abs_agent ユーザーマニュアル

abs_agent User Manual Rev A.2.37

abs_agent version: 2.37

2025/6/15

オールブルーシステム (All Blue System)

ウェブページ: www.allbluesystem.com

コンタクト:contact@allbluesystem.com

1　はじめに

このマニュアルでは abs_agent プログラムの機能や使い方について詳しい情報を提供します。

abs_agent プログラムは、センサーデバイスや I/O 装置をコントロールするシステムを簡単に構築するための動作環境を提供します。クラウドへデータを定期的に送信するような大規模なデータを扱うシステムや、LAN または組み込み環境等の処理スピードが要求されるセンサーネットワークシステムも、テキストエディタでスクリプトを作成するだけで簡単に構築することができます。Webサーバーや Web API 機能も搭載していますので、アプリケーションサーバーとして動作させることもできます。

abs_agent は、システムを動作させる現場でテキストエディタだけで簡単にシステムを構築・変更できます。このため、特にプロトタイプを作成する段階で、機能や動作の変更を繰り返す場合などに適しています。また同時に、abs_agent は長時間の運用にも安定して動作させることができるように設計しています。サーバープログラムはネイティブにコンパイルされていますので高速動作して、使用するメモリリソースを最低限に抑えています。ユーザースクリプト部分も Lua の特徴である高速動作と、abs_agent が持つスクリプトエンジンのキャッシュ機能によって、スクリプト起動・終了を繰り返すようなイベント処理でもでもパフォーマンスが落ちることはありません。

abs_agent は汎用のゲートウェイ装置に組み込んでセンサーネットーワークを構築するのに最適です。また、ユーザー毎に現地でのカストマイズが必須なシステム製品に利用する場合にも適しています。

abs_agent では標準的でオープンな技術を使用してシステムを構築しています。ネットワーク通信には TCP/IP や、その上で動作する HTTP, MQTT 等のプロトコルをサポートしています。また、スクリプト制御にはスピードと安定性で実績のある Lua を使用しています。このため、システムを入れ替えたりバージョンアップする場合でも最低限のコストで移行することも可能です。

abs_agent プログラムは、市販されている Linuxボード(Raspberry Pi等)や組み込み用の Linux システムにインストールすることができます。ハードウエアタイプ別にバイナリパッケージを提供していますので、直ぐにインストールして使用できます。お客様独自のハードウエア向けに、abs_agent プログラムを移植して特注パッケージを提供することもできます。この場合にはお客様独自のハードウエア機能を、追加のライブラリ関数やイベント機能として提供することもできます。

このマニュアルでは "Raspberry Pi" ハードウエアと"Intel x86"タイプのハードウエアで動作する、バイナリパッケージの使用方法について説明しています。特定のハードウエア向けの機能を除いて、このマニュアルで説明している機能は全てのプラットフォームで使用することができます。

abs_agent には動作ログをネットワーク経由で出力する機能があります。ネットワーク出力されたログを収集するログサーバーは、abs_agent が動作しているコンピュータとは別の Windows PC 上にインストールします。複数の abs_agent のログ出力をまとめて１つのログサーバーで管理することができます。ログサーバーはオールブルーシス

テムが提供する ABS-9000 LogServer インストールキットを使用してインストールしてください。ログサーバーの詳しい機能については、ABS-9000 DeviceServerユーザーマニュアル中の "ログサーバー" と "ログコンソールプログラム"の項を参照して下さい。

2 表記について

2.1 著作権および登録商標

2.2 連絡先

オールブルーシステム (All Blue System)
ウェブページ http://www.allbluesystem.com
メール contact@allbluesystem.com

3 abs_agent の機能・特徴

abs_agent には下記のような機能・特徴があります。

(1) Linux 上のデーモンプロセスとしてサーバーが動作して、定期的なセンサーデータ取得や他のシステムとの通信、I/O 操作などを自動で行います。

(2) システムの動作は、ユーザーが定義した Lua スクリプトでコントロールすることができます。簡単なテキストエディタを使用するだけで複雑なシステムを簡単に構築できます。システムを設置した現地でのデバッグ作業も容易に行うことができます。スクリプトエンジンには Lua[1] (ver5.2.3) を使用していて、高速動作とスクリプト記述の柔軟さを実現しています。

[1] Lua (http://www.lua.org/about.html)

(3) コンピュータに接続されたシリアルポートに簡単にアクセスできます。文字列や各種バイナリデータを受信したときの動作を作成できます。また、シリアルポートには計測器や I/O 装置、Arduino[2] CPU ボード等を接続することができます。

(4) MQTT[3] ブローカとの接続機能を使用することができます。ブローカにトピックと QoS を指定して任意の文字列メッセージやバイナリデータを送信したり、購読対象に指定したトピックをブローカから受信して abs_agent のイベントハンドラスクリプトで処理することができます。abs_agent で定義したエンドポイントを使用して複数の MQTT ブローカと同時に接続することもできます。

(5) abs_agent で発生するイベントやユーザースクリプトから詳細なログ情報を出力できます。ログは、abs_agent が動作しているコンピュータとは別に、Windows PC 上に設置した ABS-9000 LogServerインストールキットでセットアップされたログサーバー機能を使用します。ログサーバーは、複数の abs_agent が動作しているコンピュータからのログ情報を一括して保存、管理できます。

(6) マイクロソフトのエクセル VBA や Win32 アプリケーションからリモートの abs_agent の機能を利用するために、API (DLLライブラリ)も提供しています。 API を使用して、ネットワーク接続された複数のクライアントPC で、エクセル(VBA)に簡単にデバイスの情報(A/D 変換値等)を取り込んで、グラフに加工することも簡単にできます。

(7) Raspberry Pi 用にビルドされた abs_agentでは、ハードウエア機能に直接アクセスする機能を提供しています。GPIO, SPI, I2C を操作するためのライブラリ関数を提供しています。また、GPIO のデータが変化した場合には abs_agent 側でイベントハンドラを実行するように設定できます。

(8) HTTP サーバー機能を内蔵していますので、abs_agent 上で Webページを公開したり、Webアプリケーションを動作させることができます。

(9) Web API 機能を使用すると、Webアプリケーションプログラムから abs_agent で管理されたデバイスやスクリプトを操作することができます。Web API にアクセスするときのユーザーアカウントを abs_agent で作成・管理することで、Web APIを使用する場合にログイン認証機能を使用することができます。

(10) フラッシュストレージなどのデバイスから OS や abs_agent が起動される場合に備えて、abs_agent ではディスク書き込み動作を最小限に抑える様にしています。ログ出力など頻繁な出力を行う機能はネットワーク経由でメッセージを送信して、別途設置したログサーバー上で集中管理できるようにしています。

(11) abs_agent には高速で動作するインメモリデータベースが内蔵されています。センサ等から取得したデータ値に任意のキー名を指定してデータベースに保存することができます。データベースに保存したデータは、スクリプト中やクライアントプログラムから簡単に集計データを取得できます。

[2] http://www.arduino.cc/
[3] MQTT http://mqtt.org/

(12) Webブラウザや他のWeb アプリケーション間とのリアルタイム通信を行う WebSocket サーバー機能を内蔵しています。abs_agent のユーザースクリプトやイベントハンドラから特定の WebSocketクライアントや全てのWebSocketクライアントに対してテキストやバイナリフレームを送信することができます。また、クライアントから受信した WebSocket フレームをイベントハンドラスクリプトで簡単に処理することができます。

以下の項ではこれらの abs_agent が持つ代表的な機能の概略を説明します。

ここで説明した基本機能に加えて abs_agent にはデータベース保存(パーマネントデータ) の為の STORAGE 機能や、ベクトル・行列を扱う MATRIX 機能を使用できます。これらの機能については "abs_agent Advanced User Manual" を参照して下さい。

3.1 デバイス管理

abs_agent で操作可能なデバイスには以下のものがあります。

- **シリアル接続された Arduino CPUボードや計測器、I/O 装置、XBee 802.15.4、XBee-ZB(Zigbee) モジュール**

  シリアルポート経由で接続された、I/O 装置、計測機器などを接続できます。Firmata プロトコルで Arduino 互換デバイスと接続するこもできます。最大１０台まで接続可能です。

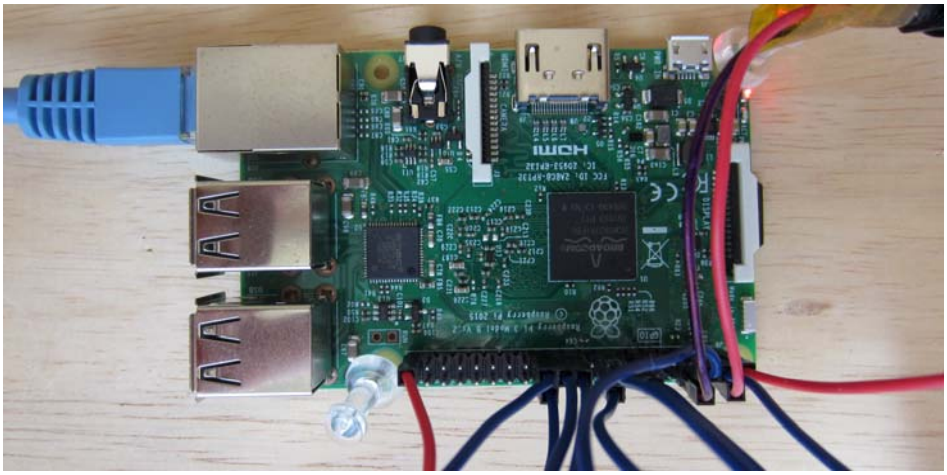
  また、シリアルポートに XBee 802.15.4 や XBee-ZB(Zigbee)モジュールを接続すると、リモートに設置した複数の XBee デバイス間でデータのやり取りを行う事もできます。

- **MQTT エンドポイント**

  MQTT ブローカとのネットワーク接続を表した通信端点です(MQTT エンドポイント)。MQTT エンドポイントに対して任意のトピックとメッセージを送信することができます。MQTT ブローカに購読をリクエストして、配信されてきたデータはエンドポイント毎にキューイングされて abs_agent のイベントハンドラで処理することができます。複数のエンドポイントを作成して機能毎に通信経路を分けたり、複数のMQTT ブローカを同時に接続することもできます。

- **Raspberry Pi ハードウエアに接続されたデバイス**

  Raspberry Pi用にビルドされた abs_agent では、I2C、SPI、GPIO 各ポートに接続されたデバイスにアクセスすることができます。



(abs_agent を使用して、Raspberry Pi3 に接続したGPIO や I2Cデバイスを制御している様子)

この他にも、Lua 標準ライブラリを利用して通常ファイルシステムデバイス、デバイスファイルへアクセスするようなスクリプトを記述することができます。

3.2 ログ出力

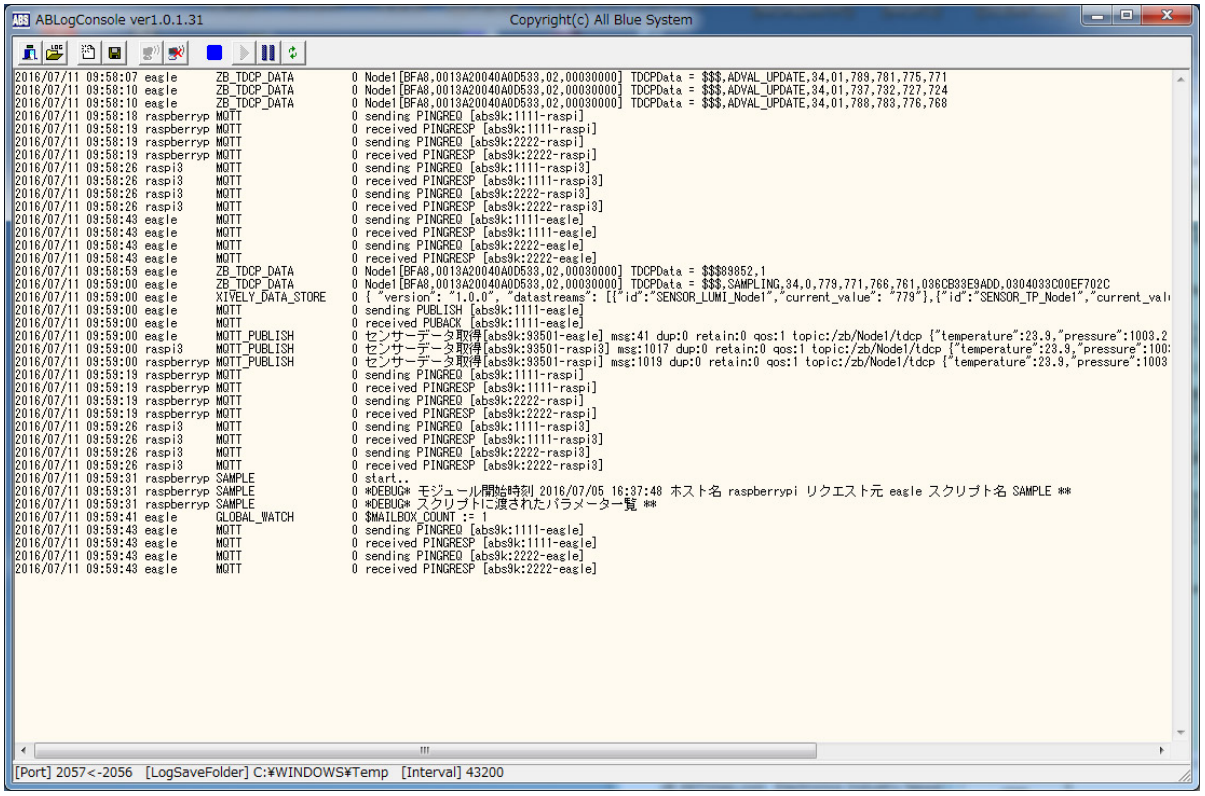
abs_agent にはネットワーク経由で各種ログメッセージを専用のログサーバーに送信する機能が組み込まれています。

ログサーバーを利用すると abs_agent で検出したエラーやイベント、クライアントアクセスの記録、ユーザーが作成したスクリプトからのメッセージ出力を集中管理することができます。abs_agent で出力するログメッセージは、オールブルーシステムの提供する ABS-9000 LogServer インストールキットでセットアップできます。abs_agent と組み合わせて使用する場合には、ABS-9000 LogServer(ログサーバー)をホームページからダウンロードしてフリーで使用することができます。

複数の abs_agent を設置して、それらから出力されるログを１つのログサーバーで管理することもできます。

ログサーバーは、Windows サービスプログラムとして常にバックグランドで動作していますので、サーバーPC にログイン(Windowsユーザーアカウントのログイン)していない状態でも、ログを記録しています。サーバーPC に定期的にログファイルを出力して、後で動作記録を参照できます。(デフォルトでは “C:¥Program Files¥AllBlueSystem¥Logs” または “C:¥Program Files (x86)¥AllBlueSystem¥Logs” に出力しています)
ログサーバーは高速動作を行うために、通常はメモリ中にログ情報を保管していて、定期的にファイルに書き出す方法を採用しています。

ログコンソールプログラムを起動することで、ログサーバーに出力しているログメッセージをリアルタイムに画面表示することもできます。これは、クライアントプログラムやスクリプト作成時のデバッグなどにも役に立ちます。ログコンソールを使用して、ログサーバーに対して、メモリ中に保管中のログを強制的にファイル出力するように指示することもできます。



（ログコンソールプログラムの表示例。複数設置した abs_agent からのログ情報を Windows 上に設置したログサーバーで集中管理しています）

ログ管理機能の詳細は ″ログサーバー・インストール″ の章を参照してください。

3.3 スクリプト実行機能

abs_agent にはユーザーの目的に応じたシステムを構築するためのスクリプト実行機能が提供されています。

デバイスのI/O 操作やデバイスから取得したデータの加工、定期的なジョブ実行機能、センサーの状態に応じたフローコントロール等を実現するために、abs_agent にはスクリプトエンジンが内蔵されています。

abs_agent のスクリプトエンジンは、高速動作でシンプルかつ柔軟な記述が可能な、Lua（ver5.2.3）を採用しています。

abs_agent は下記の様な場合にスクリプトを実行します

- API からスクリプト実行関数をコール
- abs_agent で定義されたイベントが発生したときにイベントハンドラとしてスクリプトを実行
- agent_script クライアントプログラムからスクリプトを起動

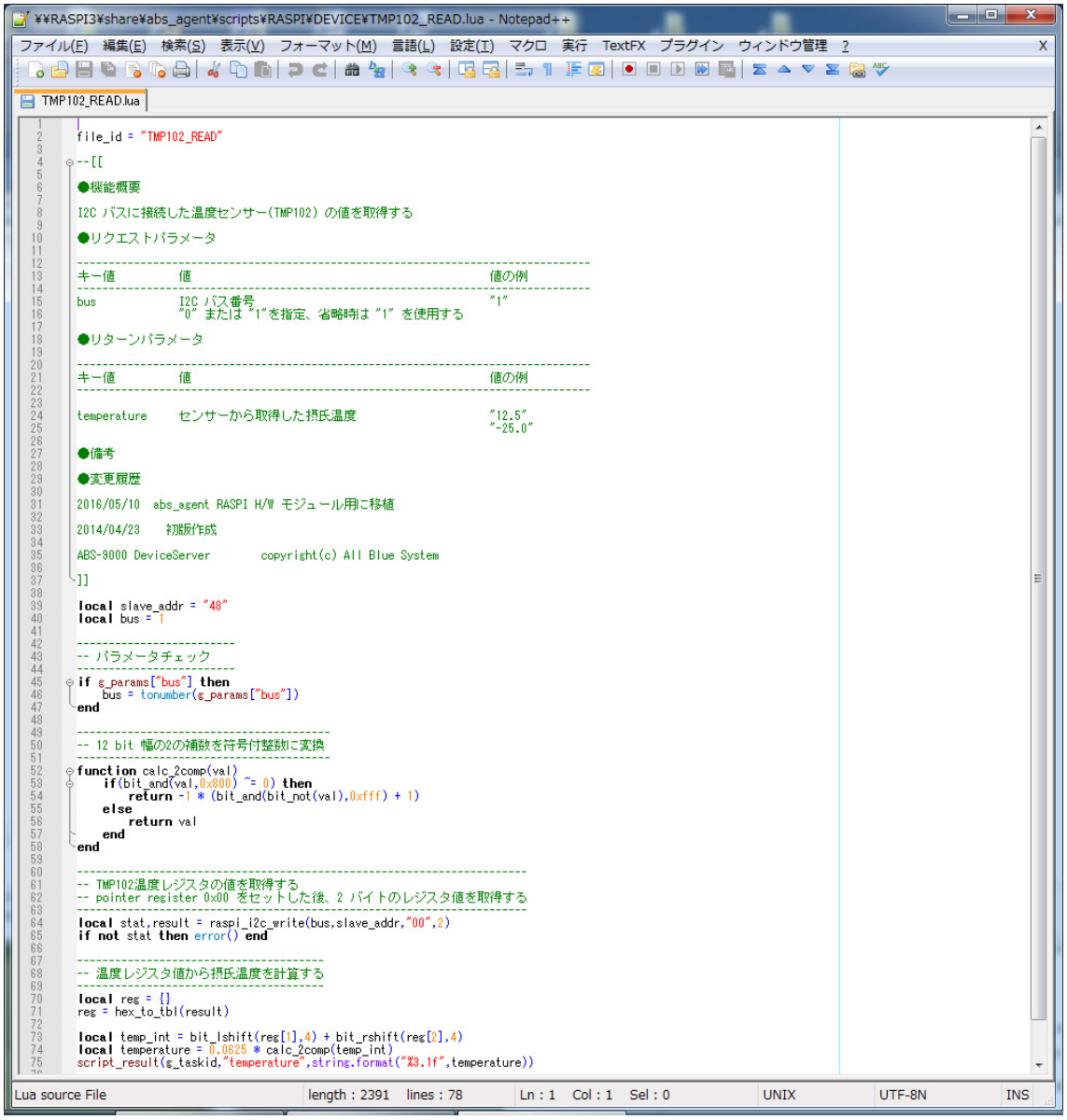
abs_agent のスクリプトエンジンは Lua が持っている基本的なライブラリ機能に加えて、独自の拡張を行っています。これによって abs_agent の持つ様々なサービスモジュール機能をライブラリ関数経由でアクセスできます。

abs_agent は通常 Linux のデーモンプロセスとして動作していますので、Lua 標準ライブラリ中の ioライブラリ，標準出力を使用するライブラリ関数等に関して制限事項があります。 これらの詳しいライブラリ関数の使用方法については “スクリプトライブラリ関数一覧” の章を参照してください。

