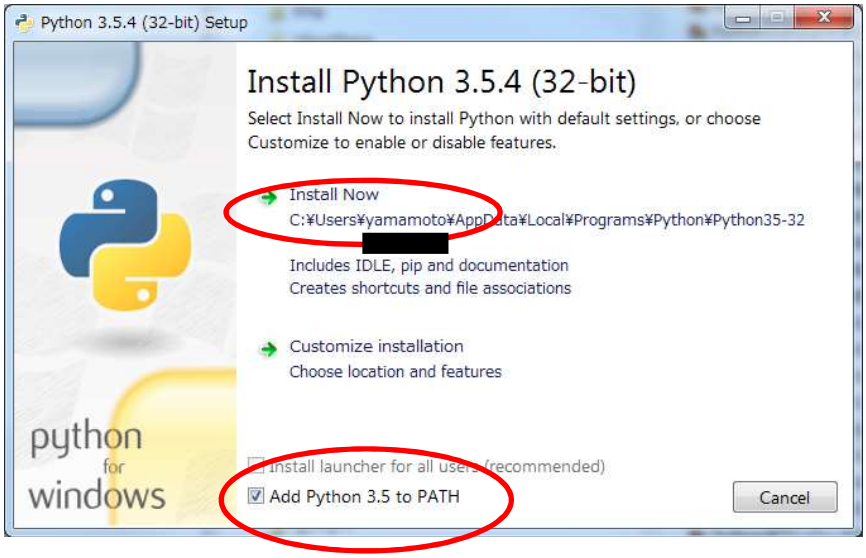


図 3-6 Python3.5.4 インストール(2/6)

インストール実行中になる。

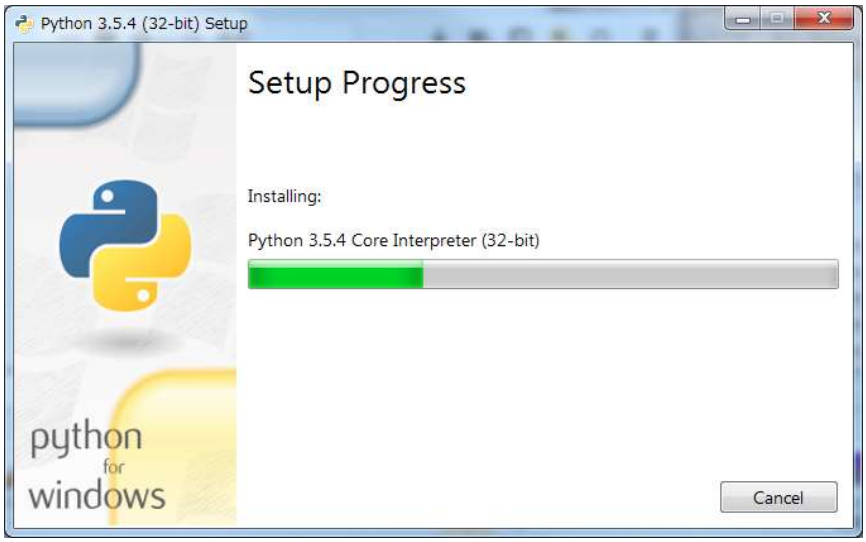


図 3-7 Python3.5.4 インストール(3/6)

インストール成功で、「Close」を選択してインストールを終了する。

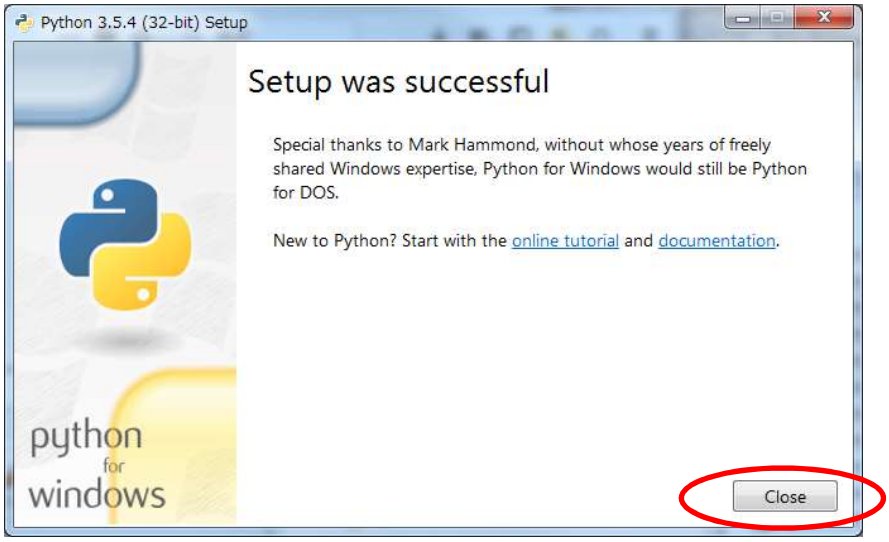


図 3-8 Python3.5.4 インストール(4/6)

(3)正しくインストールされたかを確認する。

　DOSプロンプトを開き、インストール確認する。

“python --version”を入力して、“Python 3.5.4”の表示を確認する。

“python”を入力して、“Python 3.5.4”が実行される事を確認する。

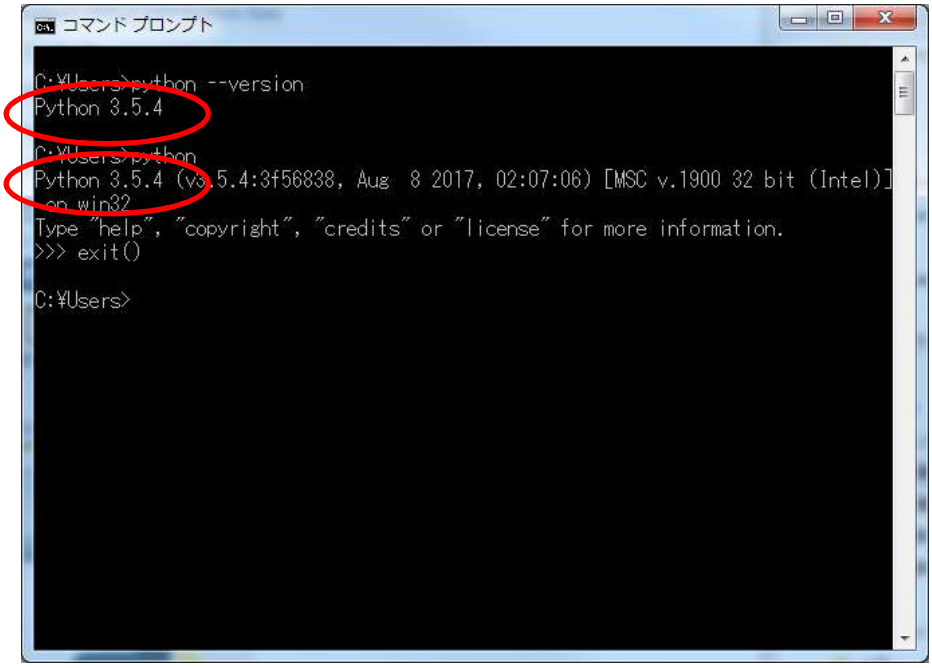


図 3-9 Python3.5.4 インストール(5/6)

　このインストールで Python 3.5.4 自身と付属のデバッガとして IDEL がインストールされます。

以下はスタートメニュー抜粋のインストール状態です。

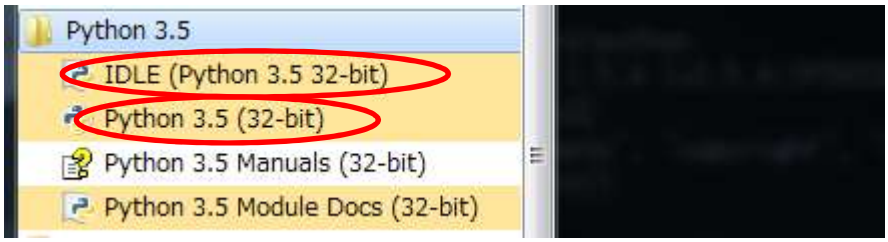


図 3-10 Python3.5.4 インストール(6/6)

　以上で、Python 3.5.4 のインストールが完了です。

3.1.2 Python 用 CANTOOL 制御ライブラリのインストール

Python で CANTOOL を制御するための制御ライブラリのインストール手順を説明します。

CANTOOL リリースイメージの Script¥Library 内の RmtDll.pyd を Python の実行するカレントフォルダまたは Python の実行環境にコピーします。

“C:¥Users¥(ユーザ名)¥AppData¥Local¥Programs¥Python¥Python35-32¥DLLs”フォルダに RmtDll.pyd をコピーすれば、カレントフォルダに関係なく使用できます。

3.2 Python スクリプト作成手順

ここでは、Windows 上でのスクリプトの作成とデバッグの手順を説明します。

3.2.1 Python スクリプト作成

Python スクリプトはテキストエディタもしくは、Python 関連のツールで作成します。
ここでは、Python 付属のデバッガである IDEL を使用した例を説明します。
IDLE は命令強調表示したり、インデントを自動で行ったりする機能があり、Python スクリプトの作成を助けてくれます。

※IDEL の操作詳細については、下記の「25.5. IDLE」を参照ください。
※URL：https://docs.python.jp/3/library/idle.html

(1)IDLEの起動

スタートメニューの「Python 3.5.4」のIDELを選択してIDELを起動します。

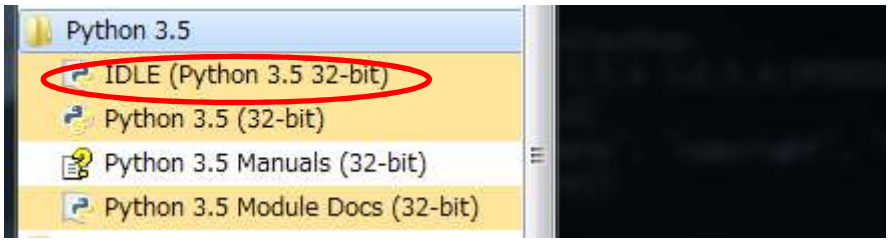


図 3-11 Python スクリプト作成手順(1/8)

以下が「Python 3.5.4 Shell」画面です。

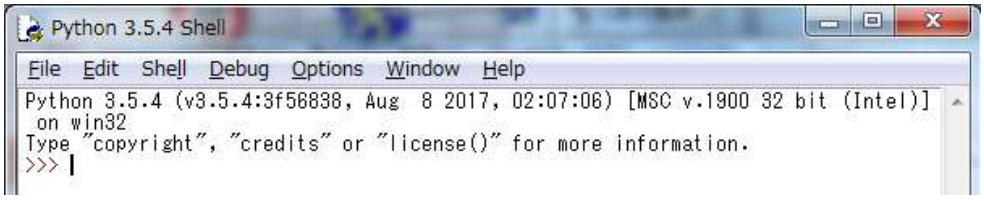


図 3-12 Python スクリプト作成手順(2/8)