スクリプトには任意の文字列パラメータを実行時に指定できます。パラメータはキー文字列値と値文字列のペアを複数指定することができます。イベントハンドラとして実行されるスクリプトには、予め決められたイベント発生時の情報がスクリプトパラメータとして渡されます。イベントハンドラ毎に決められたパラメータの詳細については “イベント” の章を参照してください。

また、スクリプトから任意の数のリターンパラメータをコール側に返すこともできます。スクリプト中からライブラリ関数でスクリプト実行した場合は、呼び出されたスクリプト中で設定したこれらのリターンパラメータを、呼び出し側のスクリプトで取得できます。

スクリプトはテキストエディタ(vi, emacs等)で簡単に作成することができます。**日本語を使用する場合には必ず UTF-8(BOMなし) のエンコード形式で保存してください。 UTF-8(BOMあり)や Windows 標準の Shift_JIS 形式、その他のエンコード形式では動作しませんので注意してください**。Linux のスクリプトディレクトリを samba 等のプログラムを使用して Windows 上から共有してスクリプトファイルを操作することもできます。



(スクリプトファイル例。Windows 上のテキストエディタ(NotePad++[4])から Raspberry Pi 上で動作している abs_agent のスクリプトを編集している様子。スクリプトファイル名は "RASPI/DEVICE/TMP102"で、Raspberry Pi の I2C バスに接続した TMP102 温度センサからデータを取得するスクリプトが記述されています)

abs_agent にはローカルやリモートに設置したスクリプトファイルにアクセスするために、クライアントプログラムを提供しています(agent_get, agent_put, agent_script) これらを利用すると、スクリプトを配置する OS 上のディレクトリやファイル名を意識することなく、スクリプト名だけで簡単にスクリプトファイルの新規作成や修正ができます。詳しくは "クライアントプログラム" の章をご覧ください。

スクリプトファイルを直接エディタで作成することも簡単にできます。作成したスクリプトは abs_agent のインストールディレクトリ中の scriptsフォルダ以下にファイル拡張子 (.lua) のファイル名で保存してください。例えば /home/pi ホームディレクトリ以下に abs_agent をインストールした場合には、スクリプトディレクトリは /home/pi/abs_agent/scripts になります。また、スクリプトファイル名やスクリプトファイル格納用のサブディレ

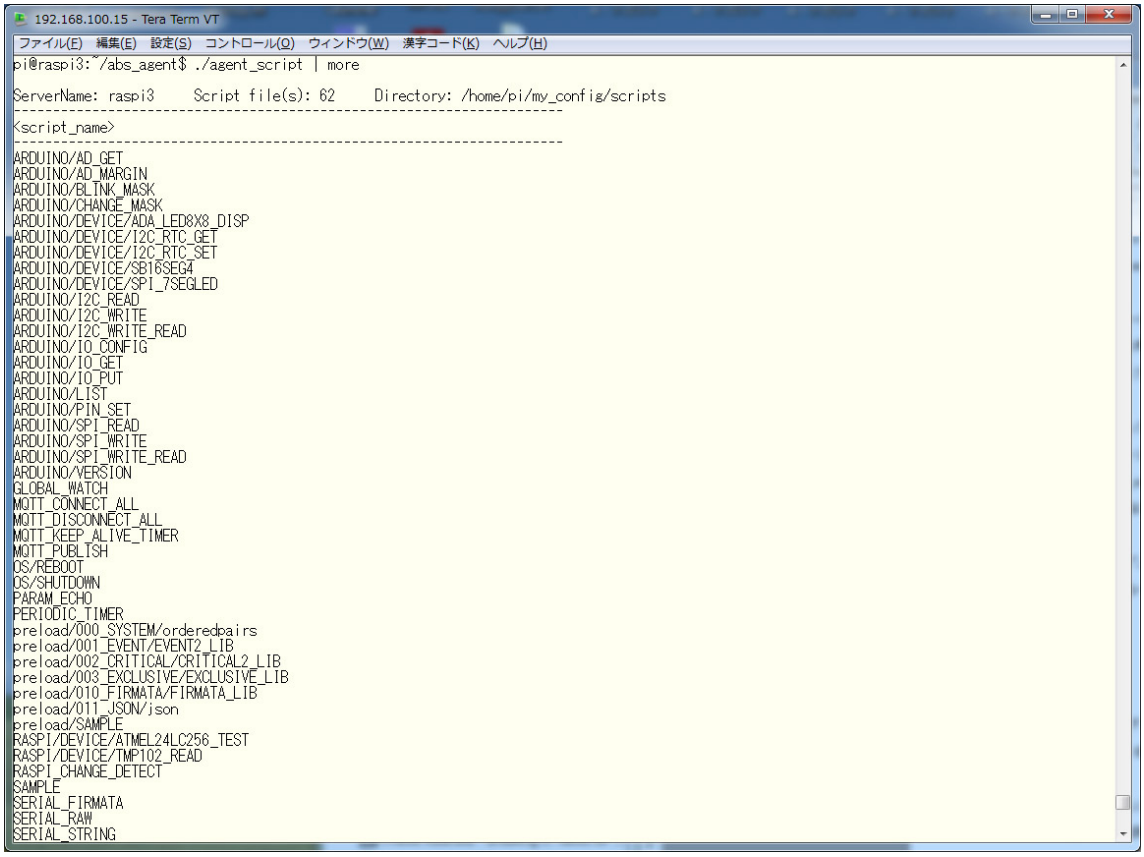
[4] NotePad++ (https://notepad-plus-plus.org/)

クトリ名(後述)に日本語を使用することもできます。(abs_agent が動作しているコンピュータのファイルシステムがサポートしている場合)

スクリプトファイルを作成または修正すると abs_agent の再起動などは不要で直後からアクセスできます。
abs_agent に設定したスクリプトを指定する場合には、ファイルの拡張子部分を除いたファイル名がスクリプト名となります。(ファイル名 “ABC.lua” のスクリプトを実行する場合のスクリプト名は “ABC” になります)。

scripts ディレクトリ内にサブディレクトリを作成して、スクリプトファイルを格納することもできます。この場合には、サブディレクトリ名を ‘/’ 文字で区切ってスクリプト名を指定してください。サブディレクトリ中にサブディレクトリを作成することもできます。例として、scriptsディレクトリ “/home/pi/abs_agent/scripts/PROJECT1/テスト” に格納した、ファイル名 “ABC.lua” のスクリプトを実行する場合のスクリプト名は “PROJECT1/テスト/ABC” になります。

前述のクライアントプログラム (agent_script) を使用すると、全てのサブディレクトリを含めたスクリプト名一覧を取得できます。



(agent_script クライアントプログラムを使用して、ローカルに設置されたスクリプト名一覧を表示している様子)

上記 scripts ディレクトリにはインストール時にデフォルトのスクリプト(イベントハンドラスクリプトを含む)が保管されていますので、ユーザースクリプト作成時の参考にしてください。

Lua 言語の詳しい仕様と資料は http://www.lua.org/、http://www.lua.org/manual/5.2/ を参照してください。
abs_agent で拡張された機能毎のライブラリ関数については “xxxxx API” の章を参照してください。

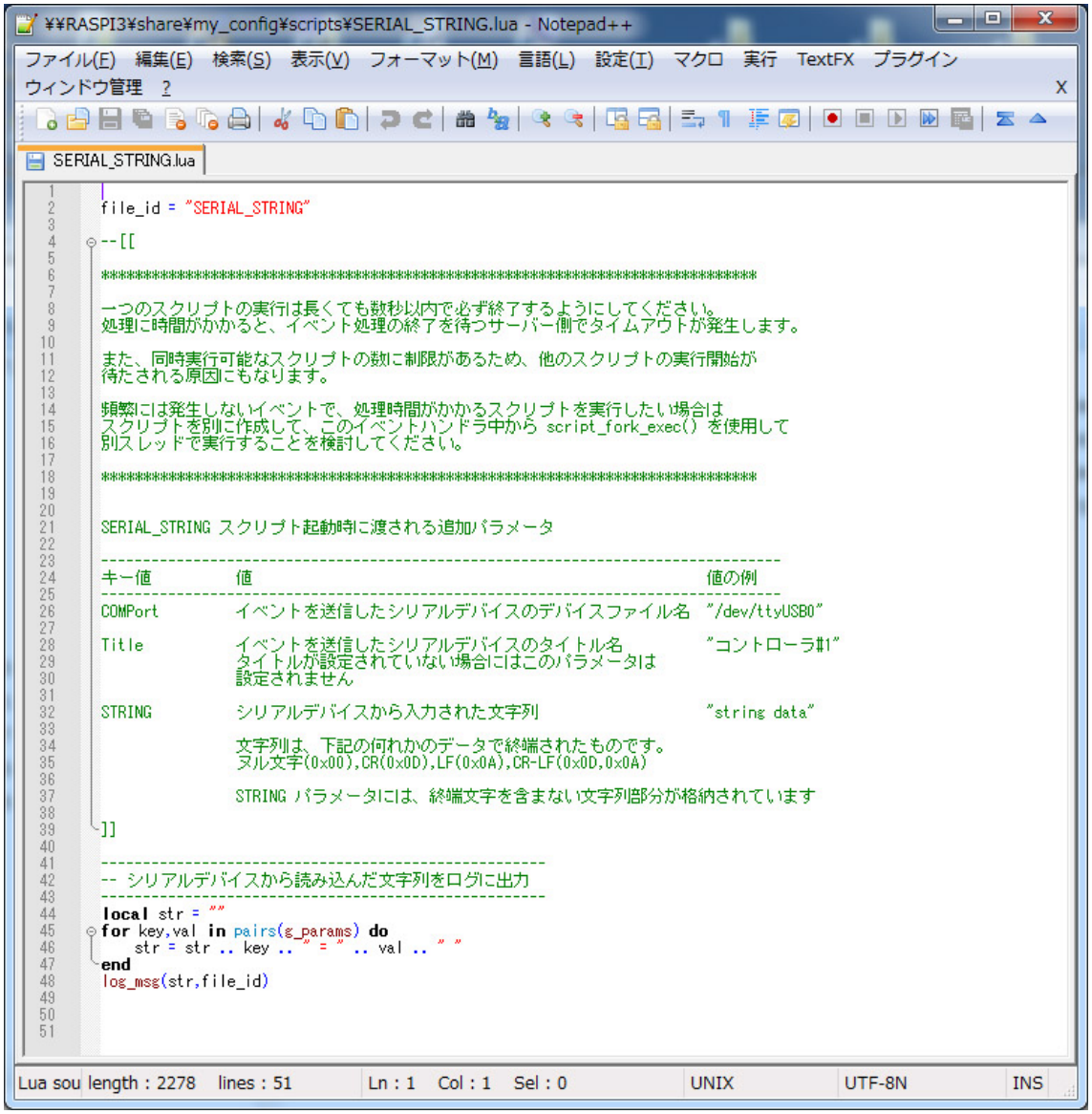
スクリプトと後述のイベントハンドラは デフォルトで 16 個まで同時実行できます。これ以上のスクリプト実行が指示された場合は、他の実行中のスクリプトまたはイベントハンドラが終了してから実行します。スタンダードライセンスの場合には、同時実行可能なスクリプト数の上限を任意の値に変更できます。詳しくは、”サーバーソフトウエア” の章中の ”サーバー設定ファイル(abs_agent.xml)” の項目を参照してください。

3.4 イベントハンドラ実行機能

abs_agent では、内部のタイマーによって決められた間隔やサービスモジュール毎に決められた条件になったときにイベントが発生します。例えば、シリアルポートからデータを受信したり、MQTT ブローカから Publish メッセージを受信した場合、Raspberry Pi の GPIO ポートが変化した場合などがあります。(詳しくは後述)

イベントが発生すると、イベントの種類ごとに予め決められたスクリプトを自動的に abs_agent で実行します。ユーザーはこのスクリプト（イベントハンドラ）の内容をテキストエディタを使って簡単にカスタマイズできます。

イベントハンドラで実行されるスクリプトとユーザーが独自に作成するスクリプトは、共通の Lua スクリプトエンジンを使用しています。イベントハンドラが実行される場合には、そのイベントに関する詳しい情報がリクエストパラメータ(g_params[]) としてスクリプトに渡されイベントハンドラのスクリプト中で利用できます。イベントの種類毎に渡されるパラメータの詳細は “イベント”の章を参照してください。



（シリアルポートから文字列データを受信したときに実行される SERIAL_STRING イベントハンドラの例）

イベントハンドラはユーザー作成のスクリプトと全く同じ様に動作しますので、イベント発生のタイミングで様々な処理を行えます。イベントハンドラ内では abs_agent 用に追加されたライブラリ関数を利用できます。共有変数を利用したデータ共有や保存、他のシステムへのデータ送信、ログ出力等の処理を記述できます。abs_agent で管理された Raspberry Pi GPIO や I2C, SPI デバイスの操作を行うこともできます。

abs_agent で定義されたイベントの種類は以下のものがあります。イベントについての詳しい説明は "イベント"の章を参照して下さい。

イベント名	イベントハンドラ実行条件
PERIODIC_TIMER	１分毎に繰り返し実行されます。
TICK_TIMER	１秒毎に繰り返し実行されます。
SERVER_START	abs_agent 起動時に全てのサービスモジュールが利用可能になった

	ときに１回だけ実行されます。
SERVER_STOP	abs_agent 終了時に実行されます。ただし、agent_shutdown クライアントコマンドを使用した時のみ実行されて、OS シャットダウンした時に abs_agent を強制終了させた場合には実行されません。
SERIAL_STRING	“STRING” タイプに定義されたシリアルポートから文字列を受信する度に実行されます。
SERIAL_FIRMATA	“FIRMATA” タイプに定義されたシリアルポートから１パケット受信する度に実行されます。
SERIAL_XBEE	“XBEE” タイプに定義されたシリアルポートから APIパケットを受信する度に実行されます。
SERIAL_ZB	“ZB” タイプに定義されたシリアルポートから APIパケットを受信する度に実行されます。
SERIAL_RAW	“RAW” タイプに定義されたシリアルポートからデータを受信すると実行されます。
GLOBAL_WATCH	予め監視対象に指定したグローバル共有変数の内容が更新されたときに実行されます。
MQTT_KEEP_ALIVE_TIMER	MQTT サービスモジュールを有効にしたときに、KeepAliveTimer 項目に設定した秒間隔で繰り返し実行されます。
MQTT_PUBLISH	MQTTサービスで設定したエンドポイントから、MQTT PUBLISH メッセージを受信した時に実行されます。
RASPI_CHANGE_DETECT	ライブラリ関数 raspi_change_detect() をコールして GPIO 値を監視している時に、該当する GPIO 値が変化した場合に発生します。（Raspberry pi 用の abs_agent のみで有効）
WEBSOCKET_DATA	WebSocketサーバーに接続したクライアントから、テキストフレームまたはバイナリフレームを受信した時に実行されます。
TCPSERVER_DATA	汎用TCPサーバーを有効にした時に、接続中のクライアントから指定されたフォーマットでフレームを受信した時に実行されます。
UDPSERVER_DATA	汎用UDPサーバーを有効にした時に、データグラムパケットを受信した時に実行されます。

インストールディレクトリ中の scriptsディレクトリ “<abs_agent インストールディレクトリ>/scripts” にはそれぞれのイベントに対してデフォルトのイベントハンドラスクリプトファイルが保管されています。ファイル名はスクリプト名に .lua の拡張子が付きます。このイベントハンドラスクリプトファイルをエディタで修正してカストマイズできます。

上記のイベントハンドラは SERVER_START, SERVER_STOP を除いて全て独立した別スレッドで実行されます。イベント発生条件になる度に、イベントハンドラスクリプトが別スレッドで起動されます。このため、スクリプトファイルに記述する Lua ステートメントはマルチスレッドの環境で実行される点に注意してください。abs_agent 上でグローバル共有変数をアクセスするライブラリ関数などは、自動でマルチスレッド下でのリソース排他制御を行っていま

すので特に意識することなくスクリプトを記述できます。ユーザーが外部 I/O 装置や GPIO、センサ等をアクセスする場合には、ライブラリ関数で提供するクリティカルセクションやイベント等の API を使用して独自の排他制御を行うことができます。

ユーザースクリプトやイベントハンドラは abs_agent 上では同時に 16 個まで実行できます(コンフィギュレーションで変更可能)。これ以上のユーザースクリプトやイベントハンドラの同時実行される場合には、他の実行中のスクリプトまたはイベントハンドラが終了してから実行します。このため、イベントハンドラを記述する場合は、できるだけイベントハンドラスクリプトの実行時間を短くするようにしてください（数秒程度）。イベントハンドラ中からライブラリ関数 script_fork_exec() 等を使用して、処理時間がかかる部分を別スレッドで実行させることもできます。このようにすると、イベントハンドラの処理自身は直ぐに終了させることができます。

3.5 グローバル共有データ機能

abs_agent ではスクリプトやイベントハンドラから共通してアクセスできるデータ領域が用意されています。データの使用目的に応じて以下の機能を利用できます。

- キー・バリュー形式で保存する**グローバル共有データ機能**
- 任意のチャンネルに複数の文字列リストを保存する**グローバル共有文字列リスト機能**
- 任意のチャンネルにバイト単位にアクセス可能なメモリー領域を提供する**グローバル共有メモリエリア機能**

これらの共有データは、全てのスクリプトやイベントハンドラから常にアクセスすることができます。マルチスレッドでスクリプトが平行して動作している場合でも、内部で排他制御が自動で行われますので、利用する側では特に意識することなく安全に使用できます。共有データ領域はスクリプトやイベントハンドラの実行終了後も最後に更新した状態を保持していますので、実行タイミングが異なるスクリプト間でのデータ共有も簡単に実現できます。

abs_agent クライアントコマンドや WebAPI 等からもこれらのデータ共有領域へのアクセスが可能です。実行中のスクリプトの動作を外部から制御したり、開発・運用中にシステムの内部状態を確認・更新することも簡単にできます。

グローバル共有データ機能の詳細については "グローバル共有データAPI", "グローバル共有文字列リストAPI","グローバル共有メモリエリアAPI" の章を参照してください。また、WebAPI で利用可能なグローバル共有機能については "HTTPサーバー&WebAPI" の章も併せて参照してください。

3.6 時系列データベース機能

スクリプトやイベントハンドラから数値データ集計用のインメモリ・時系列データベース("FASTDB")機能を利用できます。任意のキーを指定して数値データを登録することで、一定期間毎の集計値を簡単に計算することができます。

データ保存やデータアクセス時にはメモリ領域のみを使用して高速に動作します。このためファイルシステムへの負荷が低減され長期間のデータ保存を安定して運用することができます。