(2)編集画面を開く

「File」メニューの「New File」を選択して、編集画面を表示します。

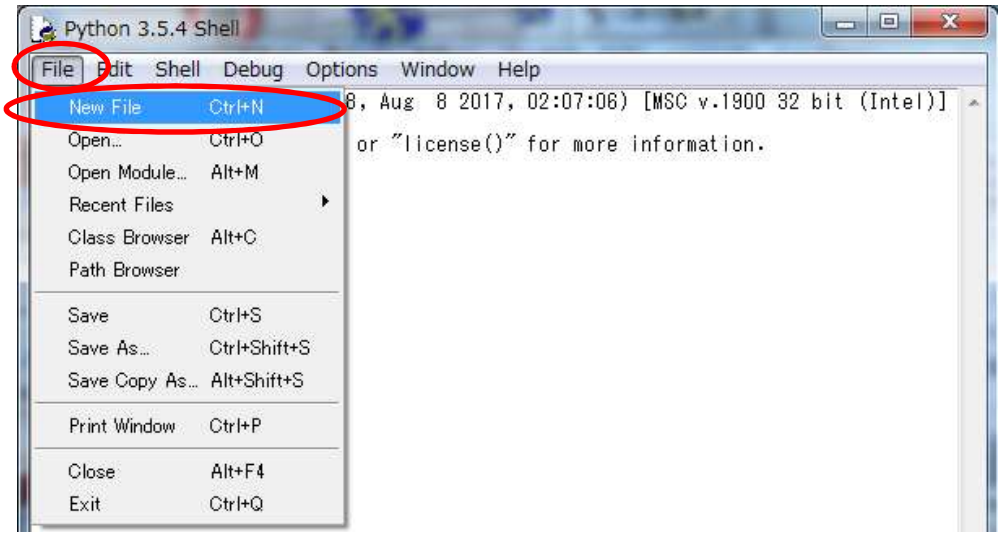


図 3-13 Python スクリプト作成手順(3/8)

以下が編集画面です。

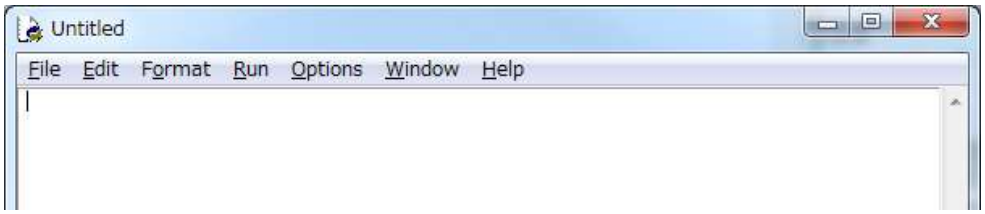


図 3-14 Python スクリプト作成手順(4/8)

(3)Pythonスクリプトコードの入力

CANTOOL 制御用 Python スクリプトの基本的な記載方法として、本体バージョンを取得する Python スクリプトの例を以下に示します。

※より実践的な利用方法については、CANTOOL リリースイメージ内の Script¥Sample 内を参照ください。

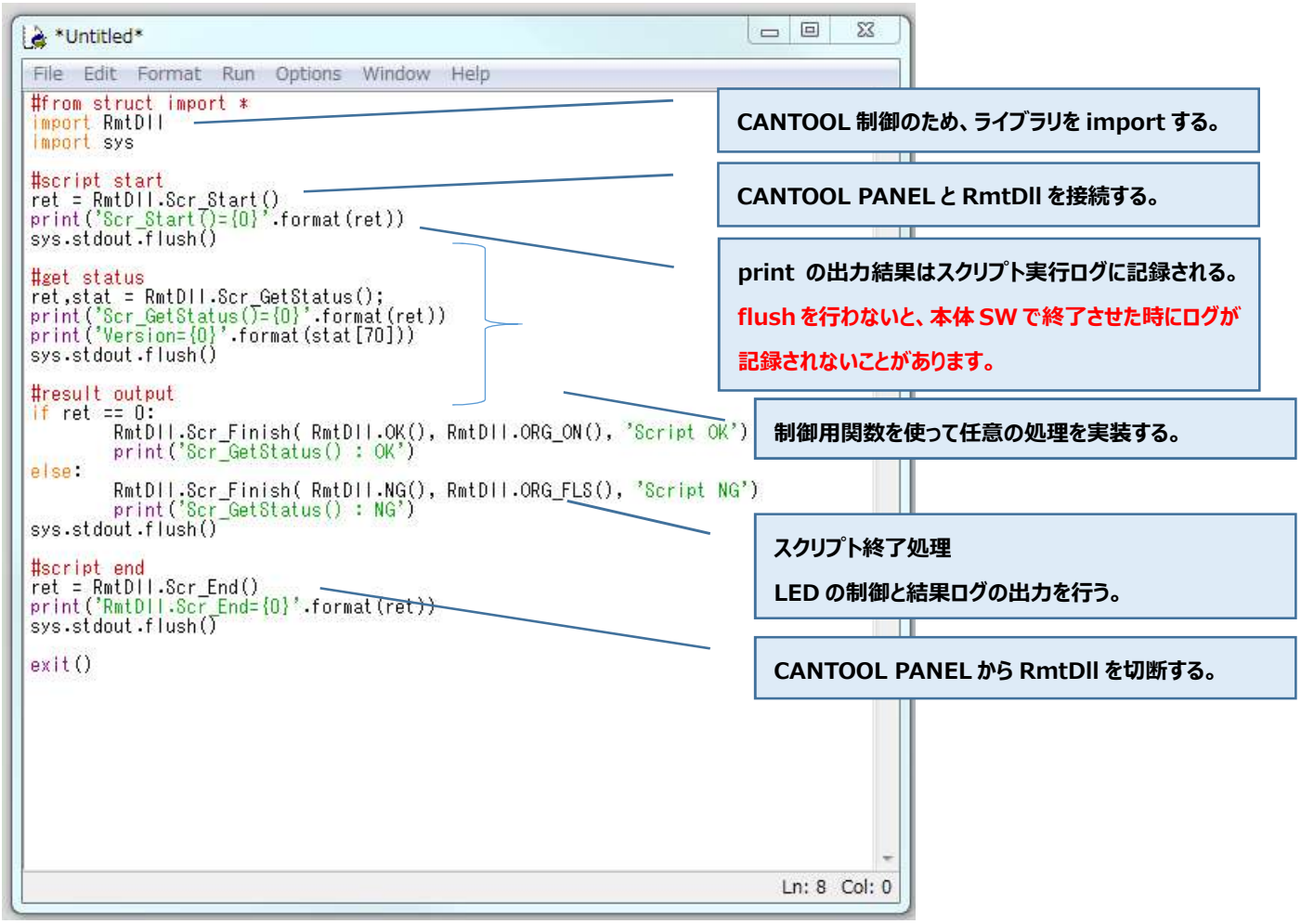


図 3-15 Python スクリプト作成手順(5/8)

(4) Pythonスクリプトファイルの保存

編集終了で、「File」メニューの「Save As...」を選択しファイルに保存します。

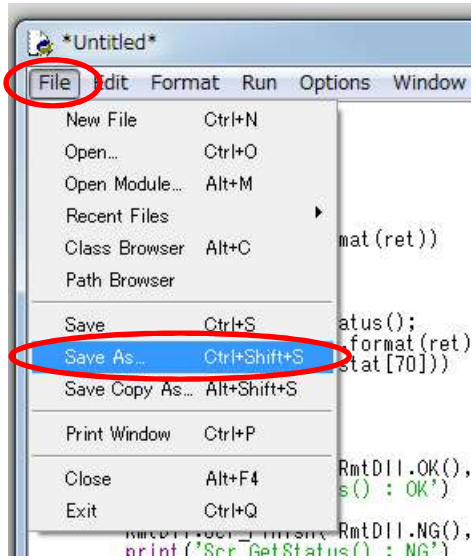


図 3-16 Python スクリプト作成手順(6/8)

ファイル名を指定して、保存します。

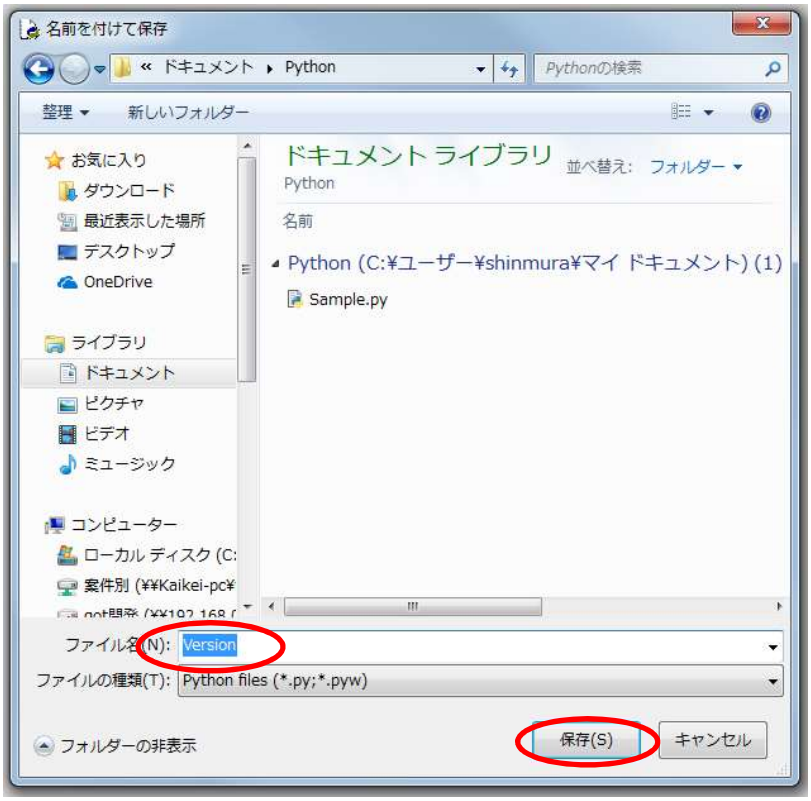


図 3-17 Python スクリプト作成手順(7/8)

ファイル保存でファイル名が表示されます。

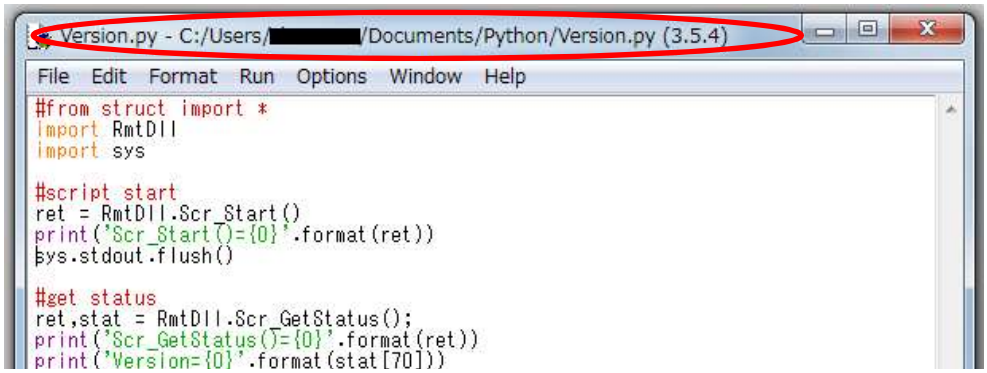


図 3-18 Python スクリプト作成手順(8/8)

(5)終了時は、各画面を閉じてください。

3.2.2　Windows 上での Python スクリプトデバッグ

ここでは Windows 上での Python スクリプトのデバッグ手順を説明します。
Python インストールでインストールされている IDEL デバッガを使用して説明します。

(1)CANTOOL PANEL起動

Windows上でCANTOOLの制御ライブラリを使用する場合は、CANTOOL PANELをあらかじめ起動しておく必要があります。
CANTOOL PANELを起動せずに実行すると。後述のScr_Start関数でエラーとなります。

(2)IDLE起動

スタートメニューの「Python 3.5.4」のIDELを選択してIDELを起動します。

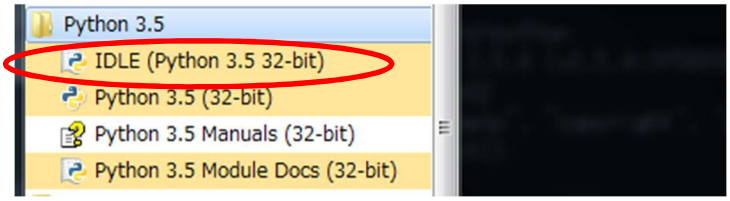


図 3-19 Python スクリプトデバッグ手順(1/21)

以下が「Python 3.5.4 Shell」画面です。

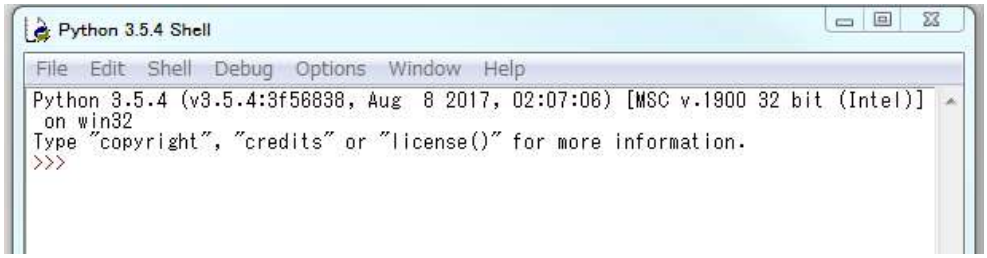


図 3-20 Python スクリプトデバッグ手順(2/21)

(3)デバッグ対象ファイルを開く

デバッグするPythonスクリプトファイルを「File」メニューの「Open..」を選択して開きます。

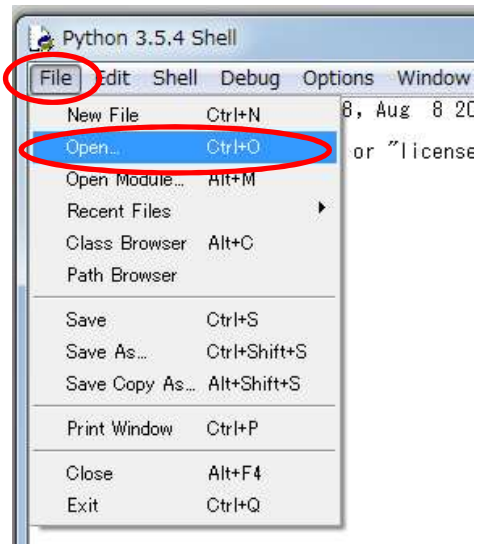


図 3-21 スクリプトデバッグ手順(3/21)

ファイ名を選択して、Python スクリプトファイルを開きます。

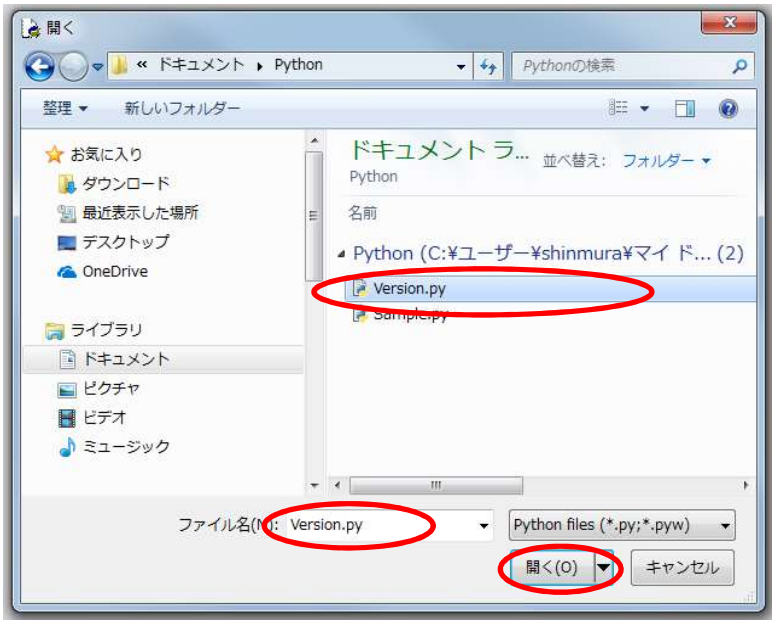


図 3-22 Python スクリプトデバッグ手順(4/21)

編集画面に Python スクリプトファイル内容が読み込まれ、表示されます。

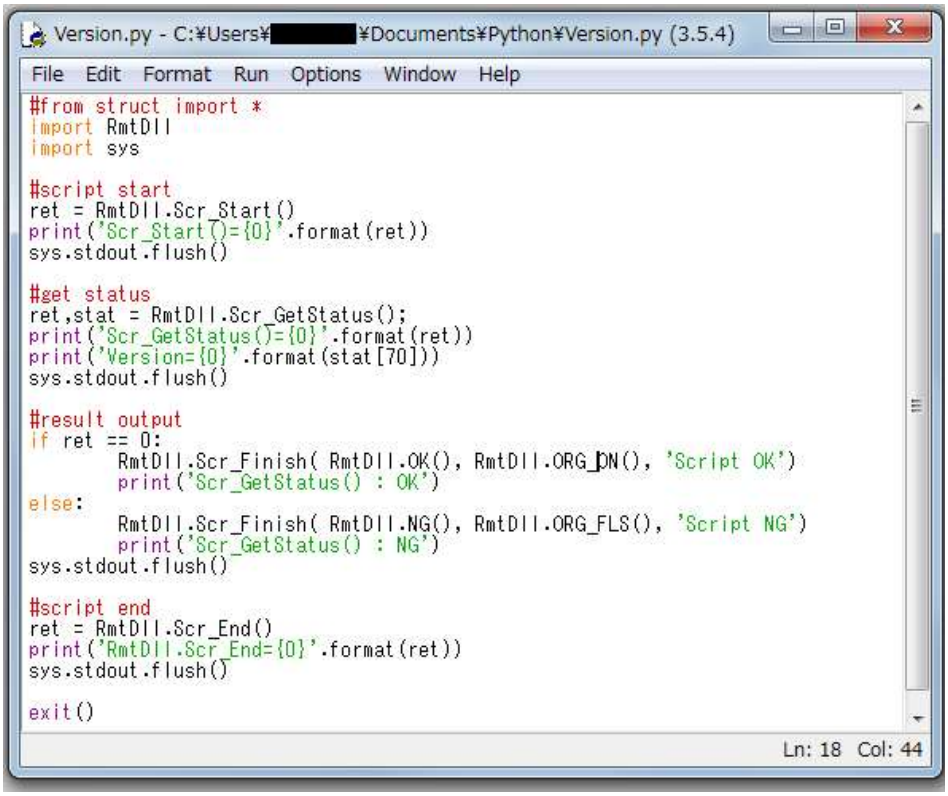


図 3-23 Python スクリプトデバッグ手順(5/21)

(4)デバッグ画面を開く

ステップ実行確認をする場合は、「Debug」メニューの「Debugger」を選択してデバッグ設定を行います。

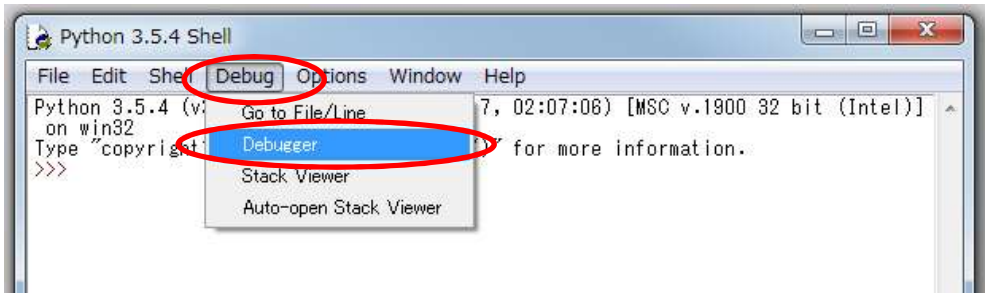


図 3-24 Python スクリプトデバッグ手順(6/21)

以下のように、４つ全てにチェックしてデバッグ設定を行います。

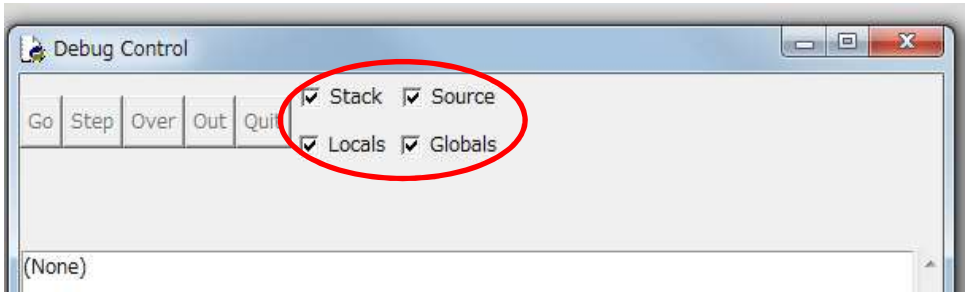


図 3-25 Python スクリプトデバッグ手順(7/21)

上記チェックで Shell 画面には「DEBUG ON」が表示されます。

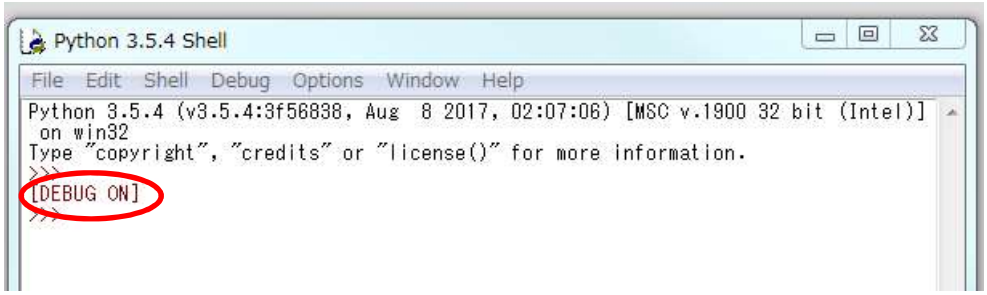


図 3-26 Python スクリプトデバッグ手順(8/21)

(5)デバッグの開始

デバッグの開始を、編集画面の「Run」メニューの「Run Module」を選択して行います。

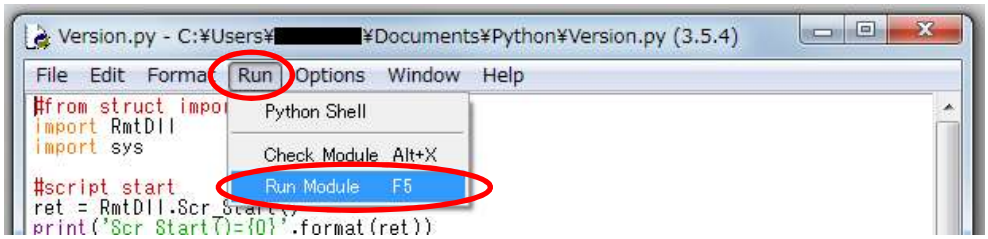


図 3-27 Python スクリプトデバッグ手順(9/21)