時系列データベースを利用するライブラリ関数は全て、マルチスレッドの環境で確実に操作できるように内部でリソ

ースの管理が行われています。このため、イベントハンドラやユーザースクリプトからスレッド間の競合やデーター貫性のロックを意識することなく使用できます。

時系列データベース機能の詳細については "時系列集計用インメモリデータベースAPI"の章を参照してください。また、時系列データベースを利用したWebアプリケーション例として "APPENDIX(A)" 章に集計グラフ作成とRawデータチャートの例の解説がありますので参照してください。また、"APPENDIX(B)" 章では Raspberry Pi に接続した各種LCDデバイスに集計グラフを表示するアプリケーションを説明しています。

3.7 MQTTクライアント機能

MQTT ブローカから配信されるデータを購読(Subscribe)したりデータの配信(Publish)を行うことができます。

abs_agent のサーバー設定ファイルにエンドポイントを登録すると、abs_agent 起動時に自動的に MQTT ブローカーに接続して、購読対象になっているトピックを受信することができます。ブローカからデータを受信すると、イベントハンドラが起動されますので、処理内容に応じたスクリプトを記述することで簡単にシステムを構築できます。

MQTT ブローカに対して PUBLISH メッセージを送信するライブラリ関数ではトピック名とペイロードデータ、QoS（MQTT 送達品質レベル）を指定してスクリプト中から任意のタイミングでメッセージを送信できます。

abs_agent では複数のブローカに対する同時接続や、ブローカ毎にエンドポイントも複数作成することもできます。

MQTTクライアント機能の詳細については "MQTTクライアントAPI"の章を参照してください。MQTTブローカーへの接続やエンドポイントのセットアップ方法については、"サーバープログラム・設定ファイル"章中の "MQTT" の項を参照して下さい。

3.8 クライアントプログラムによる操作

abs_agent で作成したネットワークシステムは基本的には自立して動作しています。定期的にセンサーにアクセスしてデータを取得後、クラウドに送信するなどの動作を人手を介さずに全て自動で行います。また、MQTT ブローカやシリアルポートで発生したイベントをトリガーに、予め決められた処理スクリプトを実行していきます。

これらの他に、ユーザーが任意のタイミングで abs_agent のネットワークシステムを操作するために、Linux のシェルから実行可能なクライアントプログラムを用意しています。これらのクライアントプログラムは複数のコンソールアプリケーションに分かれていて、それぞれの機能は以下になります。これらクライアントプログラムの詳しい使用方法については、"クライアントプログラム" の章を参照してください。

クライアントプログラム名	機能
agent_stat	サーバーステータスの表示、ライセンス登録。
agent_hosts	リモートアクセス可能なクライアントコンピュータの管理
agent_shutdown	abs_agent サーバーのシャットダウン
agent_script	サーバー上のスクリプトを実行、スクリプト一覧表示

agent_data	グローバル共有変数の参照や更新
agent_task	サーバーで現在実行中のスクリプト・イベント一覧表示や強制終了
agent_get	サーバー上に設置されたスクリプトの表示・ダウンロード
agent_put	サーバーにスクリプトを新規作成や既存スクリプトの更新
agent_strlist	グローバル共有文字列リストの参照や更新
agent_webuser	Web API アクセス時に使用するユーザーアカウント管理
agent_session	Web API アクセス時に作成するセッション情報管理
agent_fastdb	時系列集計用インメモリデータベース管理
agent_shmem	グローバル共有メモリエリアの参照や更新、ファイルへのセーブ・ロード
agent_mgcp	リモートに設置した MGCP デバイスの管理とリモートコマンドの実行
agent_websocket	WebSocket クライアント一覧やクライアントへのデータフレーム送信
agent_xbee	XBee 802.15.4 デバイス管理
agent_zb	XBee-ZB Zigbee デバイス管理
agent_copycfg	動作環境(スクリプト、Webアプリ、マスターファイル等)の複製・チェック
agent_serial	シリアルポートの文字列バッファ操作
agent_net	汎用 TCPサーバー(ModbusTCP)、TCPクライアント管理

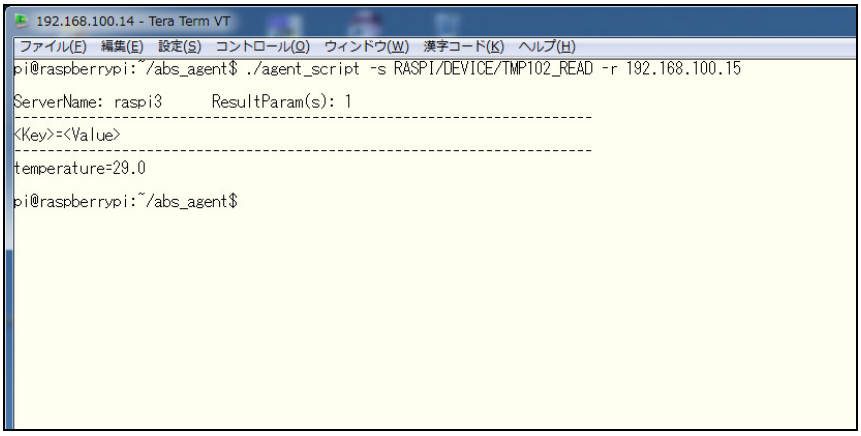
クライアントプログラムは、abs_agent をインストールしたディレクトリに配置されています。必要に応じてシェルの実行パスをインストールディレクトリに追加指定してください。

クライアントプログラムは任意のコンピュータに複数設置することができます。たとえば、１つの abs_agent サーバーを対象に、リモート操作を行うクライントプログラムを複数のコンピュータに設置することができます。このとき、クライアントプログラム自身を動作させるためのライセンスは別途必要ありません。ただし、abs_agent サーバー側ではリモートアクセスを許可するための agent_hosts コマンドで、複数のリモートアクセスを許可するためにスタンダードライセンスが必要になります。

クライアントプログラムが動作するコンピュータと操作対象の abs_agent のハードウエアタイプは異なっていても構いません

クライアントプログラムは、ローカルサーバーで動作している abs_agent 以外にもリモートコンピュータで動作している abs_agent に対しても操作することができます。(一部のコマンドを除く)

これらのリモート操作を行う場合に、クライアントプログラムと abs_agent サーバーが動作しているコンピュータのハードウエアタイプが異なっていても操作できます。たとえば、Intel x86 ハードウエア上の Debian GNU/Linux で動作させている abs_agent に対して、Raspberry Pi(Raspbian) にインストールした abs_agent のクライントプログラムから操作することができます。もちろん、スクリプト中からリモートアクセスする API でも同様に、動作してるハードウエアタイプを意識する必要はありません。



（リモートの Raspberry Pi で動作している abs_agent のスクリプトをクライアントプログラムから実行している様子。リモート側の I2C インターフェイスで接続された温度センサの値を読み込んでいます）

3.9 リモートアクセス機能

複数の abs_agent を設置して、それぞれの abs_agent 間で協調して動作するような分散処理システムを構築することができます。複数のコンピュータに処理を分けることで、大量のセンサーノードやリモート制御機器をコントロールすることが可能になります。

物理的に離れた場所にある複数のセンサーネットワークをまとめて管理したい場合や、タスクや機能ブロック毎に処理を行うコンピュータを分離したいときにも利用できます。信頼性を確保するために常時複数の abs_agent を平行して動作させておいて、１つのコンピュータダウンした場合でもセンサーデータの収集を継続して行うようなシステムに応用できます。

リモートコンピュータ上の abs_agent へのアクセスはスクリプトやイベントハンドラから、リモートアクセス用に作成されたライブラリ関数を使用して行います。スクリプトは簡単に修正できますのでデバッグも簡単で、分散システムの構築や運用・拡張時にも簡単に対応できます。万が一システムに障害が発生したときにも、構成の変更を簡単に実行することができます。

abs_agent には分散システムを構築するために以下の機能が用意されています。

- リモートに設置した abs_agent のスクリプトやイベントハンドラを実行

リモートに設置した任意のスクリプトやイベントハンドラを実行できます。リターンパラメータを取得したり、別スレッドで実行することでスクリプトの終了を待たずに別の処理を平行して実行できます。詳しくは、スクリプト実行 API ライブラリ関数のリファレンスを参照してください。

- リモートに設置した abs_agent のグローバル共有データを直接参照または更新

リモートに設置した abs_agent のグローバル共有データ領域を参照したり、更新することができます。詳しくは、グローバル共有データAPI ライブラリ関数のリファレンスを参照してください。

- リモートからのスクリプト実行やデータ領域の参照・更新を、予め許可したサーバーのみに限定

予め指定した複数の abs_agent に限定してリモートアクセスを許可します。詳しくは、"クライアントプログラム" 章中の agent_hosts コマンドの項を参照してください。

- abs_agent とクライアント間の通信データを暗号化

クライアントプログラムから abs_agent にアクセスする場合や、複数の abs_agent 間でライブラリ関数を使用した通信を行う場合には、通信経路に流れるデータパケットは暗号化され外部からのアクセスを防止しています。また、インターネット間で通信する場合には必要に応じて VPN や専用線等のより秘匿性の高い手段を併用することで高いセキュリティを維持したシステムを構築することができます。

3.10 Microsoft Excel VBA, Win32 アプリからのコントロール

abs_agent インストールキットには、Windowsアプリ(Win32)やマイクロソフト・エクセル(32ビット版)から、リモートに設置した abs_agent をコントロールするための DLLライブラリ(XASDLCMD.DLL)が添付されています。

既存のアプリケーションに組み込む事や、独自の Windows クライアントプログラムを作成して、abs_agent で管理されたグローバル共有変数のアクセスや、ユーザースクリプトの実行ができます。

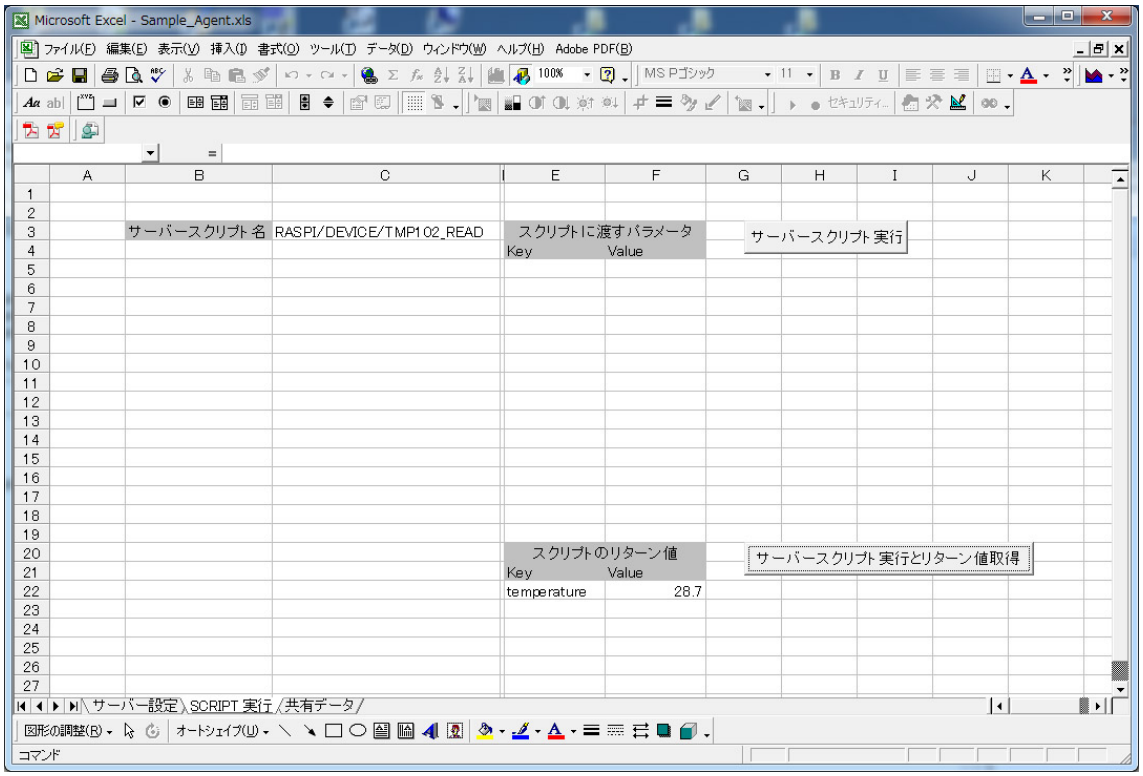
DLLライブラリは abs_agent とネットワーク接続された任意の数の Windows PC に配布して使用することができます。

ライブラリ(XASDLCMD.DLL) で提供される主な機能

- abs_agent 内のグローバル共有変数データの操作
- ユーザースクリプトの実行（任意のパラメータを実行時に指定することが可能）

ライブラリは再配布可能な DLLファイルで提供しています、様々な開発環境で使用することができます。サンプルとして、エクセル VBA でライブラリを利用したワークシートが用意されています。これには、VBA から DLL関数をコールするための全ての関数定義が記述されていますので、ユーザーがエクセルワークシートやVB でクライアントプログラムを作成する場合に応用が簡単にできます。

DLL ライブラリ関数の詳細は" API (XASDLCMD.DLL)"の章を参照してください。



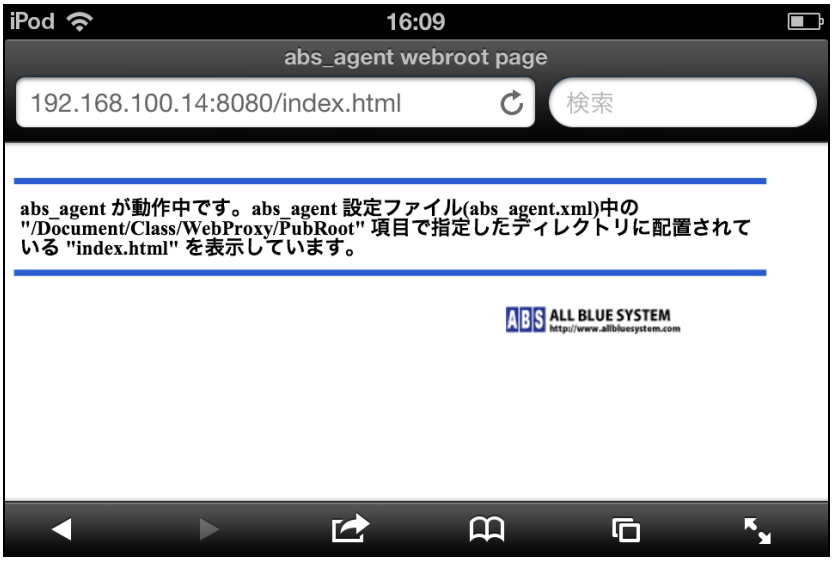
（リモートの Raspberry Pi で動作している abs_agent のスクリプトを Windows 上のエクセルから実行している様子。リモート側の I2C インターフェイスで接続された温度センサの値を読み込んでいます）

3.11 Web(HTTP)サーバー機能

abs_agent には Webサーバー機能が内蔵されていて、簡単に Web ページや Webアプリケーションの配信することができます。一般的な HTTP サーバーとして動作しますので、ユーザーが独自に作成した Webページを公開することができます。

abs_agent の Webサーバー機能は、主にイントラネット(LAN)からの Webアプリによるデバイス操作を想定して、必要最低限の機能のみインプリメントしています。このため公開サーバー(インターネット上の HTTPサーバー)で利用する場合にはセキュリティ等を十分に考慮してから設置してください。

インストール直後はデフォルト設定の TCP/IP ポート番号 (8080) で Webサーバーが動作しています。また、Web 上で公開するトップディレクトリは、abs_agent をインストールしたディレクトリ内の "webroot" ディレクトリに設定されています。これらの設定値は、サーバー設定ファイル(abs_agent.xml) を編集することで変更することができます。



(abs_agent をインストールしたコンピュータに Webブラウザからアクセスした様子。インストールキットで配置された デフォルトWebページが表示されています)

3.12 Web API機能

abs_agent 内蔵の Webサーバー機能を使用して、イントラネットやインターネット上に Web API を公開することが出来ます。ネットワーク接続されたクライアント(Webブラウザや Webアプリケーション等)から XMLHttpRequest 形式でアクセス可能であれば、abs_agent のユーザー認証やスクリプト実行、グローバル共有メモリやグローバル共有文字列リストへのアクセス機能を URL形式で行えます。

Web API の実行結果やスクリプト実行等のリターンパラメータを、JSON 文字列やコールバック呼び出し形式(JSONP)受け取ることが出来ます。これによって 他の Webアプリケーションや Node.js 等から簡単に abs_agent の機能を利用したシステムを構築することができます。

abs_agent の Web API 機能は、クロスドメイン通信を行うための “XMLHttpRequest Level 2” と “XdomainRequest” の処理に必要な “Access-Control-Allow-Origin” ヘッダを両方共サポートしています。また、JSONP 形式でのクロスドメイン通信もサポートしています。これによって、様々な Web ブラウザやアプリケーションサーバー、他のWebサービスから abs_agent の機能を呼び出してアプリケーションを構築することができます。

> ⚠ **URLの長さ制限**
> URL で指定する文字列は、エンコード後のサイズが 2000 バイトを超えないようにしてください。

Web API 機能で利用可能なコマンド一覧は下記になります。
Web API コマンドの詳しい使用方法については、"HTTPサーバー&Web API" の章を参照してください。

Web API コマンド	説明
/command/json/script	スクリプト実行とリターンパラメータ取得
/command/json/shared_data	グローバル共有変数の参照や更新

/command/json/shared_strlist	グローバル共有文字列リストの参照や更新
/command/json/session_login	ログイン認証を行ってセッションを作成
/command/json/session_logout	セッションを削除する
/command/json/session_update	セッションのタイムスタンプ更新やセッショントークン文字列の更新
/command/json/session_password	Web API にアクセスするユーザーアカウントのパスワード変更
/command/json/getmyip	Web API にアクセスするクライアント側の Peer IP アドレス取得
/command/log	abs_agent で設定されたログサーバーにメッセージ出力

3.13 WebSocketサーバー機能

abs_agent には WebSocketサーバー機能が内蔵されていて、Webブラウザで動作中のアプリケーションと双方向リアルタイム通信を行うことができます。abs_agent の WebSocketサーバーは前述の Web(HTTP)Server機能とは独立して動作して、WebSocket専用ポート番号を使用します。

abs_agent の WebSocket サーバーは標準プロトコル仕様(RFC6455) に基づいたインプリメントを行っています。このため、PC や スマートフォン上で動作する最新の Web ブラウザでサポートされている WebSocket インターフェイスを使用して簡単にアクセスすることができます。

abs_agent の WebSocketサーバーに接続するときは、ブラウザ等の JavaScript で標準サポートされている WebSocket()関数をコールします。その際、URLパラメータに指定するパス名(リソース名) 部分にチャンネル名 <channel> とセッショントークン文字列<SessionToken>を指定します。

JavaScript例: ws = new WebSocket("ws://<abs_agentホスト名またはIP>:9090/<channel>/<SessionToken>");

チャンネル名は任意の文字列を指定可能で、abs_agent 側からはチャンネル名を指定することで特定のクライアントグループにのみフレームを送信することができます。接続中の全クライアントに送信することも可能です。

セッショントークンは WebSocket サーバー側の接続認証に使用します。指定するセッショントークンは abs_agent の WebAPI機能を使用してログイン認証で取得したものや、スクリプトライブラリ関数で作成したものを指定します。

WebSocket クライアント(Webブラウザ等)からテキストやバイナリデータが送信されると、abs_agent 側で WEBSOCKET_DATA イベントハンドラが起動されます。このイベントハンドラ中にデータを処理するためのスクリプトを記述します。

abs_agent のイベントハンドラやユーザースクリプトからは何時でも、テキストやバイナリデータを Lua ライブラリ関数を使用して WebSocketクライアントに送信することができます。

クライアントプログラム agent_websocket を使用して、コマンドラインからWebSocket クライアントにデータを送信したり、現在接続中の WebSocket クライアント一覧を確認することもできます。

インストール直後はデフォルト設定の TCP/IP ポート番号 (9090) で WebSocketサーバーが動作しています。
この設定値は、サーバー設定ファイル(abs_agent.xml) を編集することで変更することができます。

3.14 汎用 TCP, UDPサーバー機能(ModbusTCP機能)

abs_agent には汎用の TCPサーバーと UDP サーバーが内蔵されています。

TCPサーバーでは abs_agent 設定ファイル中にフレーム構造(データ長)の定義を行うことで、任意のフォーマットのパケットを受信した後、受信データを処理するためのイベントハンドラ "TCPSERVER_DATA"が自動起動されます。デフォルト設定では ModbusTCPフォーマットのパケットを受信できる設定になっています。

UDPサーバーではクライアントから送信されたデータパケットを受信すると、イベントハンドラ "UDPSERVER_DATA"が起動され、この中に記述したスクリプトで受信データを処理できます。

TCPサーバーとUDPサーバー共、イベントハンドラ内でスクリプトのリターン値を設定することで、フレーム送信元にリプライデータを簡単に送信することができます。

汎用 TCP,UDPサーバーのセットアップ方法については、"サーバープログラム・設定ファイル"章中の "NETSERVER" の項を参照して下さい。また、サーバーでフレームを受信した時に実行されるイベントについては "イベント"章中の "TCPSERVER_DATA" と "UDPSERVER_DATA" の項を参照して下さい。

abs_agent では他のコンピュータやサーバーで動作している TCP, UDP サーバーへの通信機能も提供しています。
TCP,UDP通信用ライブラリ関数については "TCP,UDPクライアントAPI"章を参照して下さい。

3.15 グラフィック描画機能

abs_agent にはモノクロやカラーで図形や文字列を描画するグラフィック描画機能があります。描画は、abs_agent 内に作成するメモリエリア(ディスプレイ・バッファ)に描画を行います。

ディスプレイバッファは abs_agent のグローバル共有メモリエリア上に作成していますので、ディスプレイバッファの内容を描画したいデバイスに転送して表示します。Raspberry Pi 用の abs_agent では I2C や SPI 経由でディスプレイバッファの内容を LCD デバイスに直接転送するライブラリ関数が提供されていますので、アプリケーションにLCD表示機能を簡単に追加できます。

インストールキットで提供されていない LCD デバイスへの表示も可能です。ディスプレイバッファの描画までは共通なので、それ以降のデバイスへの転送部分のみをユーザーがスクリプトで記述することで実現します。ユーザーが作成する必要があるのは、デバイスの初期化とディスプレイバッファ転送部分のみです。これらのスクリプトも既存のドライバ(スクリプト)ソースをコピーすることで簡単に移植できます。

グラフィック描画機能の詳細については "カラー(RGB)・モノクロ(Bitmap)グラフィックAPI"の章を参照して下さい。
また、"APPENDIX(B)"章に Raspberry Pi に接続した各種LCDデバイスに集計グラフを表示するアプリケーションを説

明しています。

4 abs_agentインストール

abs_agent のインストールは、Linux ボードにインストールする abs_agent インストールキットと、ログサーバー機能を Windows PC にインストールするための ABS-9000 LogServer の２つのキットを使用します。ログサーバーの設置は必須ではありませんが、運用中の abs_agent を外部から監視できるようになります。abs_agent 設置・運用中のトラブルやエラー発生に備えて、先に次の章の "ログサーバー・インストール"の項を参照して予めログサーバーを設置しておくことをお勧めします。

ログサーバーをインストールしなくても abs_agent を動作させることができます。この場合には、abs_agent をフォアグランドで動作させることで、全てのログメッセージをコンソールに出力させることも可能です。詳しくは "サーバープログラム" の章をご覧ください。

abs_agent のインストールキットはハードウエアタイプ毎に分かれていますので、間違えないように選択してください。現在、インテル x86 の Debian GNU/Linux 8 用(X86)と、Raspberry Pi ver1,2,3,4 の Raspbian用(RASPI) の２つのタイプがあります。このマニュアルでは Raspberry Pi ver3 にインストールする手順について記述しています。その他のハードウエアタイプについても手順は全く同じです。

4.1 abs_agent動作環境

abs_agent は下記の環境で動作します。

(1) Raspberry Pi 用 (Hardware Type: RASPI)

項目	必要なバージョン・リソース
ハードウエア	Raspberry Pi ver1,2,3,4,ZeroW,Zero2W
OS	各ハードウエアに対応した、最新バージョンの Raspbian OS
ディスク容量	15Mbytes 以上の空き容量
ネットーワークポート	10Mbps/100Mbps イーサネットポートまたは Wi-Fi ネットワークインターフェイス

(2) Intel x86用 (Hardware Type: X86)

項目	必要なバージョン・リソース
ハードウエア	Intel x86 プロセッサ CPU 450MHz以上、メインメモリ 128MBytes以上
OS	Debian GNU/Linux 8.2 (i386) 以降のバージョン
ディスク容量	15Mbytes 以上の空き容量
ネットーワークポート	10Mbps/100Mbps イーサネットポートまたは Wi-Fi ネットワークインターフェイス

オールブルーシステムでは、上記以外のお客様のハードウエア環境に合わせたビルドを用意することができます。基本的には、Debian GNU/Linux 系のネイティブ開発環境が構築できれば abs_agent を移植可能です。また、お客様のハードウエア特有の機能にアクセスするための Lua ライブラリ関数を用意することも出来ます。プロセッサの詳細な資料があれば、メモリマップしたレジスタアクセスで高速なハードウエア操作も可能です。詳しくはメールでお問い合わせ下さい。(Email: contact@allbluesystem.com)

4.2 abs_agent で使用するネットワークポート

abs_agent ではクライアントプログラムとサーバー間の通信やログ出力用にネットワークポートを使用します。詳しい仕様は以下になります。

abs_agent プログラムが使用するプロトコルとポート番号		
プログラム・機能	**プロトコル**	**ポート番号**
abs_agent サーバー間のリモートアクセス abs_agentクライアントから abs_agent にアクセス XASDLCMD.DLL 経由で abs_agent にアクセスする Win32 アプリケーション	TCP	27101
abs_agent の HTTP サーバー機能と、Web API で公開するサービス	TCP	8080 (変更可能)
abs_agent の WebSocket サーバー機能	TCP	9090 (変更可能)
ログサーバーへのメッセージ送信 ログサーバー・ログコンソール間の通信	UDP UDP	2056 2057
MQTTクライアント・エンドポイント(ユーザー設定時のみ)	TCP	1883 (変更可能)
汎用TCPサーバー機能・ModbusTCP等(ユーザー設定時のみ)	TCP	502 (変更可能)
汎用UDPサーバー機能(ユーザー設定時のみ)	UDP	8090 (変更可能)

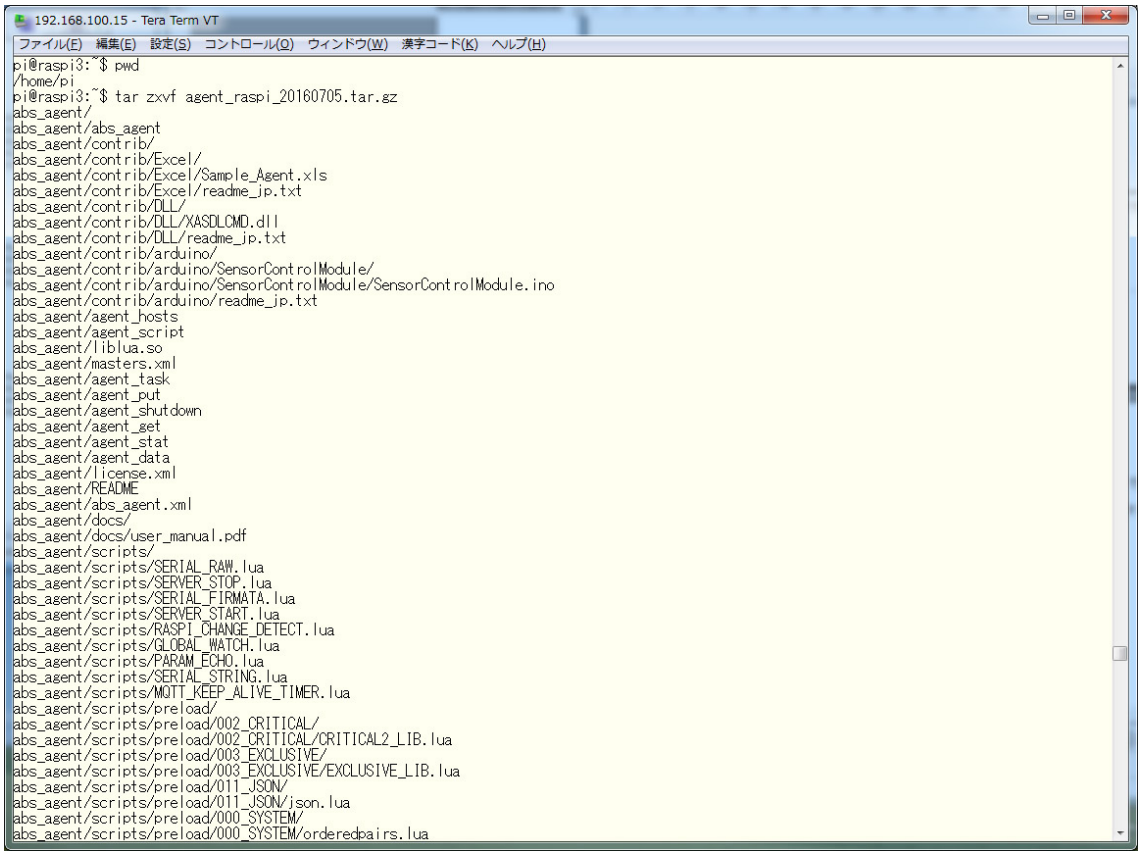
4.3 abs_agent インストール

abs_agent のインストールはインストールキットのアーカイブファイル(tar + gzip形式)を tar コマンドで展開するだけで完了します。

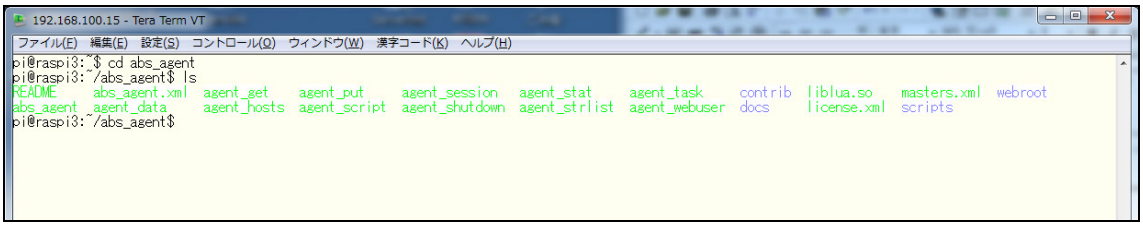
下記のコマンドでインストールできます。

```
tar zxvf <インストールキットファイル名>
```

abs_agent プログラムは任意のディレクトリに配置することができます。下記の例では Raspberry Pi で動作している Raspbian OS にユーザー名 "pi" でログインして、ホームディレクトリ(/home/pi) にインストールする例です。



キットを展開すると、abs_agent ディレクトリが作成されてその中に abs_agent プログラムで必要な全てのファイルが格納されています。



abs_agent がサーバープログラムで agent_xxxxx の名前が付いているプログラムがクライアントプログラムです。

> ⚠ **abs_agent プログラムと agent_xxxxx クライアントプログラムは同一パスに配置してください**
>
> インストールキットで設置されたサーバープログラムやクライアントプログラムは、任意のディレクトリに移動させても構いません。ただし、スクリプト中から agent_xxxxx クライアントプログラムをコールする場合のために、abs_agent サーバープログラムと同じディレクトリに agent_xxxxx クライアントプログラムを配置するようにしてください。このようにすることで、PATH 環境変数等に依存しないでスクリプト中からクライアントプログラムをコールできます。
>
> 参考：service_module_status() API 関数、BACKUP_RESTORE スクリプト

4.4 ハードウエア・タイプ設定 (Raspberry Piのみ)

Raspberry Pi 用の abs_agent では、プロセッサ内部のレジスタに直接アクセスして GPIO や I2C, SPI デバイスを操作できます。この時 Raspberry Pi ハードウエア・バージョン毎にレジスタアクセスの方法を変える必要があるため、予め使用する Raspberry Pi のハードウエア・タイプを abs_agent の設定ファイル(abs_agent.xml)に書き込んでおきます。

インストールキットで展開したディレクトリ内にある、abs_agent コンフィギュレーションファイル(abs_agent.xml)を viエディタ等で開いて XML タグの内容を下記の様に書き込みます。<RASPI> .. </RASPI> で囲まれた下記の部分を以下の様に書き換えて下さい。

コンフィギュレーションファイル(abs_agent.xml) 中の <RASPI> .. </RASPI>タグの内容。(Raspberry Pi ver3 の場合の例)

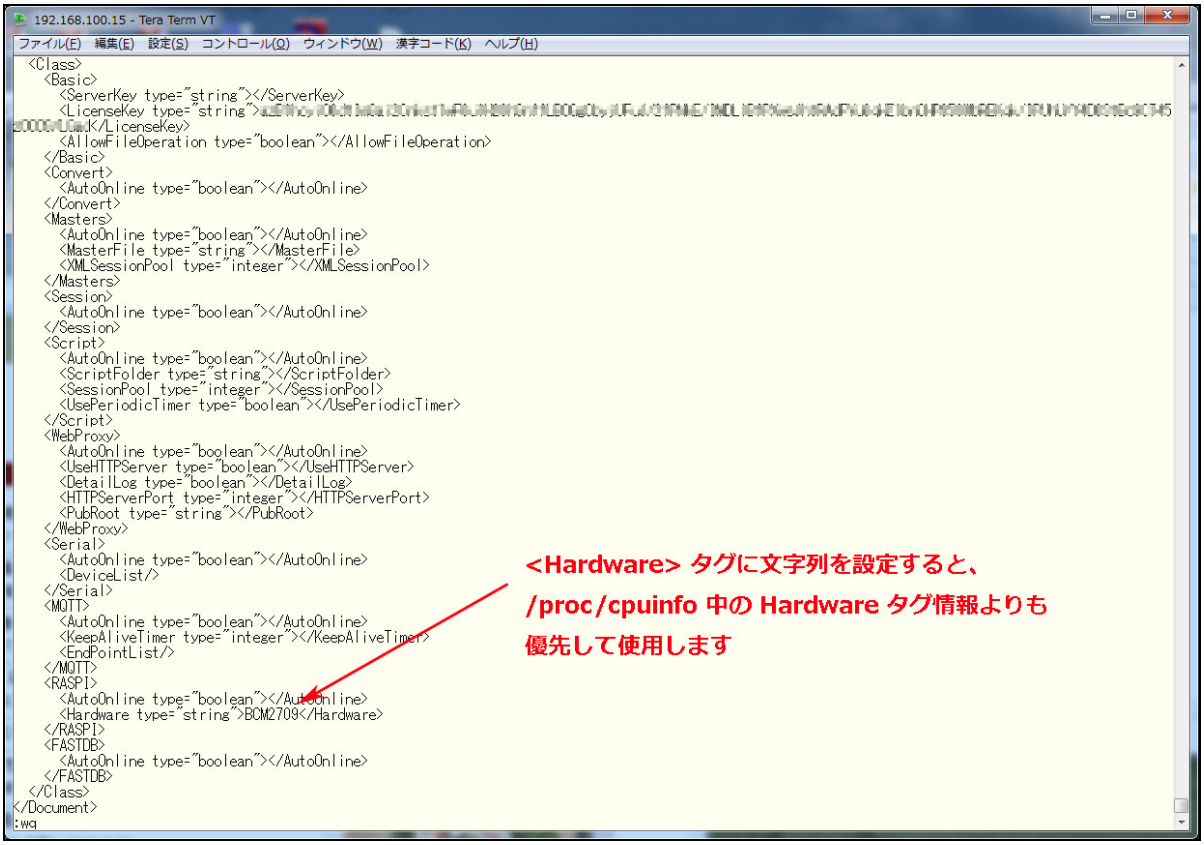
```
<RASPI>
  <AutoOnline type="boolean"></AutoOnline>
  <Hardware type="string">BCM2709</Hardware>
</RASPI>
```

<Hardware> .. </Hardware> タグ部分に SoC タイプを表す文字列を挿入します。インストール直後は空白になっていますので、Raspberry Pi バージョンに合わせて下記の文字列を書き込んでください。

<Hardware>タグに書き込む文字列	使用する Raspberry Pi のバージョン
BCM2708	Raspberry Pi Model B+ Raspberry Pi Zero, Zero W (BCM 2835 ARMv11 Single Core 32bit)
BCM2709	Raspberry Pi 2 Model B (BCM 2836 ARMv7 Cortex A7 Quad-Core 32bit)
BCM2709	Raspberry Pi 3 Model B (BCM 2837 ARMv8 Cortex A53 Quad-Core 64bit)
BCM2710	Raspberry Pi Zero 2 W (BCM 2710A1 ARM Cortex A53 Quad-Core 64bit)
BCM2711	Raspberry Pi 4 Model B (BCM 2711 ARMv8 Cortex A72 Quad-Core 64bit)

(*) "BCM2709" と "BCM2710" は abs_agent 内の動作は全く同じなのでどちらを記述しても構いません

下記は、vi エディタで編集しているときの様子です。編集後はファイルを更新(上書き)してエディタを終了します。

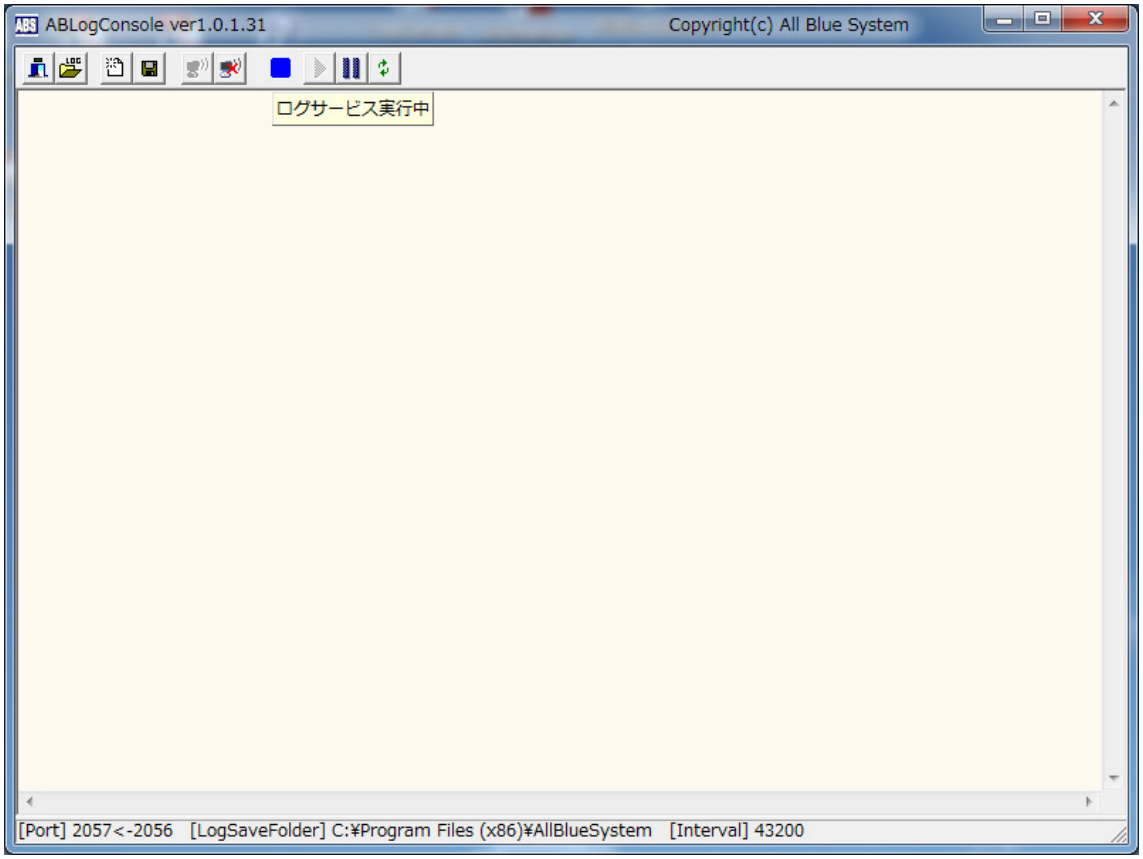


4.5 abs_agent 手動起動

最初に abs_agent を手動で起動してみます。実際に運用する場合には、OS が起動すると同時に abs_agent を自動起動させる方が便利なのですが、その方法については後述します。

abs_agent 起動メッセージをログサーバーで確認するためには、Windows PC にインストールしたログサーバーが必要です。ログサーバーのインストール方法については次の章で説明していますので参照して下さい。