デバッグ開始で、プログラムの最初の行で停止します。

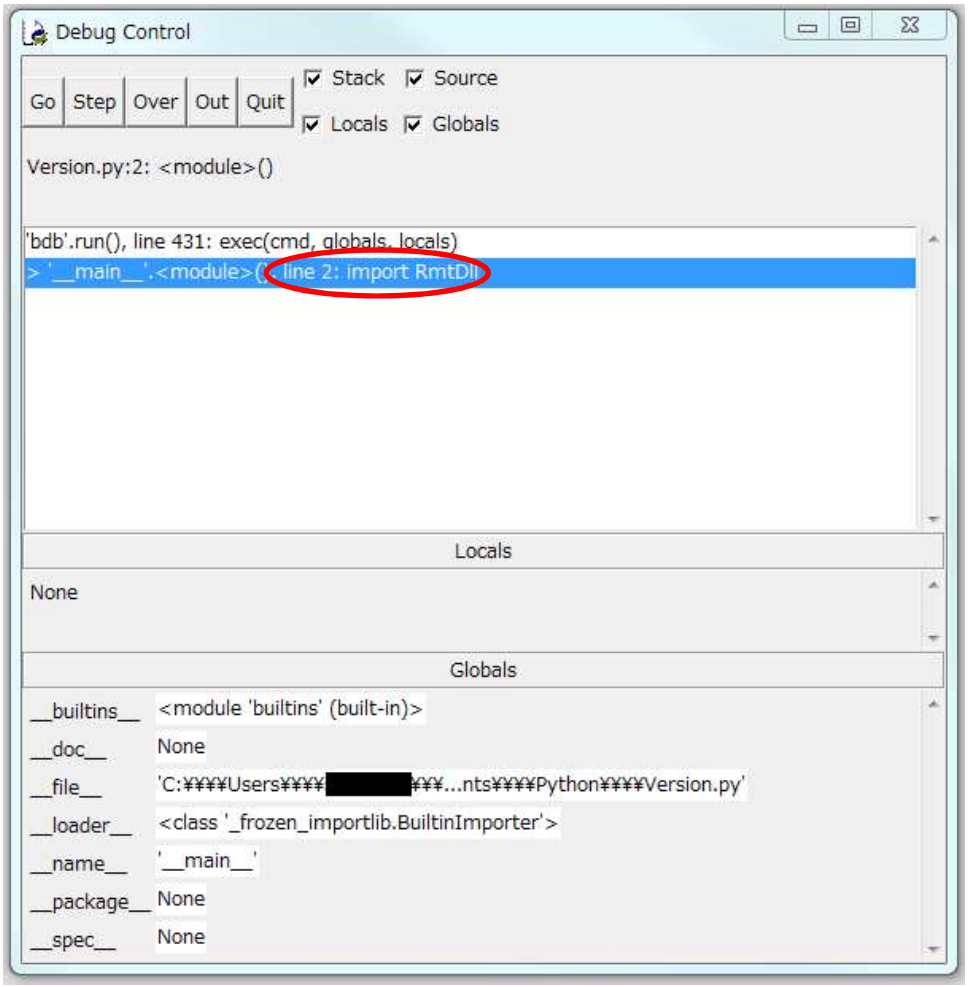


図 3-28 Python スクリプトデバッグ手順(10/21)

Shell 画面には Python スクリプト開始メッセージが表示されます。

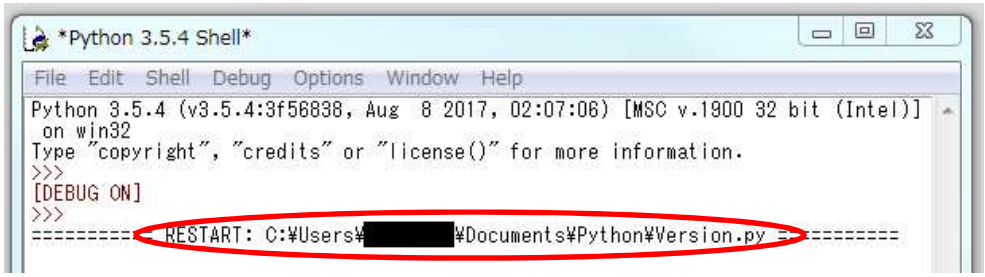


図 3-29 Python スクリプトデバッグ手順(11/21)

(6)デバッグ実行

デバッグ画面操作で、デバッグを開始します。

「Over」でプログラムを進め、6 行目に停止した状態です。

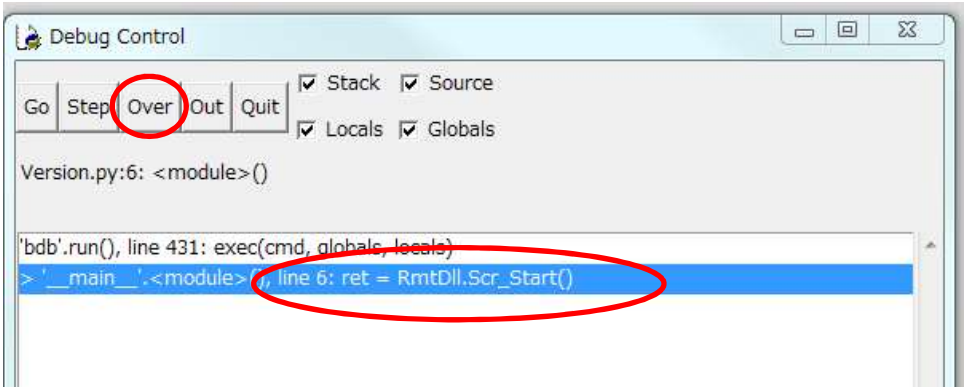


図 3-30 Python スクリプトデバッグ手順(12/21)

編集画面の 6 行目で停止状態表示。

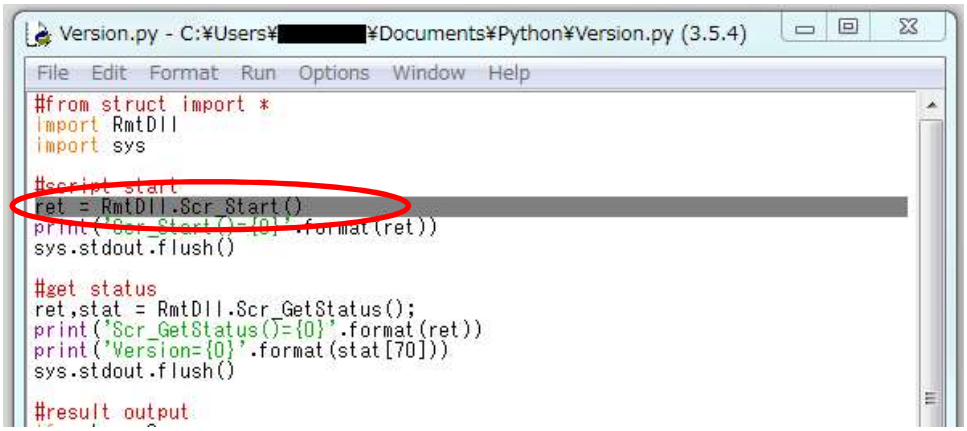


図 3-31 Python スクリプトデバッグ手順(13/21)

デバッグ操作は以下の操作で行います。

「Go」ボタン

プログラムをブレークポイントの直前まで実行。ブレークポイントがない場合は、最後まで実行される。

「Step」ボタン

現在の行を実行して、次の行で停止します。現在の行に関数呼び出しがあるとデバッガーはその関数の中に入ります。

「Over」ボタン

現在の行を実行して、次の行で停止します。現在の行に関数呼び出しがあるとその関数を実行してから次の行で停止します。

「Out」ボタン

現在の関数を最後まで実行して、関数の外に移ります。Step で関数の中に入った場合、Out で関数の外に出ることができます。

「Quit」ボタン

デバッグを終了します。残りのコードは実行されません。もう一度、デバッグを開始するには、最初と同じように、 Debug > Debugger を選択します。

Step と Over の違いは、現在の行に関数呼び出しがある場合に、関数の中に入るかどうかです。通常は、関数の中に入らない Over の方をよく用います。

「Over」でプログラムを進め6行目の関数実行後、7行目に停止した状態です。
「Globals」欄で、変数等の使用した状態が確認できます。下図では、6 行目の関数の戻り値が 0 であることがわかります。

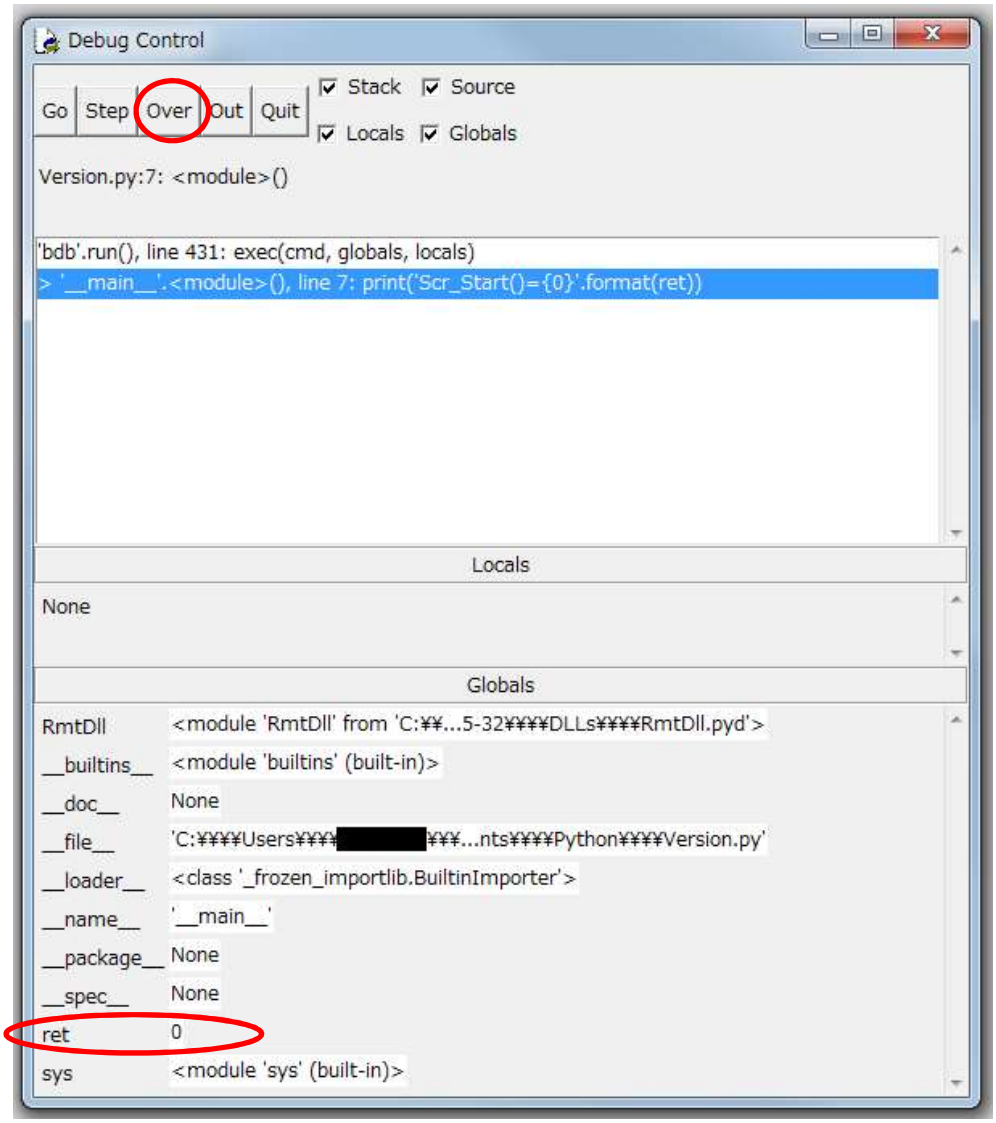


図 3-32 Python スクリプトデバッグ手順(14/21)

現在の停止位置です。

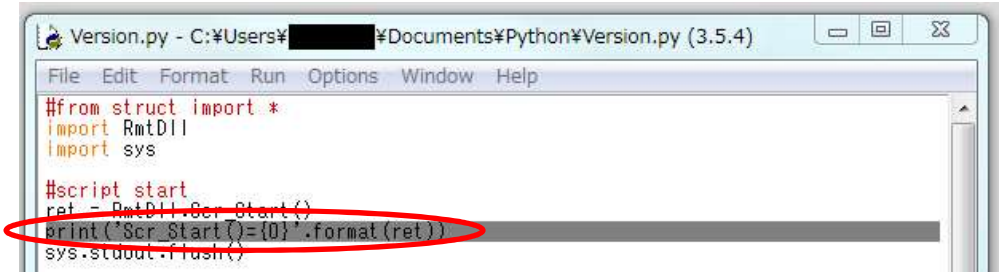


図 3-33 Python スクリプトデバッグ手順(15/21)