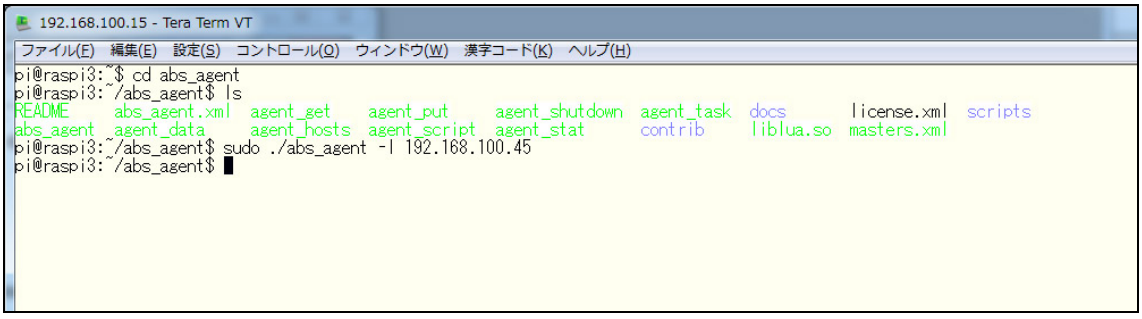
ログサーバーを設置した Windows PC のプログラムメニューから "All Blue System" -> "ログコンソール" を選択してログコンソール画面を表示させます。このとき、画面上のツールバーに表示されているサーバーステータス(四角いランプ)が青色で "ログサービス実行中" になっていることを確認します。
尚、この例では ログサーバーを設置した Windows PC の IP アドレスは 192.168.100.45 になっています。



次に、Raspberry Pi に abs_agent をインストールしたディレクトリで abs_agent を手動起動します。起動コマンドは下記になります。"-l"（エル）オプションでログサーバーを設置した Windows PC の IP アドレスを指定しています。また、Raspberry Pi向けにビルドした abs_agent では Raspberry Pi のハードウエア機能(GPIO,SPI,I2C) にメモリマップ経由でダイレクトアクセスしています。このため、実行には root 権限が必要になりますので "sudo" コマンドを併用しています。詳しい abs_agent プログラムの使い方については "サーバープログラム" の章を参照して下さい。

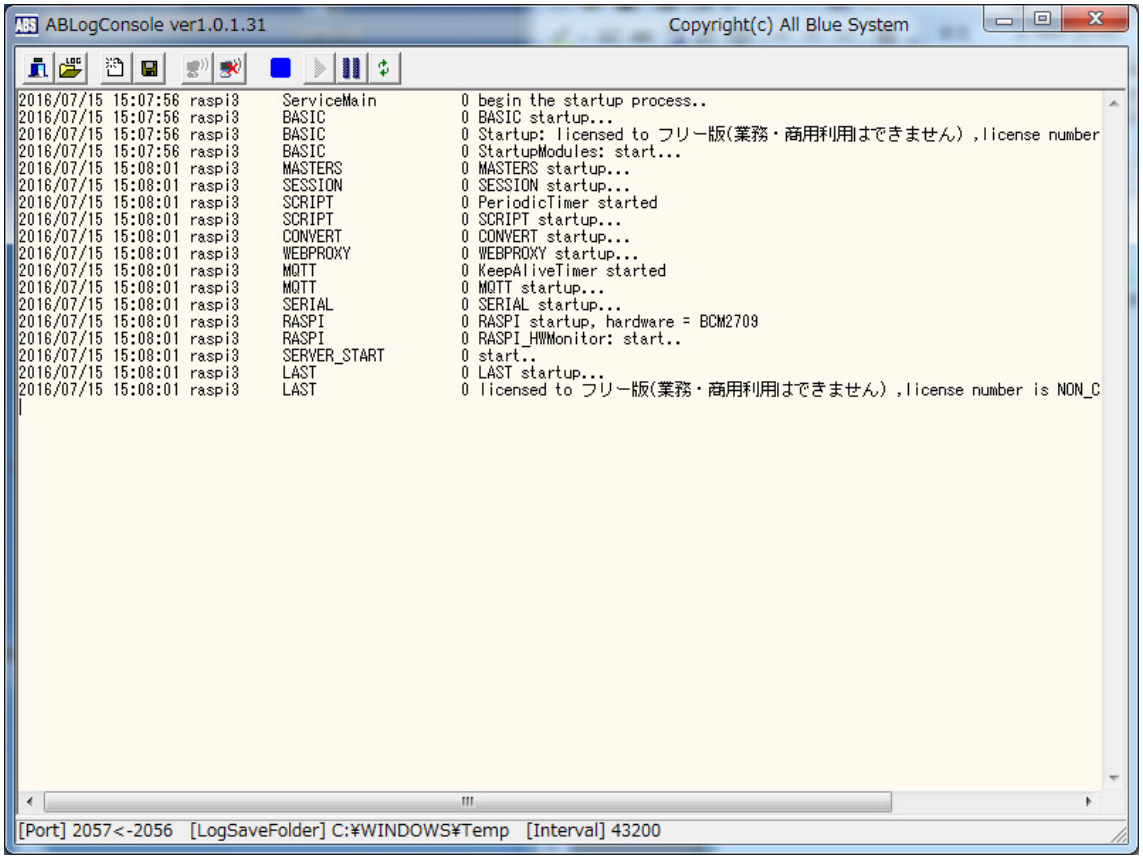
```
sudo ./abs_agent -l <log_server_ip_address>
```

実際に起動している様子は下記になります。



abs_agent プログラムは直ぐにプロンプトを戻して、別プロセスのデーモンとしてシステムに常駐します。この後、コンソールからログオフしても abs_agent プログラムは起動したままになります。

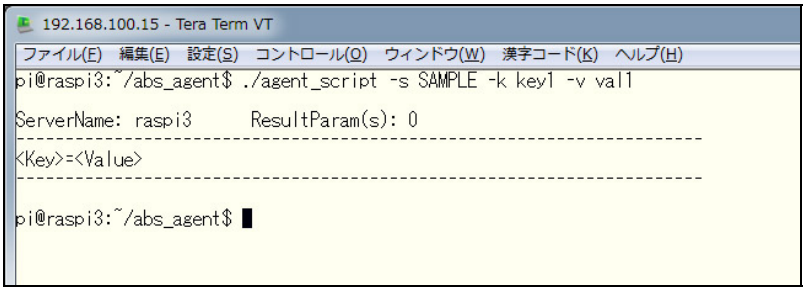
abs_agent を起動すると、ログコンソール画面には起動メッセージが下記の様に表示されます。



ここで、インストールされているサンプルスクリプトを実行してみます。スクリプト名は "SAMPLE" で、実行時のリクエストパラメータをログに出力するだけの動作を行います。スクリプト実行には agent_script クライアントコマンドを使用します。"-s" オプションでスクリプト名を指定して、"-k <key> -v <val>" オプションで実行時のリクエストパラメターをキー名と値のペアで与えています。

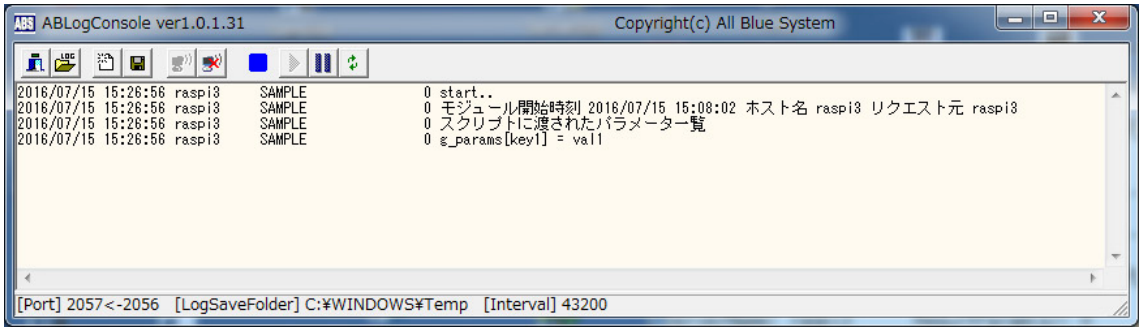
```
./agent_script -s SAMPLE -k key1 -v val1
```

コマンド実行例は下記になります。詳しい agent_script コマンドの仕様については、"クライアントプログラム" の章を参照してください。



スクリプトの実行が完了すると、実行したスクリプト中でリターンパラメータを設定していた場合には画面に表示さ

れます。この例ではリターンパラメータはありません。また、ログコンソール画面には下記の様なメッセージが表示されます。



4.6 abs_agent 動作環境を別ディレクトリに作成

abs_agent はインストール直後には初期設定状態で起動しています。abs_agent からシリアルデバイスに接続した計測器やI/O 装置を操作したり、MQTT ブローカとの通信を行うためには abs_agent の設定ファイルに詳細設定を記述することで使用可能になります。

デフォルトではインストールしたディレクトリ内にある abs_agent.xml ファイルにこれらの詳細設定が書き込まれています。このサーバー設定ファイルは XML 形式のファイルで、何時でもテキストエディタで編集することができます。サーバー設定ファイルの内容を変更した場合には、有効にするために abs_agent を再起動させます。abs_agent のサーバー設定ファイルについての詳しい内容は "サーバープログラム" の章をご覧ください。

abs_agent を起動するときには、起動オプション "-c <config_file>" で任意のディレクトリに配置したサーバー設定ファイルを指定することもできます。サーバー設定ファイル中には abs_agent で利用するスクリプトファイルのトップディレクトリやマスターファイル名など、abs_agent が動作中に参照するファイルやディレクトリのパス情報が記載されています。この設定ファイルの記述を変更するかもしくは、別の設定ファイルを新規に作成することでユーザーが abs_agent の動作環境をインストールディレクトリ以外にも複数配置することができます。

たとえば、abs_agent をインストールしたディレクトリが "/home/pi/abs_agent" として、これとは別ディレクトリ "/home/pi/my_config" に動作環境を構築してみます。

クライアントプログラム agent_copycfg の使用

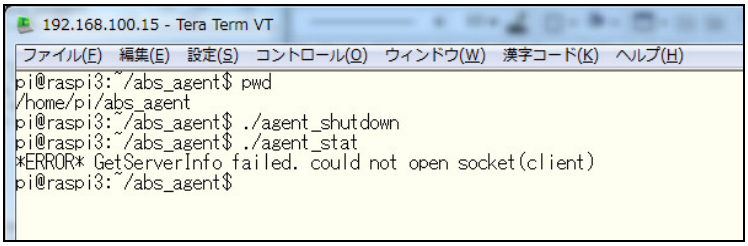
この項で説明するのと同じ作業内容を自動で実行するためのコマンド agent_copycfg が abs_agent インストールキットに同梱されています。以下がコマンド実行例になります。詳しいコマンドの説明は "クライアントプログラムの章" 中の "agent_copycfg" の項を参照してください。

```
(1) cd                                          # home ディレクトリに移動
(2) mkdir my_config                             # コピー先実行環境ディレクトリ作成
(3) cd abs_agent                                # abs_agent インストールディレクトリに移動
(4) ./agent_copycfg -v -w -c -s . -d ../my_config # コピー先実行環境ディレクトリに全ファイルコピー
```

最初に、abs_agent が既に動作中の場合にはこれを停止させます。クライアントコマンド agent_shutdown を実行してサーバーを停止します。

```
./agent_shutdown
```

下記は実行したときの様子です

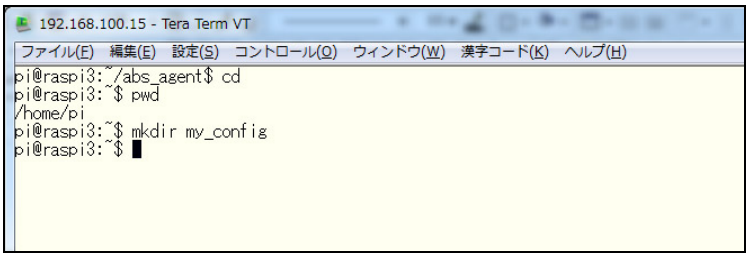


agent_shutdown 実行後に abs_agent サーバープログラムが完全に停止するまで 10 秒程度掛かります。上記の例ではその後、agent_stat コマンドを実行してサーバーから応答が無いことを確認しています。ログコンソールを表示している場合には、"service monitor stopped" のメッセージが表示されたときに、完全に abs_agent が停止していると判断できます。

次に、新しい動作環境用のディレクトリを作成します。任意のディレクトリ名で作成できます。

```
mkdir my_config
```

下記は実行したときの様子です



作成した "my_config" ディレクトリに、abs_agent をインストールしたディレクトリから必要なファイルをコピーします。新しい動作環境にコピーするファイルは以下になります。Linux "cp" コマンドを使用してコピーしてください

(1) abs_agent サーバー設定ファイル (abs_agent.xml)

(2) マスターファイル (masters.xml)

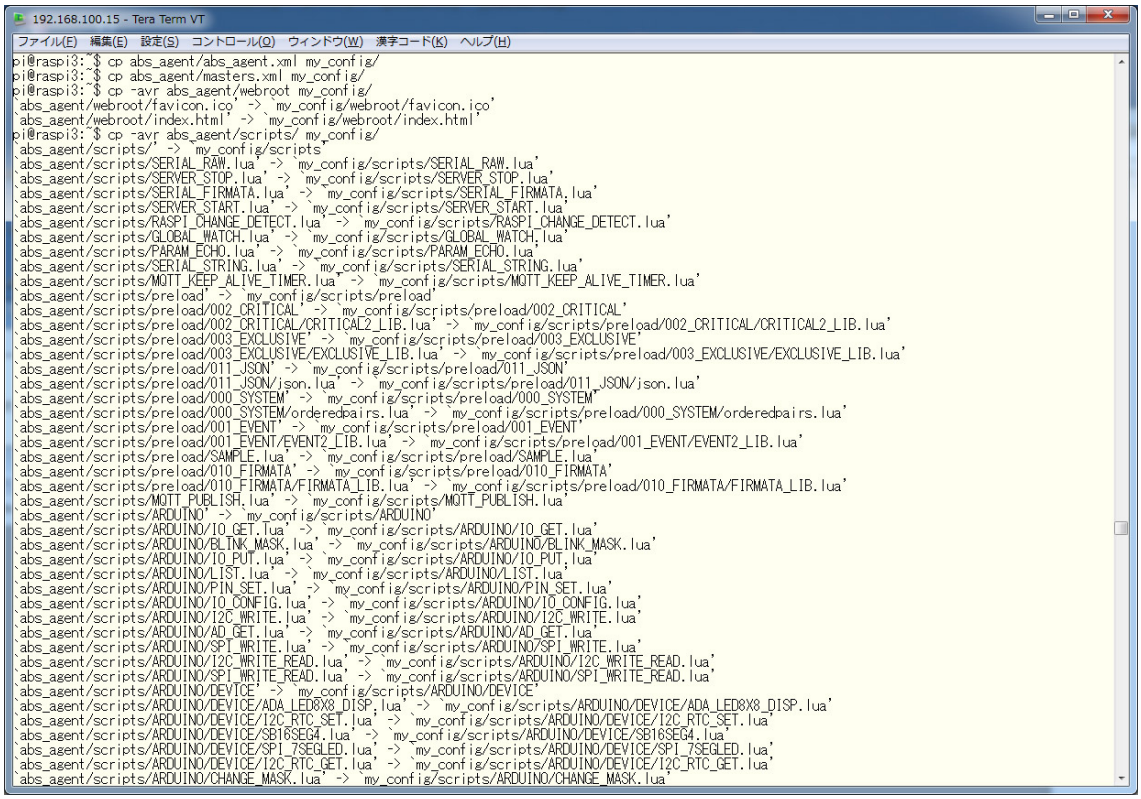
(3) HTTPサーバー公開用に設置した、HTML,CSS,JavaScript等のファイル

(4) スクリプトファイル（プリロードライブラリ、イベントハンドラ、ユーザースクリプト）

```
cp abs_agent/abs_agent.xml my_config/

cp abs_agent/masters.xml my_config/

cp -avr abs_agent/webroot my_config/
```

```
cp -avr abs_agent/scripts my_config/
```

HTTPサーバー用のファイルとスクリプトファイルをコピーする場合には、webroot と scripts ディレクトリ以下にあるサブディレクトリを含む全てのファイルをコピーするために "cp -avr " オプションを指定している点に注意していください。下記は実行したときの様子です



（インストールディレクトリで使用していた動作環境を、my_config ディレクトリにコピーしている様子）

⚠ サーバー設定ファイル(abs_agent.xml) のデフォルト値は初回の abs_agent 起動時に書き込まれます

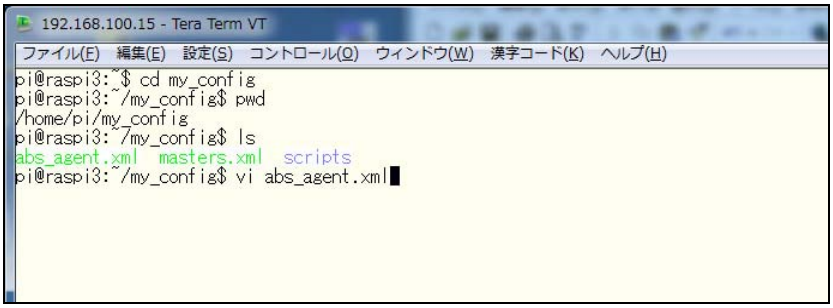
abs_agent サーバー設定ファイルには、abs_agent プログラムを最初に起動したときにデフォルト値が書き込まれます。このため、インストール直後には設定内容が空になっている項目があります。この場合には、一度 abs_agent を起動してから編集する様にすると作業がやり易くなります。

これで、新しい動作環境に必要なファイルの準備が出来ました。次に、my_config ディレクトリにコピーした abs_agent サーバー設定ファイル(abs_agent.xml)をエディタで開いて、マスターファイル名のパス名とスクリプトファイルのトップディレクトリ名、HTTPサーバーの公開ディレクトリ名を my_config ディレクトリに合わせます。

ℹ コンフィギュレーションファイル中のパス名に $CONFIGDIR$ を使用する

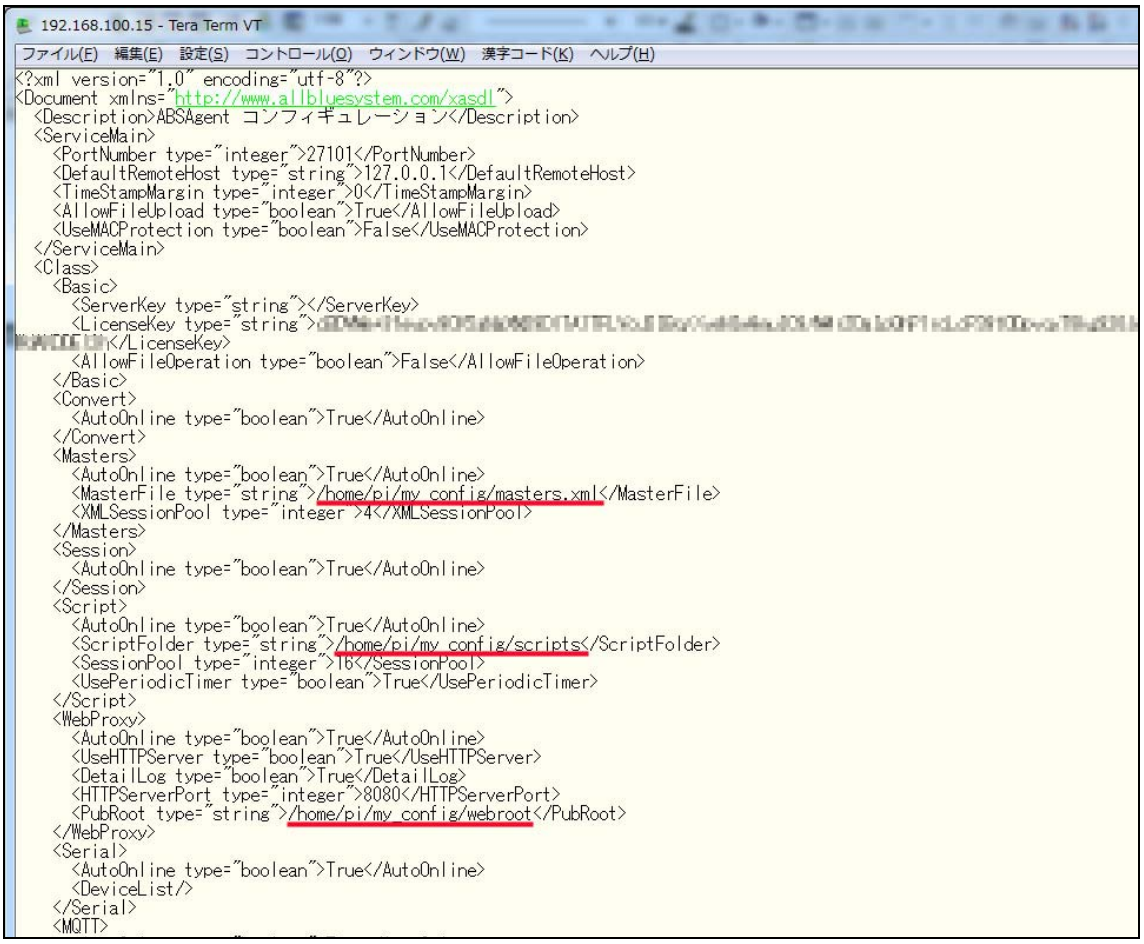
もし上記の作業と同様に、新しく作成した動作環境のコンフィギュレーションファイルと同じディレクトリにマスターファイルや、スクリプトディレクトリ、HTTP 公開ディレクトリを配置した場合には下記のパス名書き換えは不要です。デフォルト・コンフィギュレーションファイル中に記載された "$CONFIGDIR$" 部分がコンフィギュレーショ

ンファイルを配置したディレクトリ名に置き換えられて解釈されるためです。



エディタを使用して、XML タグの下記のパス名部分を "my_config" ディレクトリ以下になるように書き換えて下さい。

変更後は下記の様になります。

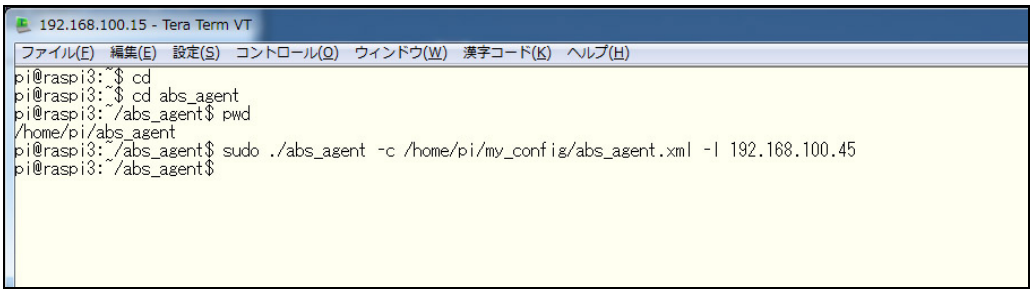


これでサーバー設定ファイルの変更は完了しました。ファイルを更新してエディタを終了します。使用するエディタによっては書き込み時に、編集前と違う文字コードを設定できる場合がありますので必ず編集前と同じ UTF-8(BOM無し）で保存してください。

これで、新しい動作環境で abs_agent を起動することができます。abs_agent 起動時には "-c <config_file>" を指定します。このとき、<config_file> で指定するファイルパス名は必ず絶対パス名を指定するようにしてください。下記のコマンド実行例では "-l <log_server>" オプションで、ログサーバーを設置した Windows PC の IP アドレスも指定しています。

```
sudo ./abs_agent -c /home/pi/my_config/abs_agent.xml -l 192.168.100.45
```

下記は実行したときの様子です



新しい動作環境に配置したサーバー設定ファイル、マスターファイル、スクリプトファイルを使用して abs_agent が動作します。クライアントコマンド(agent_script) でスクリプトを実行する場合や、スクリプトファイルの更新 (agent_get, agent_put) を行う場合にも、これらの新しい動作環境に配置されたスクリプトを参照します。

abs_agent の動作環境を任意のディレクトリに作成するこれらの方法は、abs_agent インストールキットを更新する場合や、複数の動作環境を使い分けたい場合に便利です。

例えば、my_config ディレクトリにコンフィギュレーションファイルやスクリプトファイル、Webファイルを配置した今回の例では、新しい abs_agent にアップデートする場合には下記の様な手順になります。

(1) 最初に abs_agent ディレクトリに移動した後、実行中の全スクリプトを強制終了した後 abs_agent をシャットダウンします

```
cd abs_agent
./agent_task -K
./agent_shutdown
```

(2) 既存の abs_agent インストールディレクトリ(/home/pi/abs_agent)の親ディレクトリ(/home/pi) に移動した後、abs_agent をディレクトリごと削除します。

```
cd ..
rm -rf abs_agent
```

(3) 新しいインストールキットファイルをカレントディレクトリ(/home/pi)に tar コマンドで展開します。

```
tar zxvf agent_raspi_20181226.tar.gz
```

(4) 必要に応じて新しいキットに格納されているイベントハンドラ等を my_config ディレクトリ以下の対応する部分にコピーするか、既存のファイルを修正します。例えば agent_copycfg コマンド用にカストマイズ部分が _app_<script_filename>.lua や _cpy_<script_filename>.lua に保存されている場合には下記のコマンドを実行します。

```
cd abs_agent
./agent_copycfg -t -s . -d ../my_config
# -t オプションによって、最新版のキット中のスクリプトと既存のスクリプト間に差異があるファイル名が
# 出力されます（実際のファイル更新はまだ行われない）
# _cpy_xxxx.lua や _app_xxxx.lua ファイルを適切に準備した後に下記のコマンドを実行して実際のファイル
# 更新を行います
./agent_copycfg -s . -d ../my_config
```

(5) abs_agent サーバーを起動します。このとき my_config ディレクトリ内のコンフィギュレーションファイルを -c <config_file> オプションで忘れずに指定します。

4.7 abs_agent 自動起動設定

コンピュータの電源を入れたときに、abs_agent サーバーを自動的に起動する様に設定できます。

OS (Debian GNU/Linuxや Raspbian) の起動シーケンスの最後に、ユーザー環境に合わせた初期化のための /etc/rc.local シェルスクリプトが実行される機能が組み込まれています。この /etc/rc.local シェルスクリプト中に下記の様な abs_agent を起動するコマンドを追加します。

```
LANG=ja_JP.UTF-8
export LANG
/home/pi/abs_agent/abs_agent -l 192.168.100.45
```

LANG 環境変数を "ja_JP.UTF-8" に設定しています。これは abs_agent 内部で日本語を含む文字列を処理する場合に必要となりますので、忘れずにシェルスクリプト中に記述して下さい。abs_agent を手動で起動するときと同様にログサーバーを指定する "-l" パラメータも指定しています。abs_agent プログラムはインストールしたディレクトリの絶対パス名で指定します。/etc/rc.local は OS のシステム初期化プロセスによって実行されますので、"sudo" コマンドで root 特権を明示的に指定する必要はありません。

OS のバージョンによっては /etc/rc.local 機能が無効になっている場合があります。この場合は動作環境に合わせて上記の動作を行う起動スクリプトを作成するか、もしくは /etc/rc.local 機能を有効にします。

OS起動直後に abs_agent を起動すると、CPU負荷が高い場合に下記のエラーが連続して発生する場合があります。

```
LUASessionPool       0 SelectFreeLUA:*WARNING* could not find the free index.
LUASessionPool       0 SelectFreeLUA:*WARNING* could not find the free index.
```

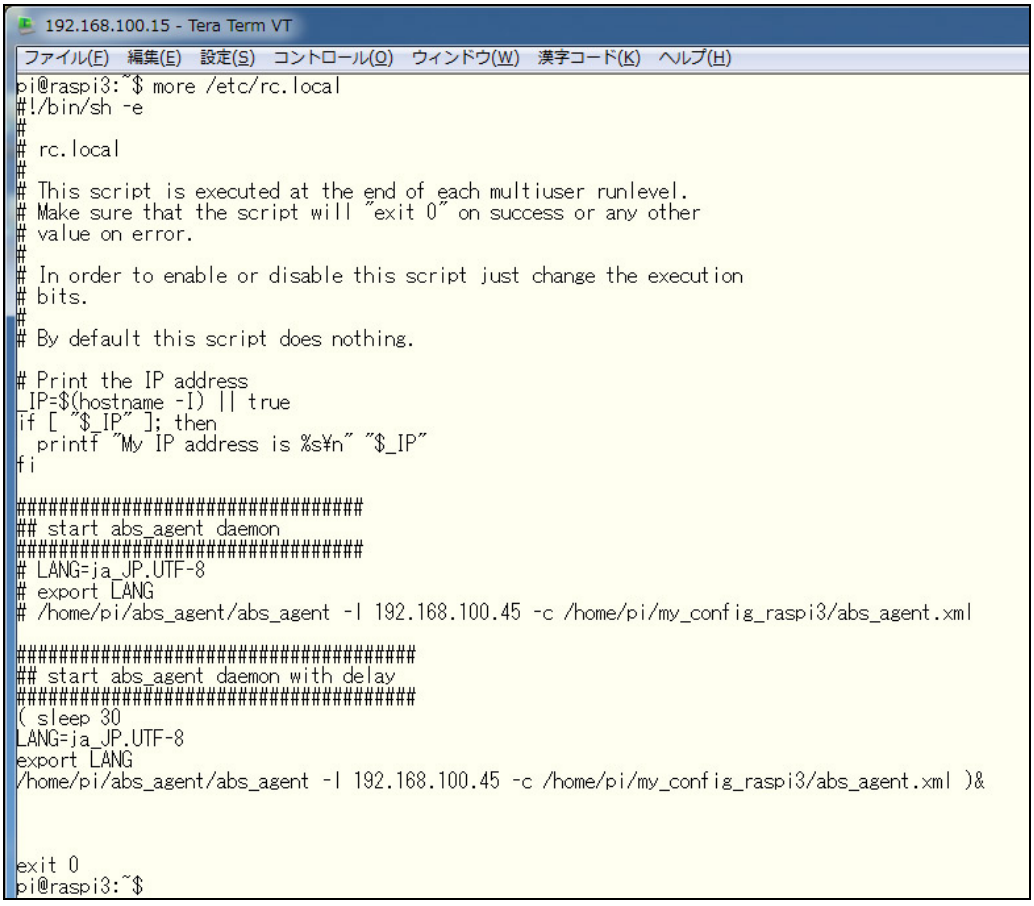
. .

このエラーが発生した場合は、OS 起動後 60秒程度経過してから abs_agent を起動するようにします。下記の例では OS起動後、周辺機器用のサービスや他のサーバープロセスの起動を待つために、60 秒経過してから abs_agent を起動しています。sleep コマンドと abs_agent 起動コマンドを括弧で囲んで全体をバックグランド("&" を指定)で実行します。

```
(sleep 60
LANG=ja_JP.UTF-8
export LANG
/home/pi/abs_agent/abs_agent -c /home/pi/my_config/abs_agent.xml -I 192.168.100.45 )&
```

この設定例では "動作環境を別ディレクトリに作成" の様に、abs_agent をインストールしたディレクトリ "/home/pi/abs_agent" とは別ディレクトリ "/home/pi/my_config" に、サーバー設定ファイルやスクリプトを格納しています。

実際の設定例は以下になります。



OS シャットダウン時の、agent_shutdown コマンド実行は省略できます

OS のシャットダウン時には、abs_agent を停止させる為に agent_shutdown コマンドを使用して abs_agent プロセスを停止させる方が望ましいのですが、この操作は省略することができます。abs_agent ではユーザーが作成したス

クリプト中からの明示的なファイル操作(マスターファイル書き込み、グローバル共有メモリエリアのファイル出力、Lua 標準 io ライブラリでの書き込み等)を実行している場合を除いて、ファイルシステムへの書き込みを行っていません。このため、OS から abs_agent サーバープロセスを強制的に停止させても特に問題はありません。

4.8 ライセンス入力

abs_agent インストールキットにはフリー版のライセンスコードが同梱されています。個人目的で使用される場合には、そのままライセンス入力操作をされなくても動作期間の制限無く御使用になることができます。

フリー版ライセンスでは、業務・商用目的での使用やお客様自身の製品やサービスへの組み込みはできません。また、接続シリアルデバイス数や MQTT QoS, エンドポイント数、リモートクライアント数などに制限があります。詳しくはインストールディレクトリ中に保存されているライセンスファイル(license.xml)の内容をエディタ等でご覧下さい。オールブルーシステムからスタンダードライセンスを購入されると、利用目的制限や機能制限が無いスタンダードライセンスを入手することができます。また、OEM 向けに発行された abs_agent のライセンスコードも御使用になることが出来ます。

ライセンスは電子メールで手軽に購入することができます。フリー版ライセンスの abs_agent が動作している環境で、クライアントプログラム agent_stat コマンドの出力結果を "agent_stat >> my_order_file" の様にファイルにコピーして、それを添付して contact@allbluesystem.com 宛てにメール頂ければ、折り返しお見積りと振り込み案内をお送りします。また複数のライセンスを同時にオーダーしていただく事もできます。この場合には動作させるコンピュータでそれぞれ abs_agent をフリー版ライセンスを動作させていただいた後、上記と同様に agent_stat コマンドの出力をまとめて１つのファイルにしてお送り下さい。価格や詳しい内容についてはホームページ(http://www.allbluesystem.com/)をご覧になるかまたは、電子メール(Email: contact@allbluesystem.com)でお気軽にお問い合わせ下さい。

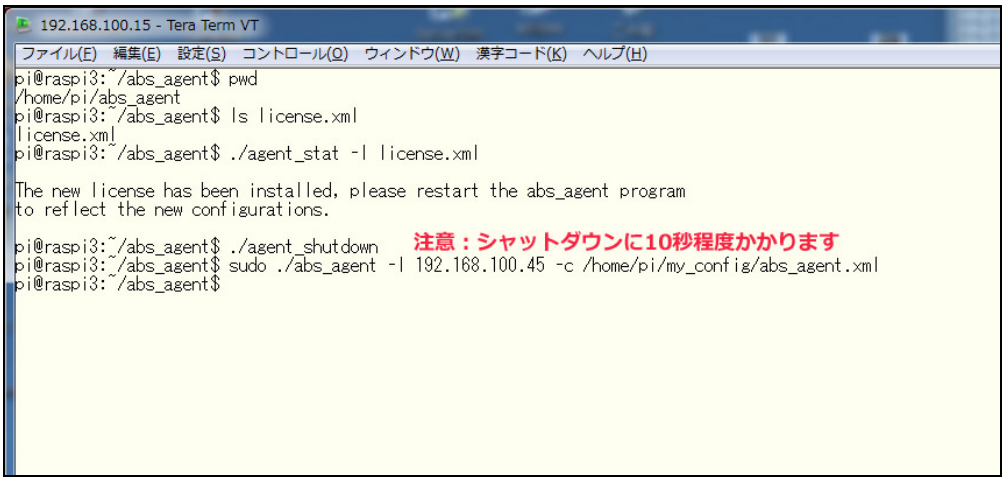
以下は、上記の方法でフリー版以外のライセンスコードを入手された場合のインストール方法について説明します。

ライセンスコードは、abs_agent サーバー設定ファイル(abs_agent.xml)中の設定項目に入力します。クライアントプログラム "agent_stat -l <license_file>" コマンドを使用すると簡単に設定できます。

入手されたライセンスファイルを、abs_agent が動作しているコンピュータにコピーしてください。例えばライセンスファイル名が "license.xml" で abs_agent をインストールしたディレクトリ "/home/pi/abs_agent" にコピーした場合には、ライセンスコードをインストールするコマンドは下記の様になります。

```
./agent_stat -l license.xml
```

下記は実行したときの様子です

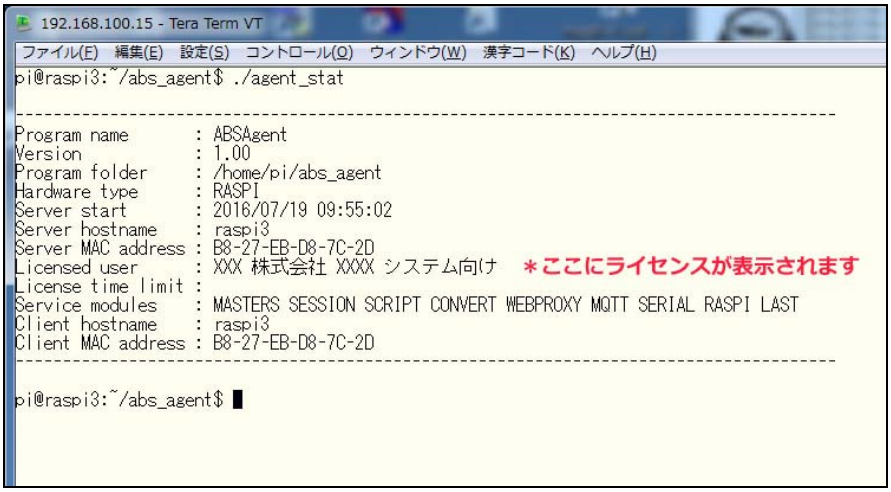


"agent_stat -l <license_file>" コマンド入力後は abs_agent を再起動させる必要があります。ここではクライアントプログラム "agent_shutdown" コマンドでサーバーを停止させた後、手動で abs_agent を起動しています。

```
./agent_shutdown

sudo ./abs_agent -l 192.168.100.45 -c /home/pi/my_config/abs_agent.xml
```