「Step」操作により、“print('Scr_Start()={0}'.format(r e t))”行で標準ライブラリ内にステップインして、別の編集画面で表示された状態です。
抜ける時は、「Out」操作で元の Python スクリプト編集画面に戻ります。

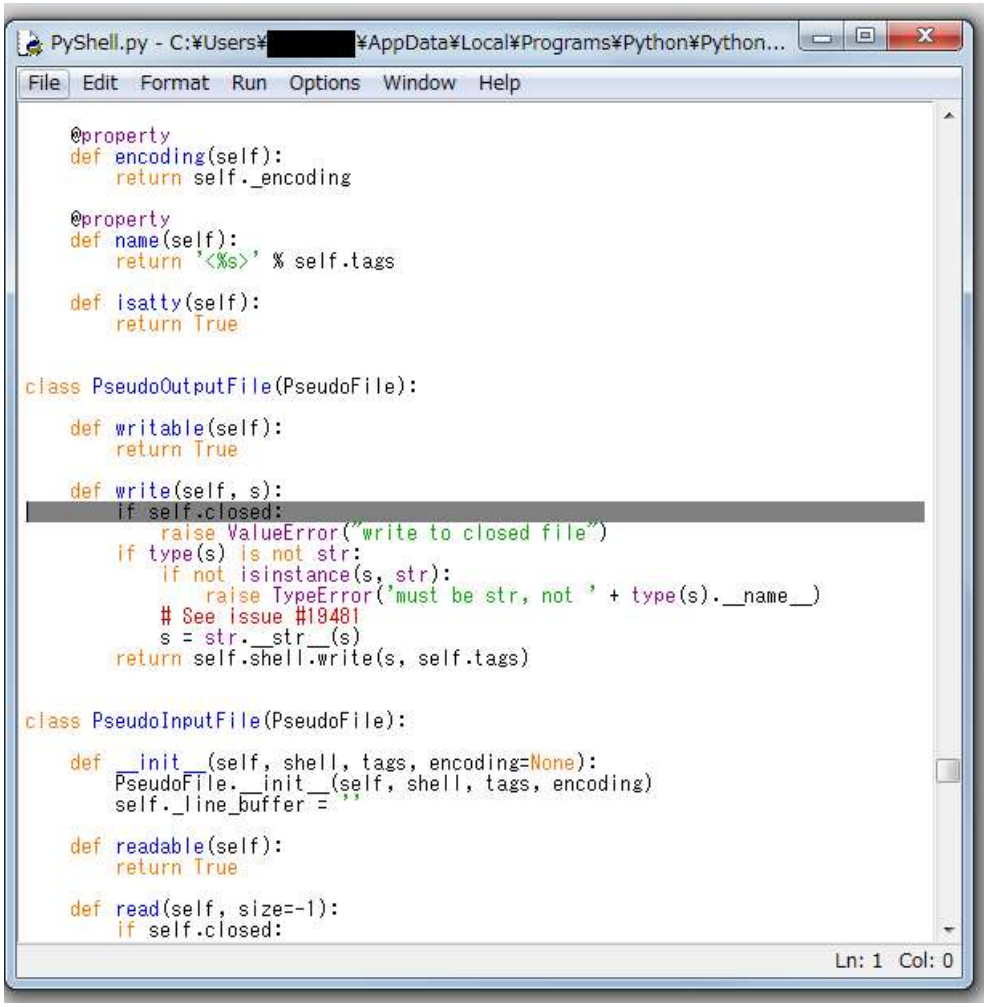


図 3-34 Python スクリプトデバッグ手順(16/21)

7行目の“print('Scr_Start()={0}'.format(r e t))”行によるログ表示内容です。

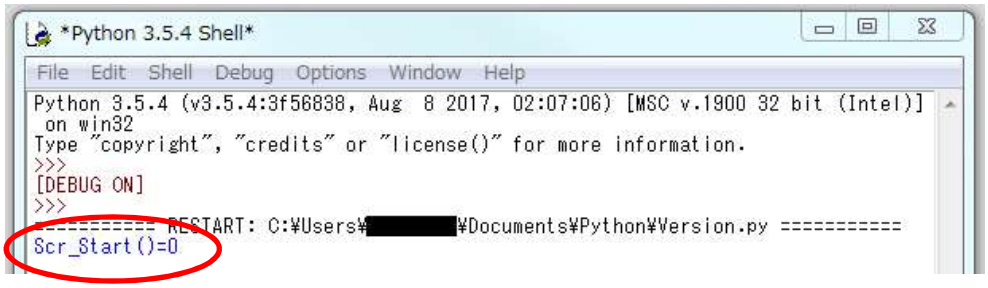


図 3-35 Python スクリプトデバッグ手順(17/21)

上記以外に、ブレーク行を指定して実行を停止する事もできます。
ブレークしたい行で、右クリックによりポップアップ表示で指定できる。
ブレーク行設定で、選択行のバックが黄色表示になります。

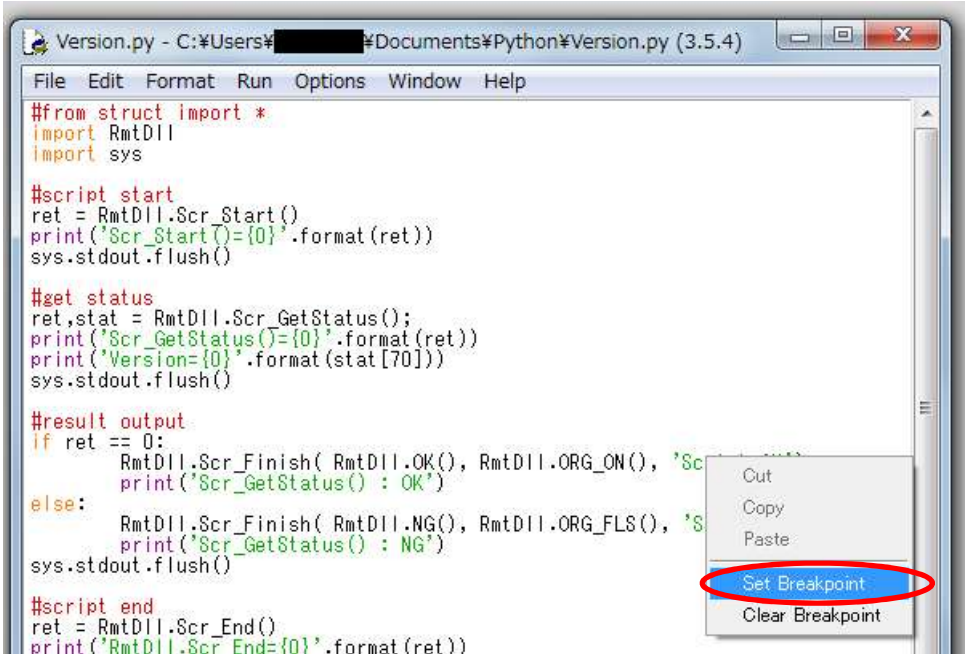


図 3-36 Python スクリプトデバッグ手順(18/21)

「Go」選択で最後まで実行する。
最後まで実行すると、下記の確認画面が表示されます。
「OK」を選択すると、実行ログを表示した Shell 画面とデバッグ操作画面が消えます。
「キャンセル」を選択すると、実行ログを表示した Shell 画面とデバッグ操作画面は残ります。

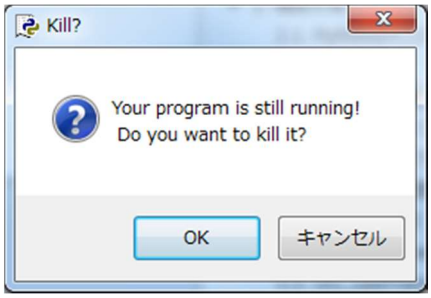


図 3-37 Python スクリプトデバッグ手順(19/21)

実行終了で、「キャンセル」を選択した時の Shell 画面です。

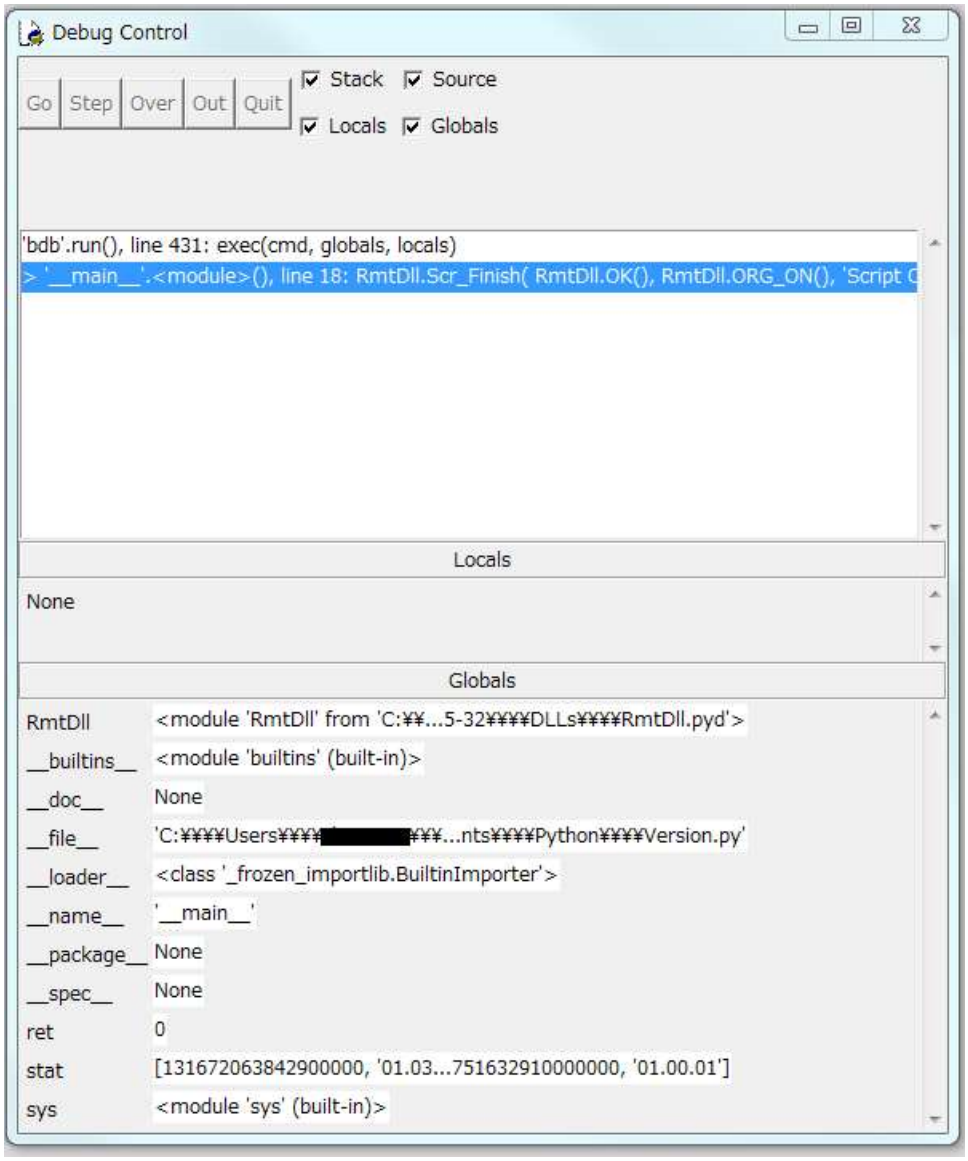


図 3-38 Python スクリプトデバッグ手順(20/21)