設定後の新しいライセンス情報は、ライセンスインストール時に使用したのと同じ "agent_stat" プログラムを使用します。"-l <license_file>" オプション無しで実行すると、現在の abs_agent 動作ステータスを表示します。

```
./agent_stat
```



4.9 スクリプト編集用エディタを用意する

abs_agent で動作するイベントハンドラやスクリプトはテキストエディタで簡単に作成できます。

Linux 標準の vim エディタ等で簡単にスクリプトの作成や修正ができます。スクリプト中に日本語を使用することも出来ますが、エディタソフトが日本語編集をサポートしている必要があります。

abs_agent クライアントプログラム(agent_script, agent_get, agent_put) を使用すると、Lua スクリプトファイルが実際に格納されているファイルシステムのパス名を意識する必要がありません。スクリプト名を指定するだけで、スクリプトファイルの取得・ダウンロード、新規作成や更新作業が出来ます。Linux のシェル上でクライアントプログラムとエディタを使用して、効率よくスクリプトの編集作業ができます。詳しい使用例などは "クライアントプログラム" の章の実行例をご覧ください。

スクリプトやイベントハンドラを変更すると、直後から abs_agent の動作に反映します

abs_agent 動作中にスクリプトをエディタで編集した場合でも、サーバーはすぐに新しいスクリプトに従って動作しますので、サーバーを再起動させる必要はありません。

abs_agent が動作しているコンピュータを samba[5] 等を使用して Windows PC から共有すると、Windows 上で動作するエディタソフトを使って abs_agent のスクリプトファイルを編集することが出来ます。この場合には abs_agent をインストールしたディレクトリ内のスクリプトファイルを直接変更したり、新規のファイルを作成することが簡単にできます。

スクリプト中で日本語を使用する

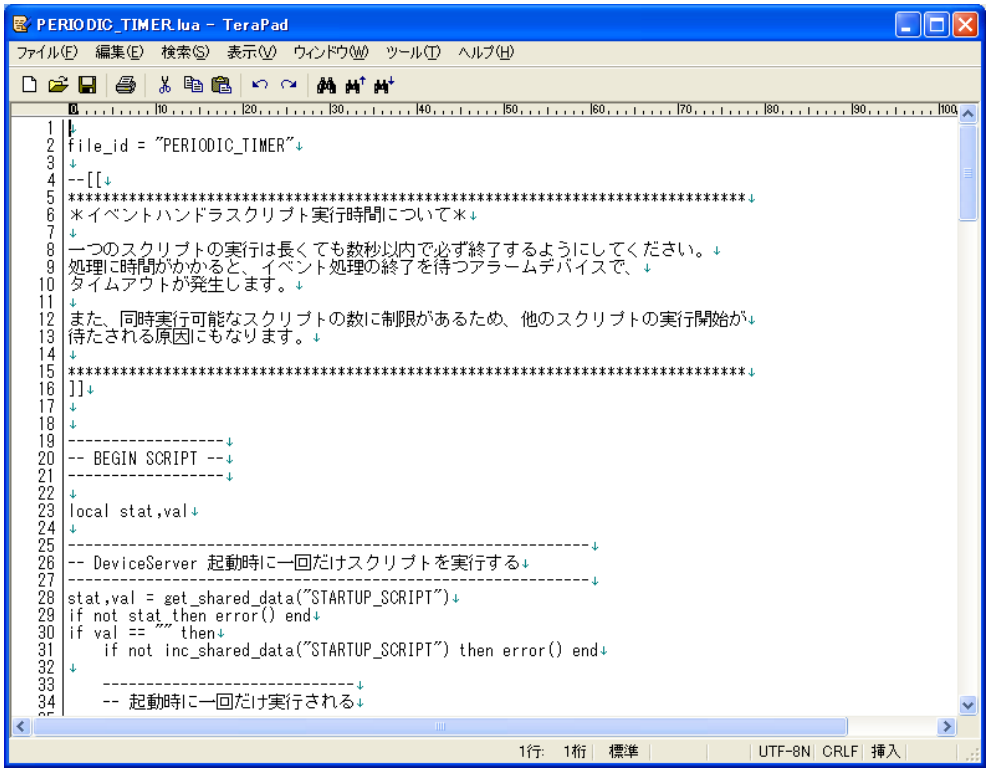
日本語を使用する場合には必ず UTF-8N(BOMなし) のエンコード形式で保存してください。 UTF-8(BOMあり)や Windows 標準の Shift_JIS 形式、その他のエンコード形式では動作しませんので注意してください。

スクリプトファイル中に使用する改行コードは、DOS形式の "CR-LF"と Unix形式の"LF"のどちらでも動作します。

Windows PC からネットーク共有してスクリプトファイルを編集する場合には、Windows付属のワードパッドやメモ帳では UTF-8N(BOM無し)形式で保存できませんので、別途 UTF-8N 形式で保存可能なエディタソフトを使用してください。エディタソフトウエアはフリーソフトの TeraPad (下記 URL 参照)をお勧めします。

http://www5f.biglobe.ne.jp/~t-susumu/

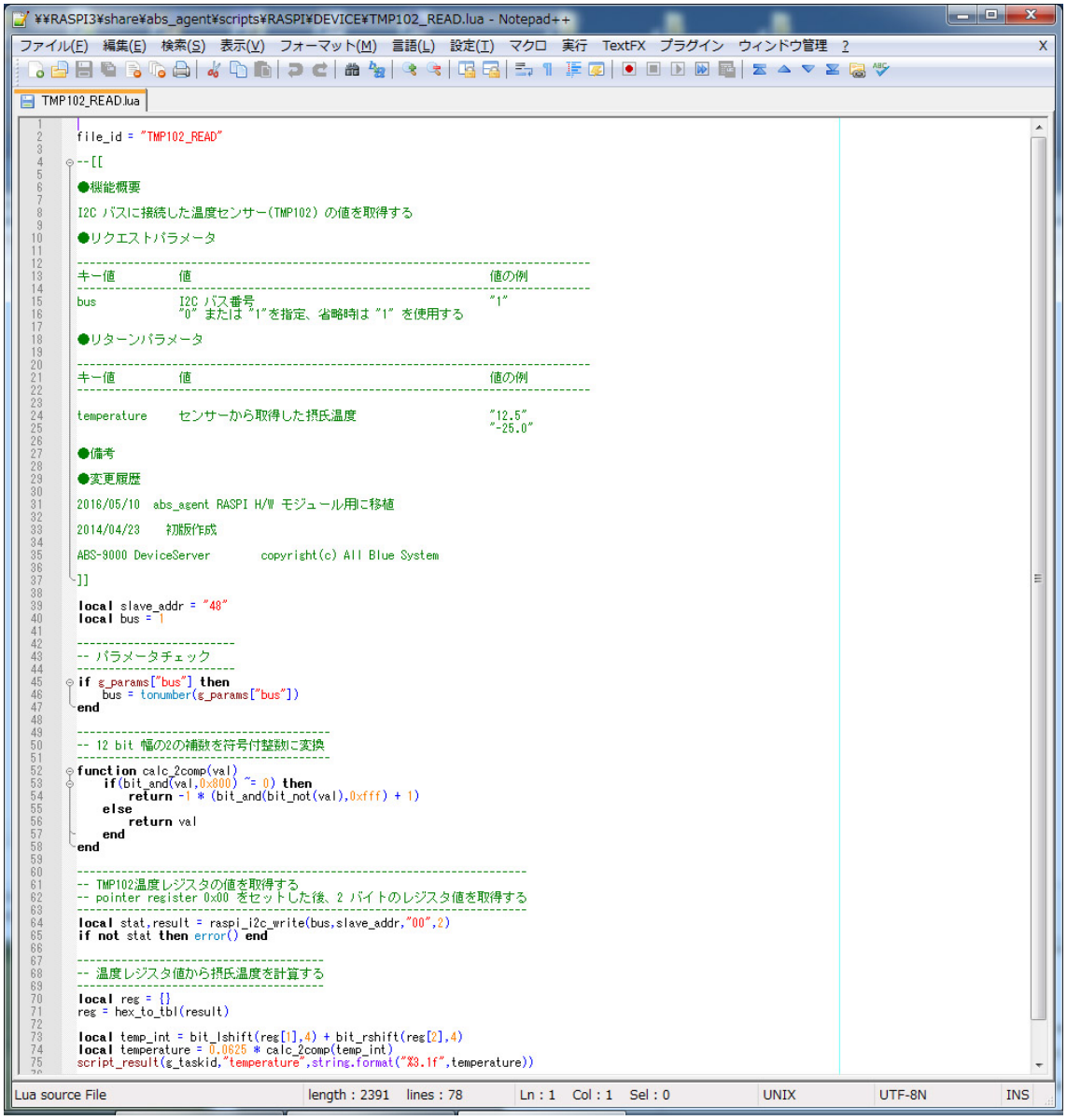
[5] samba https://www.samba.org/



TeraPad で abs_agent のスクリプトファイルを編集している画面です。この様に日本語をスクリプト中に記述する場合には、画面右下に表示されているコードが “UTF-8N” になっている必要があります。もし “SJIS” になっている場合には、ファイルメニューから”文字コード指定保存”を選択して、”UTF-8N” を選択して一度保存してから、再びオープンして編集作業を行ってください。スクリプトファイル中に日本語が含まれていない場合には “SJIS” のままで構いません。

また、Lua ステートメントの強調表示などを使用したい場合には下記の NotePad++[6] もお勧めします。

[6] NotePad++ (https://notepad-plus-plus.org/)



5 ログサーバー・インストール(オプション)

abs_agentは実行時に発生するイベント、エラーやその他のシステムメッセージ等をログに出力する機能があります。ログサーバーを設置しない場合でも abs_agentの動作に影響はありませんので、ログサーバーの設置は任意オプションになります。

ログの出力先は abs_agent 起動時のコマンドオプション "abs_agent -l <log_server_ip_address>" で指定したログサーバーに出力します。デフォルトではループバック・ホスト(127.0.0.1) が指定されていてログ出力は無効になっています。<log_server_ip_address> を ABS-9000 LogServer プログラムを設置した Windows コンピュータの IP アドレスもしくはホスト名に指定すると、abs_agentで出力したログメッセージを収集・保管したり、リアルタイムにメッセージを確認することができます。

複数の abs_agentで出力するログを１つのログサーバーに出力することもできます。オールブルーシステムの

abs_agent, DeviceServer, MGCP デバイス製品等も共通のログサーバーに出力できますので、全てのログを集中して管理することができるようになります。

abs_agent のログ出力用に ABS-9000 LogServer はフリーでご使用になれます

abs_agent で出力するログメッセージは、ネットワーク経由で専用のログサーバーに出力するように設計されています。これによって、複数の abs_agent で発生するイベントやメッセージを集中管理することができます。ログサーバー機能は ABS-9000 LogServer をインストールすると、Windows のサービスプログラムとして動作します。ログサーバーを設置しない場合や、ログサーバーに指定した PC が起動していない場合でも、abs_agent サーバーの動作には影響を与えません。ただし、ログサーバーが起動していない期間中に発生したログメッセージは保存されませんので注意して下さい。

5.1 ログサーバー動作環境

ログサーバーのインストールには ABS-9000 LogServer インストールキットを使用します。ABS-9000 DeviceServer のインストールキットを使用して、ログサーバー部分を併用することもできます。abs_agent のみを運用する場合には、ログサーバーとログコンソールプログラムのみをインストールする ABS-9000 LogServer インストールキットを使用する方が便利です。このマニュアルでは、ABS-9000 LogServer インストールキットを使用したインストール方法について説明します。ABS-9000 LogServer はフリーでインストールして使用することができます。

ログサーバーは、"ABLogService" の名前が付いた Windows サービスプログラムと、ログコンソールプログラム（Win32アプリケーション）の２つで構成されています。

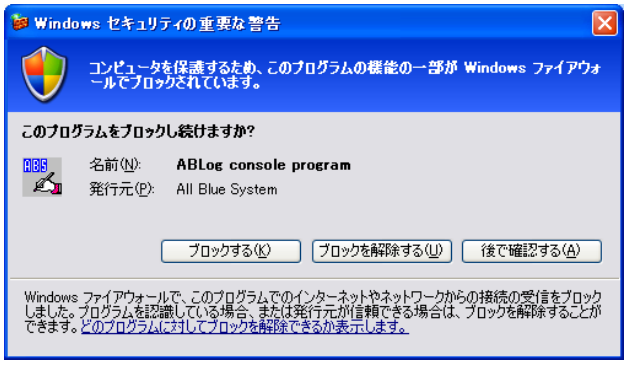
ログサーバーの動作環境は以下になります。インストール前に動作環境を満たしていることを確認してください。ここで示した Windows 以外の最新版 Windows OS で正常に動作可能と思いますが、これらの OS にインストールする場合には、Windowsの "UAC"（ユーザー・アカウント制御）機能を無効に設定してからインストールしてください。（詳しくは後述）

項目	必要なバージョン・リソース
オペレーティングシステム	Windows 2000 Professional Windows XP (32bit) SP2 または SP3 Windows 2003 Server (32bit) Windows 7(64bit) SP1
CPU クロックスピード	2GHz 以上
メインメモリ	1Gbytes 以上
ディスク容量（Cドライブ下の "Program Files"フォルダ）	150Mbytes 以上の空き容量
ネットーワークポート	10Mbps/100Mbps イーサネットポート１つ以上

MGCP デバイスなどログメッセージ送出側から、ログサーバーが動作しているコンピュータへ安定してネットワーク・アクセスするために、ログサーバーをインストールする Windows PC には必ず LAN 内の固定 IPアドレスをアサインしてください。

5.2 ログサーバーで使用するポート番号をWindows で利用可能にします

ログサーバーとログコンソールプログラムでは下記の表で示したネットワークポートを使用しています。ファイアウォールプログラムやセキュリティソフト等を使用している場合は、abs_agent 側からログメッセージを受信するために、Windows PC 側の設定変更が必要になります。ここで設定しない場合には、ログコンソール起動時や ABS-9000 LogServer インストール作業中に下記のようなダイアログが表示される場合があります。



この画面が表示された場合には、”ブロックを解除する” を選択することで該当するポートが使用可能になります。ただし、セキュリティ上 ABS-9000 LogServer インストールキット起動前に手動でポートを使用可能にしておくことをお勧めします。全てのポートを正しく設定した場合には上記のダイアログは表示されません。

ログサーバーとログコンソールプログラムが使用するプロトコルとポート番号		
プログラム・機能	**プロトコル**	**ポート番号(名称)**
ログサーバーのメッセージ送受信	UDP	2056 2057

ここでは、Windows XP と Windows 7のファイヤーウォール設定について説明します。他の OS や市販のセキュリティソフトを使用している場合には、それらのソフトウエアマニュアルを参照して同様のポート開放の設定を行ってください。

⚠市販のセキュリティソフトを使用している場合にはポート番号を開放して外部から接続可能にして下さい

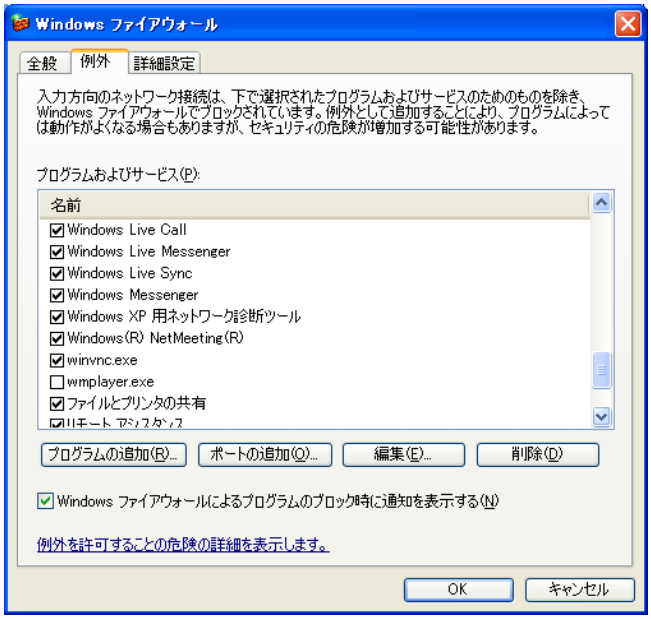
ログサーバーはネットワーク経由でアプリケーションやデバイス等からアクセスされますので、上記のポート番号を外部から接続可能に設定してください。セキュリティソフトがアクセス可能にする対象の、プログラム名を必要とする場合には

”C:¥Program Files¥AllBlueSystem¥ABLogServer.exe” (Windows XP) または、

”C:¥Program Files (x86)¥AllBlueSystem¥ABLogServer.exe” (Windows7 64bit) を指定してください。

5.2.1 Windows XP(SP3) でポート開放操作

コントロールパネルから Windows ファイヤーウォールを選択して、”例外” タブを選択します。

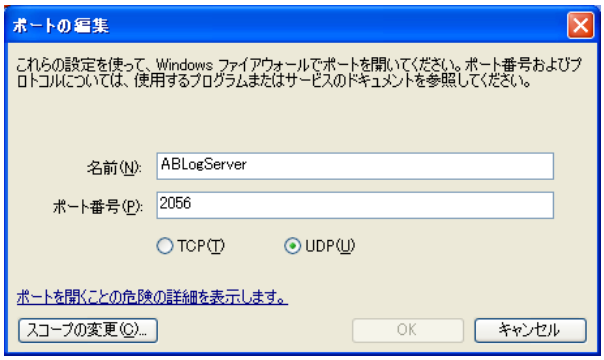


(Windows ファイヤーウォール 設定画面 Windows XP SP3 の場合)

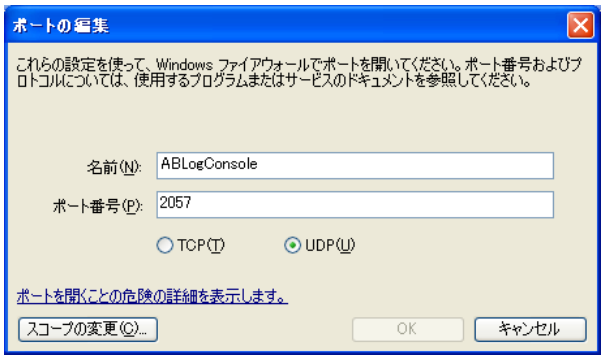
“ポートの追加” を押して、ログサーバーで使用するポート 2056/UDP の設定を追加して、

“OK” を押してください。ポートの編集ダイアログで記入する “名前” には、他の文字列を設定しても構いませんが、

ポート番号と TCP/UDP の区別は間違えないように注意して下さい。



ログサーバーのポート 2056/UDP の設定を追加します。(プロトコルが UDP になっている点に注意してください)