実行終了時の操作画面です。Python バージョンがログに出力されています。
Scr_Finish 関数の結果はログ出力されません。

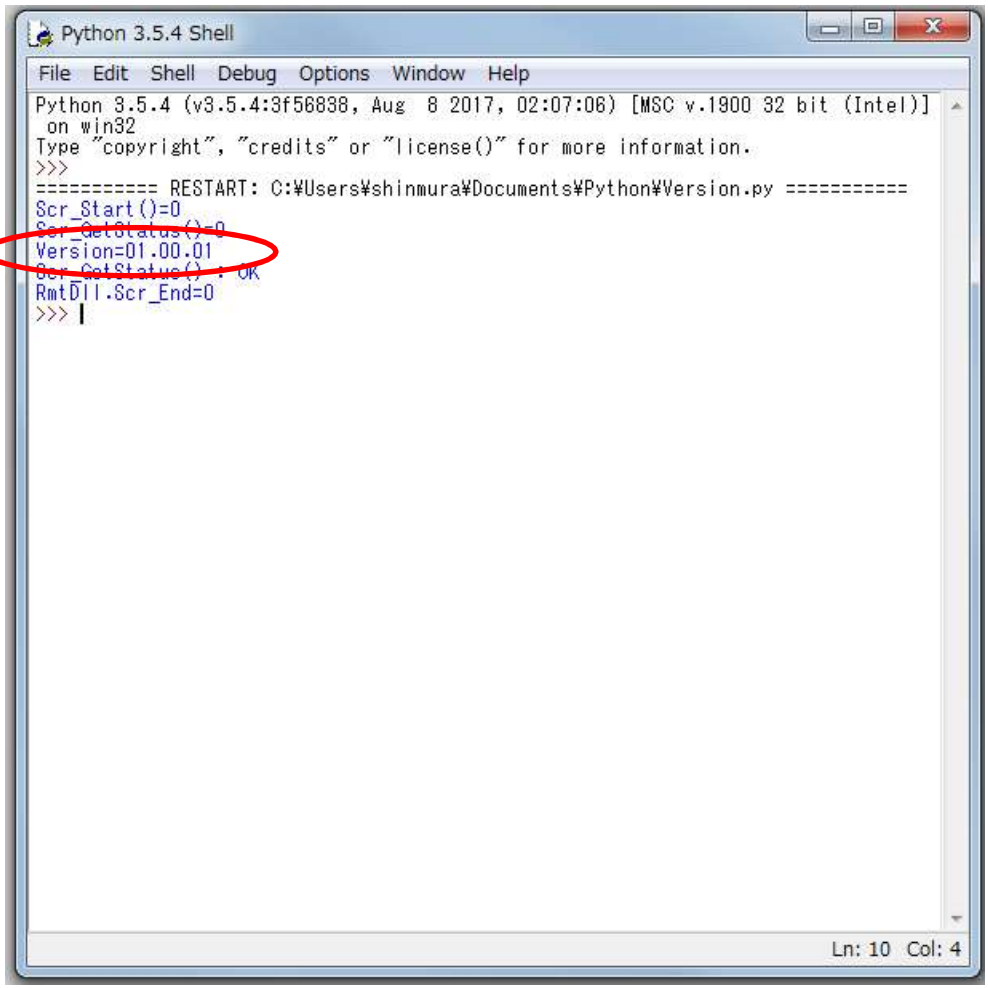


図 3-39 Python スクリプトデバッグ手順(21/21)

3.3 Python スクリプト実行手順

　CANTOOL 本体での Python スクリプト実行手順を説明します。
なお、Python スクリプトの実行には SD カードが必要となります。

3.3.1 事前準備

(1)CANTOOLにPythonスクリプト実行環境を構築する

　CANTOOL PANELを起動します。

「Setting」を選択します。

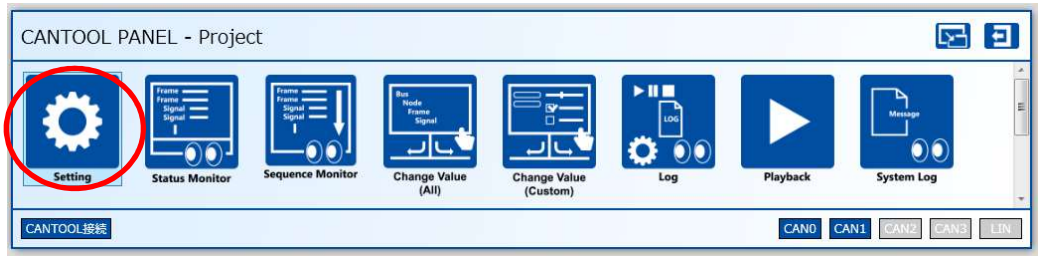


図 3-40 Python スクリプト実行手順(1/9)

　「プロジェクト設定」で「スタンドアロン設定」を選択します。

　CANTOOL に SD カードを挿入し、「スクリプト実行環境」「構築」ボタンを選択します。

図 3-41 Python スクリプト実行手順(2/9)

Python実行環境のインストールファイルを指定します。

CANTOOL の¥Support¥ScriptImage¥内にある python_env_verXX.XX.XX.zip を選択し、「開く」ボタンを選択します。

※XX はバージョンを表す数字

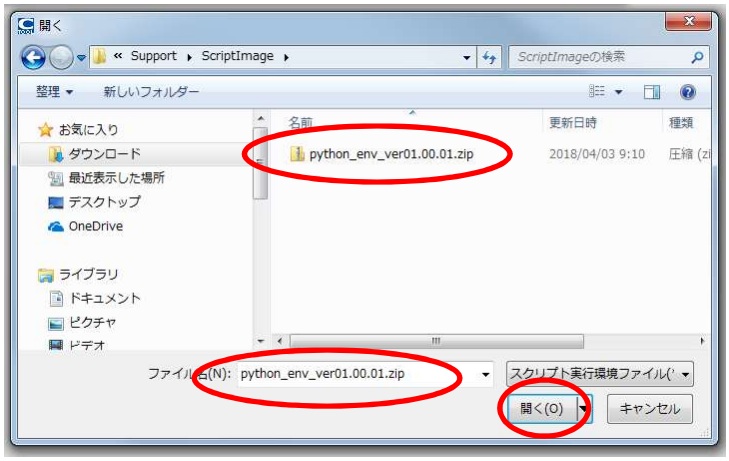


図 3-42 Python スクリプト実行手順(3/9)

構築が完了すると以下のメッセージが表示されます。

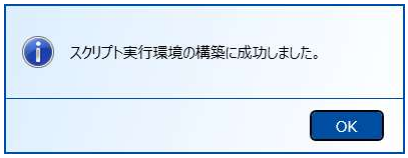


図 3-43 Python スクリプト実行手順(4/9)

3.3.2 Python スクリプトファイルを本体へ設定する

(1) CANTOOL PANELでPythonスクリプトファイルを設定する。

「プロジェクト設定」で「スタンドアロン設定」を選択します。

各ユーザスイッチに Python スクリプトファイルを設定します。

「実行データ操作」で「スクリプト」を選択し、実行する Python スクリプトファイルを指定します。

画面例ではユーザスイッチ 1 に 4 章で作成したスクリプトファイルを指定しています。

図 3-44 Python スクリプト実行手順(5/9)

(2)設定したPythonスクリプトファイルを本体に設定します。

「スタンドアロン設定を反映」「反映」を選択します。

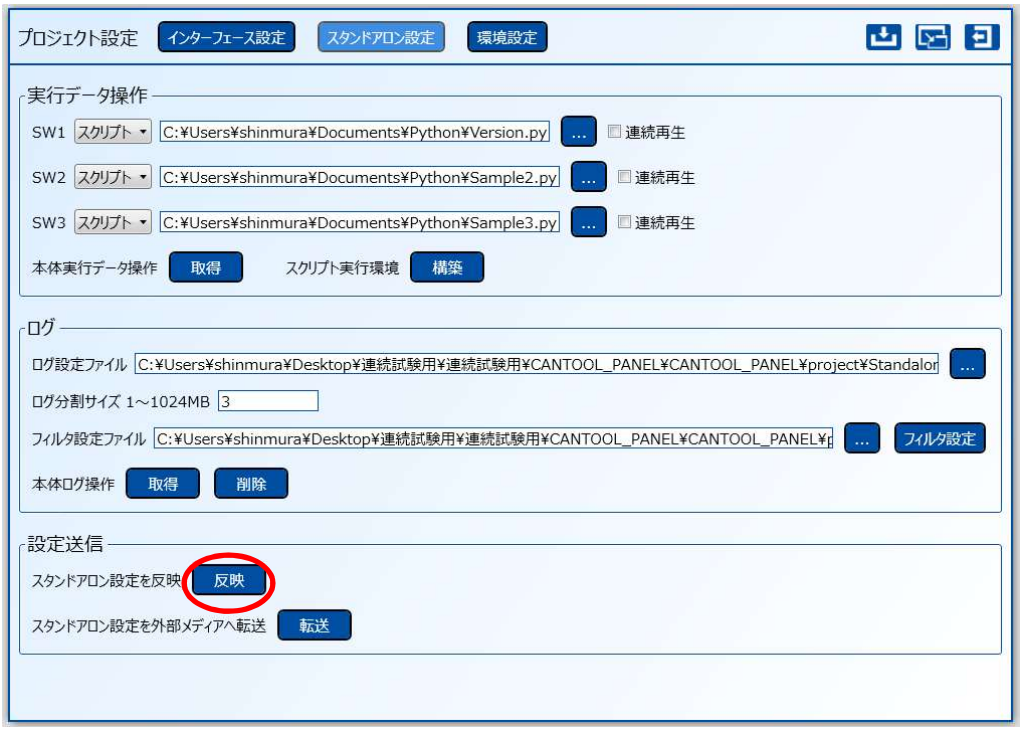


図 3-45 Python スクリプト実行手順(6/9)

反映が完了すると以下のメッセージが表示されます。

CANTOOL の SW に対応する LED が緑点灯し、Python スクリプトが実行可能状態であることを示します。

図 3-46 Python スクリプト実行手順(7/9)

3.3.3 本体での実行

実行したい Python スクリプトを登録した CANTOOL の SW を押下します。

実行開始されると対応する LED が緑点滅します。

緑点滅中に SW を押下した場合は、スクリプトを強制終了します。

Python スクリプトの実行が終了すると、Scr_Finish()関数で指定した LED 状態になります。

システムやスクリプトの問題で Python スクリプトが停止した場合は LED が赤点灯状態になります。

3.3.4 実行結果の確認

(1) Pythonスクリプトの実行ログ取得

「スタンドアロン設定」の「本体ログ操作」＞「取得」を選択する。

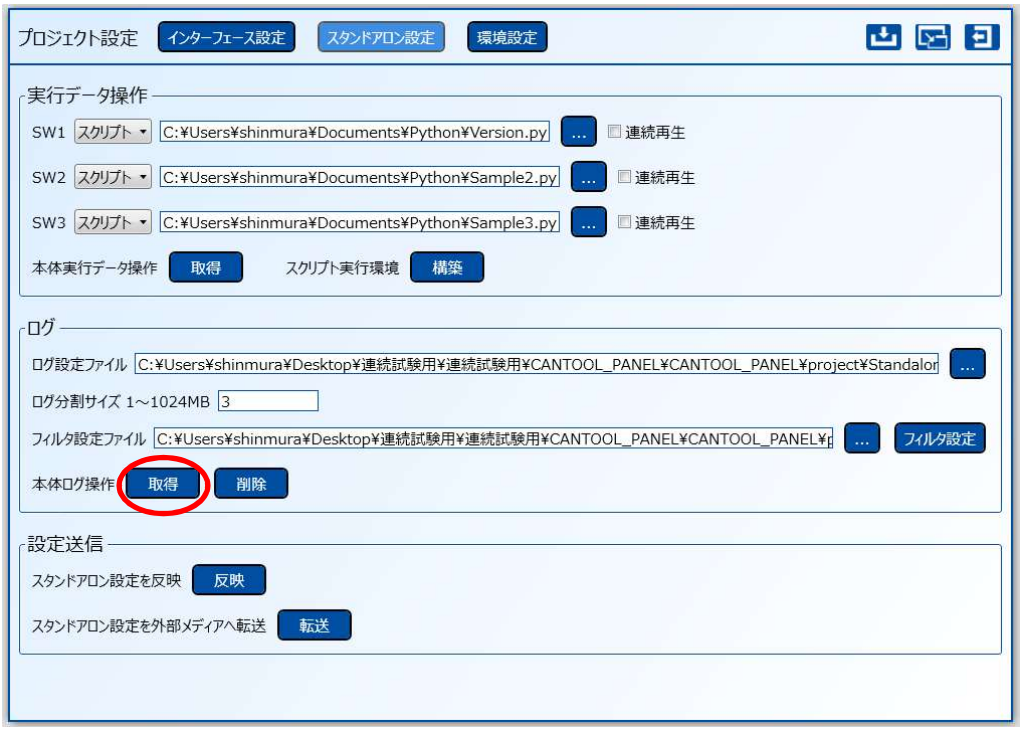


図 3-47 Python スクリプト実行手順(8/9)

ログの保存先を選択(任意)し、「OK」ボタンを選択します。

保存先フォルダ内に"Log_YYYYMMDDhhmmss¥ScriptLog¥SW１¥0001～N¥"(※１)内に拡張子"ｔｘｔ"と"result"のログファイルが出来ます。

※１　YYYYMMDDhhmmss はログ取得した年月日時分秒、N は 0001 から実行の度連番となる。

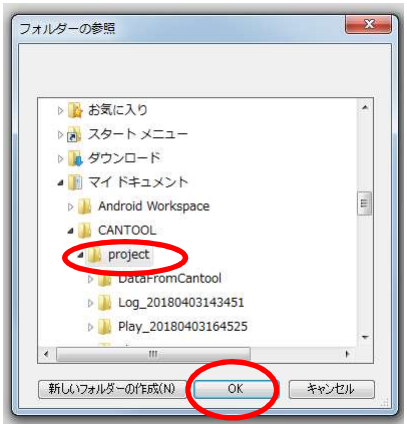


図 3-48 Python スクリプト実行手順(9/9)

拡張子" t x t "のファイルには、Python スクリプトで print 出力した内容がログ出力されます。

```
Scr_Start 01.03.00　2018/04/02 : script_RmtDll_Version
Scr_Start()=0
Scr_GetStatus()=0
Version=01.00.01
Scr_GetStatus() : OK
RmtDll.Scr_End=0
```

拡張子"result"のファイルには、後述の Scr_Finish 関数で指定した出力がログ出力されます。

```
OK,ORG_ON,Script OK
```

3.4 注意事項

特になし

4. ライブラリインターフェース仕様

CANTOOL Script が提供する API の一覧を以下に示します。各 API の詳細は
「V016002-703_CANTOOL_A1_Script-ReferenceManual-API」を参照してください。

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	スクリプト使用開始通知処理	Scr_Start
エンド	スクリプト使用終了処理	Scr_End
ステータス	各種状態を取得する。	Scr_GetStatus
	指定の LED を制御する。	Scr_SetLED
	スクリプト終了時に、スクリプトを登録した SW に対応した LED を制御する。 また、実行ログとは別に結果ログに文字列を出力する。	Scr_Finish
	結果 OK を判定する文字列を返す。Scr_Finish の引数①用。	OK
	結果 NG を判定する文字列を返す。Scr_Finish の引数①用。	NG
	LED を OFF に制御する文字列を返す。Scr_Finish の引数②用。	LGT_OFF
	LED を緑点灯に制御する文字列を返す。Scr_Finish の引数②用。	GRN_ON
	LED を緑点滅に制御する文字列を返す。Scr_Finish の引数②用。	GRN_FLS
	LED を橙点灯に制御する文字列を返す。Scr_Finish の引数②用。	ORG_ON
	LED を橙点滅に制御する文字列を返す。Scr_Finish の引数②用。	ORG_FLS
	LED を赤点滅に制御する文字列を返す。Scr_Finish の引数②用。	RED_FLS
	Frame に割り当てられている Signal のリストを取得する。	Scr_ConvertSignalListFromFrame
	BusID と XID から FrameID を取得する。	Scr_ConvertXID2FrameID
	BusID と XID と SignalSymbol から SignalID を取得する。	Scr_ConvertXID2SignalID
	本体の UART の設定の変更を行う。	Scr_SetUartSetting
	本体の UART の設定を取得する。	Scr_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。	Scr_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。	Scr_GetSignalSize
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。	Scr_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。	Scr_SendSignalData
	指定ポートの DA の値を変更する。	Scr_SendDaData
	指定ポートの GPO の値を変更する。	Scr_SendGpoData
	任意の CAN Frame データを送信する。	Scr_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。	Scr_SendCyclicData2
	MultiPacket データを送信する。	Scr_SendMultiPacketData2
	MultiPacket への応答内容設定。	Scr_ResponseMultiPacketData

カテゴリ	説明	名称
	UART へのデータを送信す。	Scr_SendUartData
データ受信	指定フレームの最新データを取得する。	Scr_GetFrameData
	指定シグナルの最新データを取得する。	Scr_GetSignalData
	指定ポートの AD の値を取得する。	Scr_GetAdData
	指定ポートの GPI の値を取得する。	Scr_GetGpiData
	MultiPacket 状態を取得する。	Scr_GetMultiPacketStatus
	MultiPacket の応答結果を取得する。	Scr_GetResponseMultiPacketDataStatus

アイテック阪急阪神 株式会社

アイテック阪急阪神 株式会社

Rev.　01.02
最終更新日：　2022/08/31

(※)各エラーや定義、構造体については、同梱の「ctdef.py」を参照してください。

種別	定義	概要	引数	設定可能範囲	取得内容(戻り値)	説明
Connection	Scr_Start	ユーザーアプリをCANTOOL本体アプリと接続する。	無し	-	①整数型：CT_result ctdef.pyのCT_resultを参照	・本APIによりCANTOOL本体アプリに接続することで、スクリプトからAPIでCANTOOL本体が制御可能となる。複数のスクリプト(最大4個)を接続することが可能。 ・すでにCANTOOL本体に読み込んでいるプロジェクトファイルを使用する。
	Scr_End	ユーザーアプリをCANTOOL-PANELから切断する。	無し	-	①整数型：CT_result ctdef.pyのCT_resultを参照	・本APIによりCANTOOL本体アプリからスクリプトを切断し、接続していたスクリプトの情報を破棄する。 ・Scr_End()を呼び出さずにスクリプトが終了した場合、その後のスクリプト動作は保証しない。CANTOOL本体を再起動すること。ただし、本体のSWを押して終了した場合は再起動の必要はない。
Status	Scr_GetStatus	各種状態を取得する	無し	-	①整数型：CT_result ctdef.pyのCT_resultを参照 ②リスト型：status 取得できる情報はctdef.pyのCT_Configを参照	・statusのリストの項目はCT_Configを使って取得する。
	Scr_SetLED	指定のLEDを制御する	①整数型 LED port [I] LED番号 ②整数型 LED color [I] 色 ③整数型 LED Status [I] 動作(点灯、点滅、消灯)	① SCR_LED_PORT_USER1 SCR_LED_PORT_USER2 ② SCR_LED_COLOR_OFF SCR_LED_COLOR_RED SCR_LED_COLOR_GREEN SCR_LED_COLOR_ORANGE ③ SCR_LED_STATUS_OFF SCR_LED_STATUS_ON SCR_LED_STATUS_BRINK	①整数型：CT_result ctdef.pyのCT_result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。
	Scr_Finish	スクリプト終了時に、スクリプトを登録したSWに対応したLEDを制御する。 また、実行ログとは別に結果ログに文字列を出力する。	①文字列型：OK/NG判定文字列 ②文字列型：LED制御文字列 ③文字列型：ユーザーメッセージ用	① OK() NG() ② LGT_OFF() GRN_ON() GRN_FLS() ORG_ON() ORG_FLS() RED_FLS() ③任意の文字列。最大1000文字。	①整数型：CT_result ctdef.pyのCT_resultを参照	・本APIをコールすると実行ログとは別に「script_log_XXXX.result」と言うファイルが結果ログとして出力される。 ①と③を利用してスクリプトの実行結果を判断に利用できる。 ・コールせずスクリプトを終了させたり、スクリプトやシステムの問題で途中で実行が停止した場合は結果ログは出力されず、対応するLEDは赤点灯となる。 ・スクリプト終了後、スクリプトを登録したSWに対応するLEDが、②で指定した点灯/消灯状態となる。ログを確認せずとも、本体LEDで実行結果判断に利用できる。 ・③に1000文字以上の文字列を設定した場合の動作は保証しない。 ※Windows上での実行は非サポート。
	OK	結果OKを判定する文字列を返す。Scr_Finishの引数①用。	無し	-	①文字列型 結果OKを判定する文字列	・「OK」の文字列を返す。
	NG	結果NGを判定する文字列を返す。Scr_Finishの引数①用。	無し	-	①文字列型 結果NGを判定する文字列	・「NG」の文字列を返す。
	LGT_OFF	LEDをOFFに制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDをOFFに制御する文字列	・「LGT_OFF」の文字列を返す。
	GRN_ON	LEDを緑点灯に制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDを緑点灯に制御する文字列	・「GRN_ON」の文字列を返す。
	GRN_FLS	LEDを緑点滅に制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDを緑点滅に制御する文字列	・「GRN_FLS」の文字列を返す。
	ORG_ON	LEDを橙点灯に制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDを橙点灯に制御する文字列	・「ORG_ON」の文字列を返す。
	ORG_FLS	LEDを橙点滅に制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDを橙点滅に制御する文字列	・「ORG_FLS」の文字列を返す。
	RED_FLS	LEDを赤点滅に制御する文字列を返す。Scr_Finishの引数②用。	無し	-	①文字列型 LEDを赤点滅に制御する文字列	・「RED_FLS」の文字列を返す。
Control	Scr_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Scr_ConvertXID2FrameID()で取得した値を使うこと。 ②0～255 ③②の数分セットすること 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。	①整数型：CT_result ctdef.pyのCT_result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。
	Scr_ConvertXID2FrameID	BusIdとXIDからFrameIDを取得する	①整数型 BusId [I] バスID ②整数型 SidOnly [I] 拡張ID使用有無 ③整数型 Sid [I] 標準フォーマットのベースID 11ビット ④整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤リスト型 FrameId [O] フレームID(フレームを一意に識別するID番号) リストの先頭に整数型の値が格納される。	① SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_BUS_ID_CAN3 SCR_BUS_ID_LIN ② 1：ベースIDのみ 0：ベースID＋拡張ID ③ CAN：0x00～0x7FF LIN：0x00～0x3F ④CAN:0x00～0x3FFFF ⑤0～4999	①整数型：CT_result ctdef.pyのCT_result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、SendFrameData、GetFrameDataなどの引数に使用する。
	Scr_ConvertXID2SignalID	BusIdとXIDとSignalSymbolからSignalIDを取得する	①整数型 BusId [I] バスID ②整数型 SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID+拡張ID) ③整数型 Sid [I] 標準フォーマットのベースID 11ビット ④整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤文字列 SignalSymbol [I] シグナル名 ⑥リスト型 SignalId [O] シグナルID(シグナルを一意に識別するID番号) リストの先頭に整数型の値が格納される。	① SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_BUS_ID_CAN3 SCR_BUS_ID_LIN ② 1：ベースIDのみ 0：ベースID＋拡張ID ③ CAN：0x00～0x7FF LIN：0x00～0x3F ④CAN:0x00～0x3FFFF ⑤1～63文字 ⑥0～29999	①整数型：CT_result ctdef.pyのCT_result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。

	Scr_SetUartSetting	本体のUARTの設定の変更を行う。	①整数型 enable[I] UART有効/無効 ②整数型 baudrate [I] ボーレート ③整数型 datalength [I] データ長 ④整数型 parity [I] パリティ ⑤整数型 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	①整数型：CT_result ctdef.pyのCT_resultを参照	・指定された値をUARTの設定に反映する。
	Scr_GetUartSetting	本体のUARTの設定を取得する。	無し	-	①整数型：CT_result ctdef.pyのCT_resultを参照 ②整数型：enable 有効/無効 ③整数型：baudrate ボーレート ④整数型：datalength データ長 ⑤整数型：parity パリティ ⑥整数型：stopbit ストップビット ⑦整数型：FreeMsgCnt 空きメッセージ数	・現在のUARTの設定内容を返信する。
	Scr_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。	①整数型 FrameId [I] フレームID ②リスト型 size [0] フレームのデータサイズ(バイト) リストの先頭に整数型の値が格納される。	①0～4999 内部管理IDのためScr_ConvertXID2FrameID()で取得した値を使うこと。 ②0～255 ③②の数分セットすること	①整数型：CT_result ctdef.pyのCT_resultを参照	・指定したフレームIDのデータサイズ(バイト)を取得する。
	Scr_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。	①整数型 SignalId [I] シグナルID ②リスト型 size [0] シグナルのデータサイズ(バイト) リストの先頭に整数型の値が格納される。	①0～29999 内部管理IDのためScr_ConvertXID2SignalID()で取得した値を使うこと。 ②0～255	①整数型：CT_result ctdef.pyのCT_resultを参照	・指定したシグナルIDのデータサイズ(バイト)を取得する。
DataSend	Scr_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)	①整数型 FrameId [I] フレームID ②整数型 size [I] 送信データサイズ(バイト) ③bytearray data [I] 送信データ	①0～4999 内部管理IDのためScr_ConvertXID2FrameID()で取得した値を使うこと。 ②0～255 ③②の数分セットすること	①整数型：CT_result ctdef.pyのCT_result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。
	Scr_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)	①整数型 SignalId [I] シグナルID ②整数型 size [I] 送信データサイズ(バイト) ③bytearry data [I] 送信データ	①0～29999 内部管理IDのためScr_ConvertXID2SignalID()で取得した値を使うこと。 ②0～255 ③②の数分セットすること	①整数型：CT_result ctdef.pyのCT_result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
	Scr_SendDaData	指定ポートのDAの値を変更する	①整数型 PortNum [I] DAポート番号 ②整数型 value [I] 値	① SCR_DA_CHANNEL_0 SCR_DA_CHANNEL_1 ②値 = (0.0000～5.0000(V)) * 10000	①整数型：CT_result ctdef.pyのCT_resultを参照	・指定されたポートのDAの値を変更する。
	Scr_SendGpoData	指定ポートのGPOの値を変更する	①整数型 PortNum [I] GPOポート番号 ②整数型 value [I] 値	① SCR_GPIO_CHANNEL_0 SCR_GPIO_CHANNEL_1 SCR_GPIO_CHANNEL_2 SCR_GPIO_CHANNEL_3 ② SCR_GPIO_BIT_VALUE_LOW SCR_GPIO_BIT_VALUE_HIGH	①整数型：CT_result ctdef.pyのCT_resultを参照	・指定されたポートのGPOの値を変更する。 ・値は、HI(1)、LO(0)を設定すること。
	Scr_SendAnyCANFrame	任意のCAN Frameデータを送信する	①整数型 BusId [I] バスID ②整数型 SidOnly [I] 拡張ID使用有無 ③整数型 Sid [I] 標準フォーマットのベースID 11ビット ④整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑤整数型 isCANFD [I] CAN-FDフラグ ⑥整数型 BitRateSwitch [I] CAN FDデータフェーズ高速化 ⑦整数型 DataLength [I] 送信データサイズ ⑧bytearray Data [I] 送信データ	① SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_BUS_ID_CAN3 ② 1：ベースIDのみ 0：ベースID＋拡張ID ③ CAN：0x00～0x7FF LIN：0x00～0x3F ④CAN:0x00～0x3FFFF ⑤ 1：CAN FDフレーム 0：CANフレーム ⑥1：有効、0：無効(⑤がCAN FDフレームの場合のみ利用可能) ⑦0～255 ⑧⑦の数分セットすること	①整数型：CT_result ctdef.pyのCT_result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、Scr_SendFrameData/Scr_SendSignalDataを使用する)
	Scr_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	①整数型 number CycleData番号 ②整数型 enable CycleData送信有効/無効 ③整数型 cycle 周期時間(ms) ④整数型 BusId [I] バスID ⑤整数型 SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑥整数型 Sid [I] 標準フォーマットのベースID 11ビット ⑦整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑧整数型 isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑨整数型 BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑩整数型 DataLength [I] 送信データサイズ(バイト) ⑪bytearray Data [I] 送信データ	①0～14 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	①整数型：CT_result ctdef.pyのCT_result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。

	Scr_SendMultiPacketData2	MultiPacketデータを送信する	①整数型 fs [I] FS値 ②整数型 bs [I] BS値 ③整数型 STmin [I] 最低送信間隔 ④整数型 timeout [I] タイムアウト時間 ⑤整数型 BusId [I] バスID ⑥整数型 SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦整数型 Sid [I] 標準フォーマットのベースID 11ビット ⑧整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨整数型 isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩整数型 BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪整数型 RecvBusId [I] バスID ⑫整数型 RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬整数型 RecvSid [I] 標準フォーマットのベースID 11ビット ⑭整数型 RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮整数型 RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯整数型 RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰整数型 DataLength [I] 送信データサイズ(バイト) ⑱bytearray Data [I] 送信データ ⑲整数型 AddressFormatType [I] ⑳整数型 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) ⑤SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_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 ⑪ SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_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 ⑲ MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF	①整数型：CT_result ctdef.pyのCT_result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Scr_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 ・Scr_SendMultiPacketData2()で送受信情報を設定し、データを送信する。 ・Scr_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。 ・Scr_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。
	Scr_ResponseMultiPacketData	MultiPacketへの応答内容設定	①整数型 fs [I] FS値 ②整数型 bs [I] BS値 ③整数型 STmin [I] 最低送信間隔 ④整数型 timeout [I] タイムアウト時間 ⑤整数型 BusId [I] バスID ⑥整数型 SidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑦整数型 Sid [I] 標準フォーマットのベースID 11ビット ⑧整数型 Eid [I] 拡張フォーマットの拡張ID 18ビット ⑨整数型 isCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑩整数型 BitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑪整数型 RecvBusId [I] バスID ⑫整数型 RecvSidOnly [I] 拡張ID使用有無(TRUE：ベースIDのみ、FALSE：ベースID＋拡張ID) ⑬整数型 RecvSid [I] 標準フォーマットのベースID 11ビット ⑭整数型 RecvEid [I] 拡張フォーマットの拡張ID 18ビット ⑮整数型 RecvIsCANFD [I] CAN-FDフラグ(1：CAN-FD、0：CAN) ⑯整数型 RecvBitRateSwitch [I] 転送速度高速化(1：有効、0：無効) ⑰整数型 DataLength [I] 送信データサイズ(バイト) ⑱bytearray Data [I] 送信データ ⑲整数型 AddressFormatType [I] ⑳整数型 N_TA [I]	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) ⑤SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_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 ⑪SCR_BUS_ID_CAN0 SCR_BUS_ID_CAN1 SCR_BUS_ID_CAN2 SCR_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 ⑲ MULTI_PACKET_ADDRESS_FORMAT_TYPE_NORMAL：Normal MULTI_PACKET_ADDRESS_FORMAT_TYPE_EXTENDED：extended ⑳0x00～0xFF	①整数型：CT_result ctdef.pyのCT_result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 ・Scr_ResponseMultiPacketData()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。 ・Scr_GetResponseMultiPacketStatus()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。 ・Scr_GetResponseMultiPacketStatus()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。
	Scr_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_SIZE_MAX ②①のサイズ分のメモリを確保	①整数型：CT_result ctdef.pyのCT_result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_SIZE_MAX(単位：Bytes)
DataRecieve	Scr_GetFrameData	指定Frameの最新データを取得する	①整数型 FrameId [I] フレームID ②整数型 size [I] 受信フレームデータサイズ(バイト) ③bytearray data [O] 受信フレームデータ ④リスト型 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)) リストの先頭に整数型の値が格納される。	①0～4999 内部管理IDのため Scr_ConvertXID2FrameID()で取得した値を使うこと。 ②0～255 ④0～0xFFFFFFFFFFFFFFFF	①整数型：CT_result ctdef.pyのCT_result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。
	Scr_GetSignalData	指定Signalの最新データを取得する	①整数型 SignalId [I] シグナルID ②整数型 size [I] 受信シグナルデータサイズ(バイト) ③bytearray data [O] 受信シグナルデータ ④リスト型 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)) リストの先頭に整数型の値が格納される。	①0～29999 内部管理IDのため Scr_ConvertXID2SignalID()で取得した値を使うこと。 ②0～255 ④0～0xFFFFFFFFFFFFFFFF	①整数型：CT_result ctdef.pyのCT_result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
	Scr_GetAdData	指定ポートのADの値を取得する	①整数型 PortNum [I] ADポート番号 ②リスト型 value [O] 値 リストの先頭に整数型の値が格納される。 ③リスト型 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)) リストの先頭に整数型の値が格納される。	① SCR_AD_CHANNEL_0 SCR_AD_CHANNEL_1 ②値 = (0.0000～50.0000(V)) * ③0～0xFFFFFFFFFFFFFFFF	①整数型：CT_result ctdef.pyのCT_result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。

	Scr_GetGpiData	指定ポートのGPIの値を取得する	①整数型 PortNum [I] GPIポート番号 ②リスト型 value [O] 値 　リストの先頭に整数型の値が格納される。 ③リスト型 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)) 　リストの先頭に整数型の値が格納される。	① SCR_GPIO_CHANNEL_0 SCR_GPIO_CHANNEL_1 SCR_GPIO_CHANNEL_2 SCR_GPIO_CHANNEL_3 ②HI(1)、LO(0) ③0～0xFFFFFFFFFFFFFFFF	①整数型：CT_result 　ctdef.pyのCT_result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、HI(1)、LO(0)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。
	Scr_GetMultiPacketStatus	MultiPacket状態を取得する	①bytearray Data [O] 受信データ	①4095Byteのメモリを確保すること。	①整数型：CT_result 　ctdef.pyのCT_resultを参照 ②整数型 データ有無状態「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 ③整数型 受信データサイズ 1～4095 データがない場合は0	・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Scr_SendMultiPacketData2()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、取得内容②が「1」となる。 ・取得内容②が「データがある」と「FC情報」の場合のみ取得内容③と引数①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。
	Scr_GetResponseMultiPacketStatus	MultiPacketの応答結果を取得する	①bytearray Data [O] 受信データ	①4095Byteのメモリを確保すること。	①整数型：CT_result 　ctdef.pyのCT_resultを参照 ②整数型 データ有無状態「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 ③整数型 受信データサイズ 1～4095Byte データがない場合は0	・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Scr_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、取得内容②が「1」となる。 ・取得内容②が「データがある」と「FC情報」の場合のみ取得内容③と引数①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。