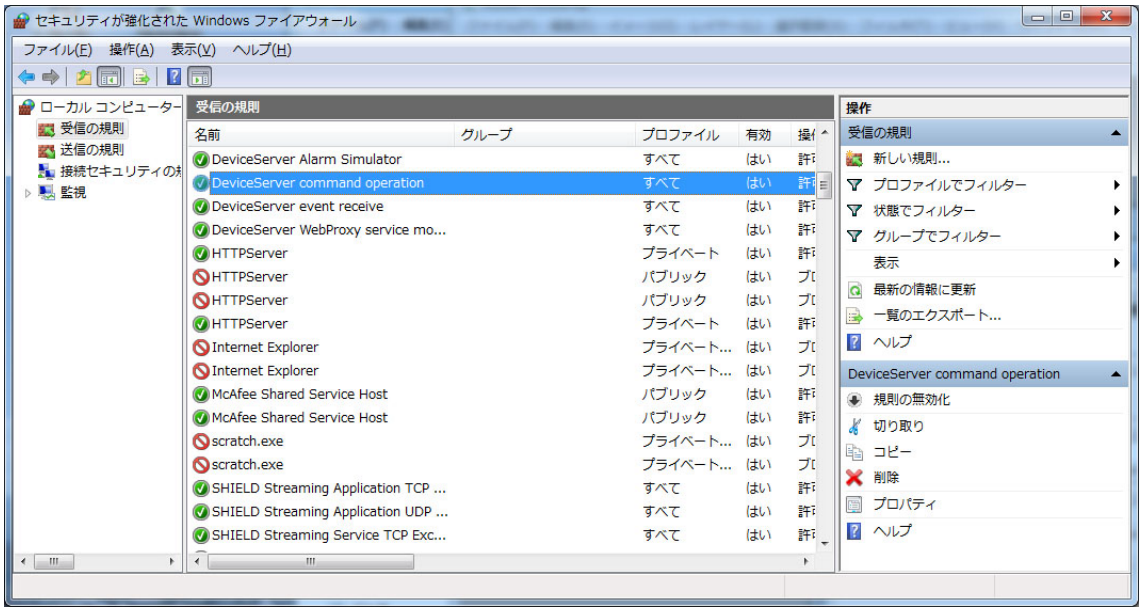


ログコンソールのポート 2057/UDP の設定を追加します。(プロトコルが UDP になっている点に注意してください)

以上でファイヤーウォールの設定は完了です。

5.2.2 Windows 7(SP1) でポート開放操作

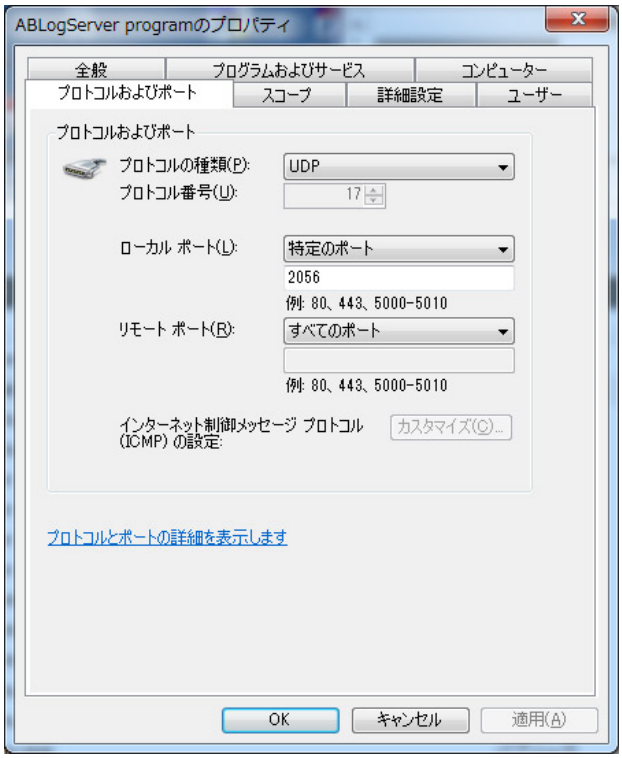
Windows 7 の場合にも同様にコントロールパネルから Windows ファイヤーウォール設定画面を表示します。詳細設定の中の "受信の規制" を選択してポートの開放操作を行います。



(Windows ファイヤーウォール 設定画面 Windows7 SP1 64bitの場合)

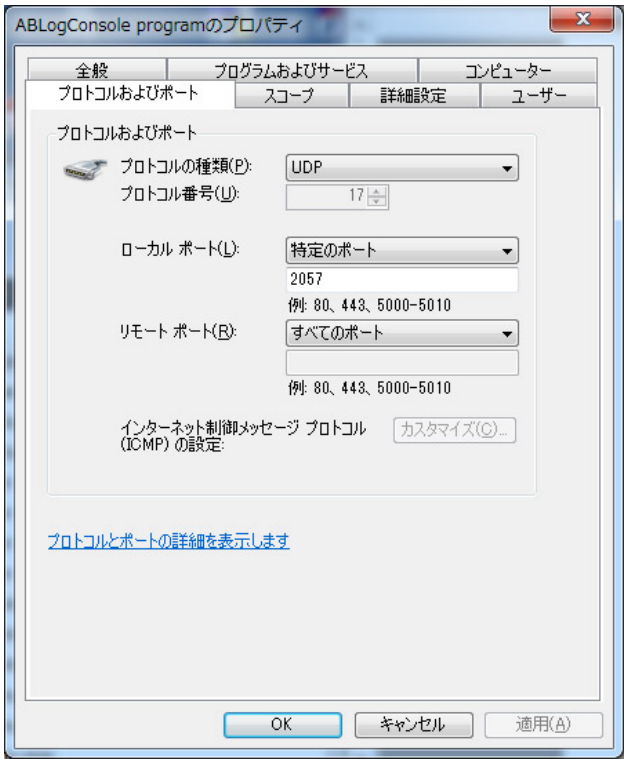
ファイヤーウォール画面の "新しい規制"を選択してウィザード形式で作業を行います。"ポート" を選択して TCP/UDP のポート番号の規制を作成する画面を出して Windows XP の場合と同様に UDP プロトコルの 2056と 2057 を開放します。

最初にローカルポート設定欄にポート番号(2056) を指定して、リモートポート設定欄には"全てのポート" を選択します。規制の名前は "ABLogServer program" にしていますが適当に付けても構いません。



(UDP/2056 ポートの開放を行う設定例 Windows7 SP1 64bitの場合)

次に"新しい規制"を選択して同様にポート番号(2057) の項目を追加します。この例では規制の名前に"ABLogConsole program" を指定しています。



(UDP/2057 ポートの開放を行う設定例 Windows7 SP1 64bitの場合)

全てのログサーバー用のポート開放作業が終了したときには、規制の一覧に下記の項目が追加されているのを確認してください。

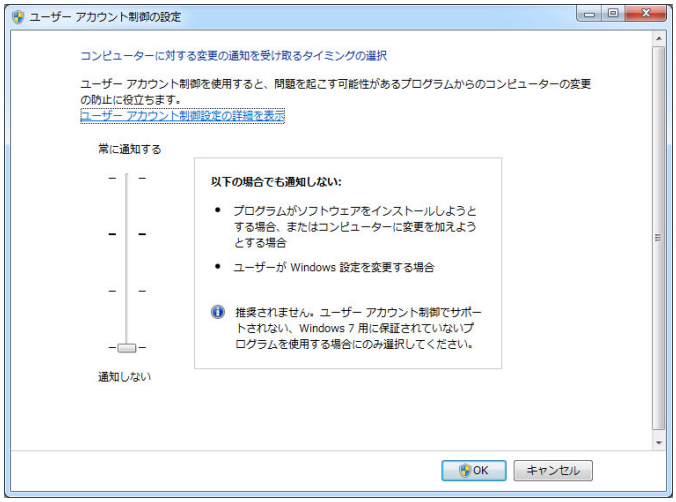

受信の規則

名前	グループ	プロファイル	有効	操作	優先	プログラム	ローカル アドレス	リモート アドレス	プロトコル	ローカル ポート	リモート ポート	許可されているユーザー	許可されているコンピュータ
ABLogConsole program		すべて	はい	許可	いいえ	任意	任意	任意	UDP	2057	任意	任意	任意
ABLogServer program		すべて	はい	許可	いいえ	任意	任意	任意	UDP	2056	任意	任意	任意

(Windows ファイヤーウォール設定画面の、ログサーバー動作用に追加した受信規制項目一覧)

5.3 ログサーバーの Windows UACを無効にする

ログサーバー は Windows2000,WindowsXP SP3、Windows2003 Server, Windows 7 SP1(64bit) の環境で動作試験をしていますが、その他の最新バージョンの Windows でも動作させることができます。OS のバージョンによって設定方法は異なりますが、コントロールパネル等から**UAC のアクセス制御を完全に "無効"(通知しない) に設定した後、OS を再起動する**ことで最新バージョンの Windows でも使用できます。



(Windows7 のユーザーアカウント制御の設定で "通知しない"に設定している様子)

これは、ABS-9000 DeviceServerインストールキットまたは、ABS-9000 LogServerインストールキットでインストールされる "ログコンソールプログラム" が、Windows の SCM (サービスコントロールマネージャ) 機能を使用しているためです。"ログコンソールプログラム" 起動時に、ユーザーアカウントの警告ダイアログが出力される事があります。この警告ダイアログが表示された場合には、上記の設定を行った後もう一度 "ログコンソールプログラム" や "サーバー設定プログラム" を起動してください。一度設定すればこのダイアログは表示されなくなります。

その他の ログサーバー等のサービスプログラムは Windows7 で完全に動作します。お客様がご使用中の Windows 環境で、セットアップ時や運用中に問題が発生した場合には下記のメールアドレス宛にご連絡下さい。

問い合わせメールアドレス

contact@allbluesystem.com

5.4 ログサーバーインストール例

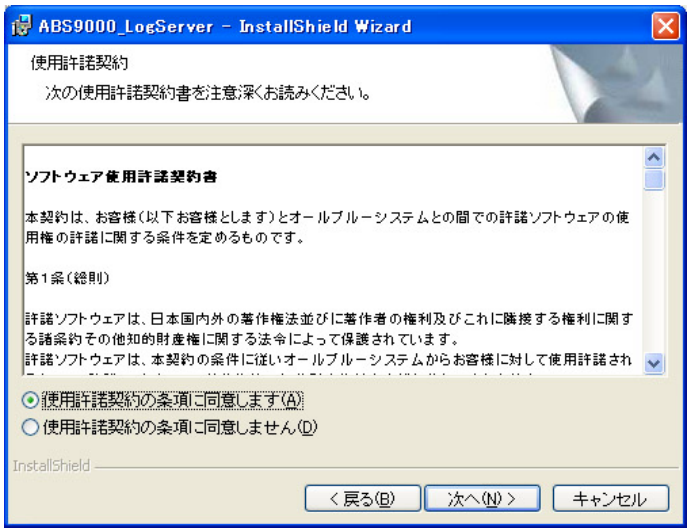
ここからは、ログサーバーに使用する Windows PC に ABS-9000 LogServer のインストールキットを使用してインストール作業を行います。

インストール時に "ABLogService" の名前が付けられた Windows サービスプログラムを登録します。全てのセットアップはウィザード形式で行われます。インストール中に表示される設定項目は、全てデフォルト値で設定できますので、特に設定内容を変更しなくてもクリックを押すだけでインストールが完了します。

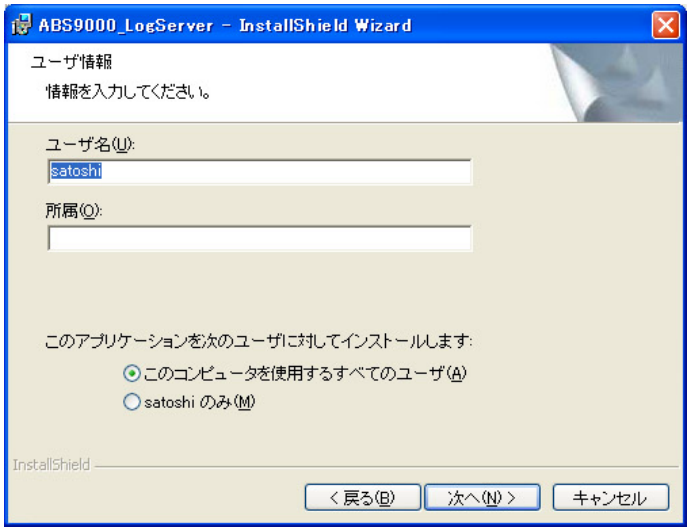
インストールキットを実行するときは、Windows のシステム管理者権限のユーザー(Administrator等)で行います。Windows 一般ユーザーの権限ではセットアップできませんので注意してください。

インストールキットは ZIP 形式で１つのファイルにまとめられていますので、適当なディレクトリにファイルを展開してください。WindowsXP, Windows2003, Windows7 の場合にはファイルを右クリックして “全て展開” メニューでファイルを展開することができます。ファイルを展開したら、”setup.exe” をダブルクリックしてインストーラプログラムを起動します。

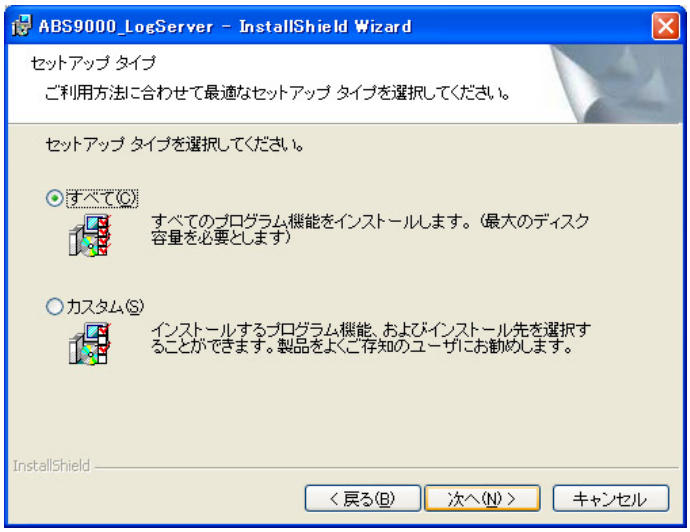
以下は Windows XP(SP3) にインストールするときの表示例です。他の Windows で作業するときも操作は同一です。



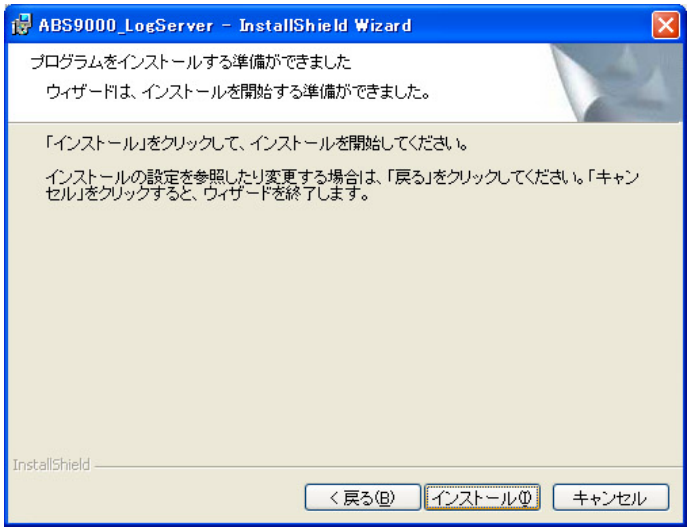
使用許諾をよく読んだ上で、同意するにチェックを付けてください。同意されない場合は、ABS-9000 LogServer を使用することはできません。



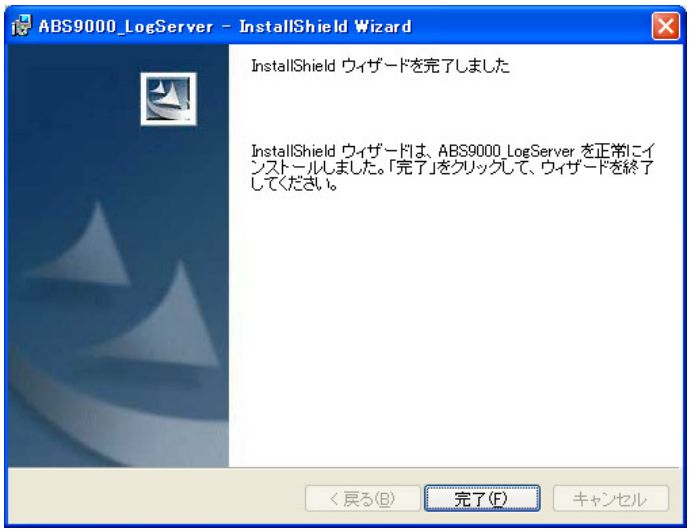
LogServer はデフォルトの ”このコンピュータを使用するすべてのユーザ “を選択してください。ユーザー名や所属などは変更する必要はありません。



セットアップタイプはデフォルトの “すべて”を必ず選択してください。デフォルトのインストールフォルダを変更した場合や、一部のコンポーネントのみのインストールを選択すると正常に動作しません。

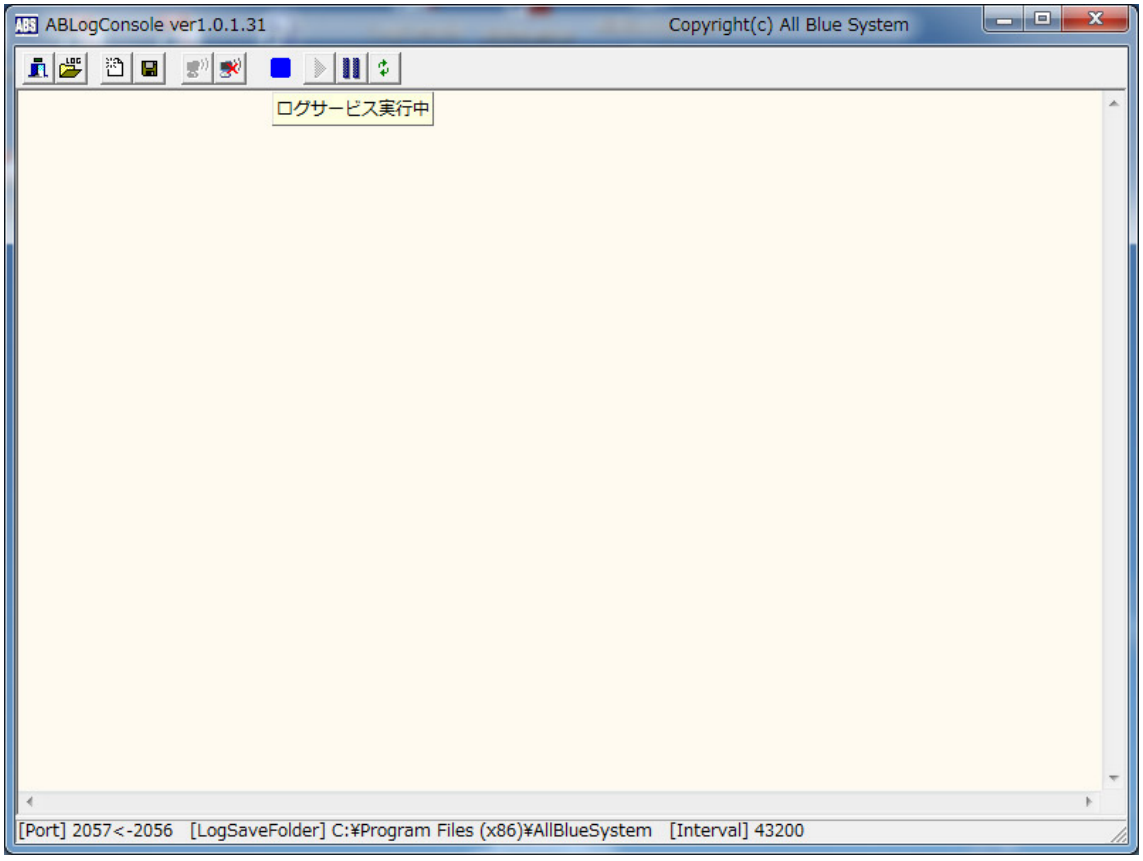


インストール準備が完了しましたので “インストール”を押してください。



これで、Windows PC 上にログサーバーの設置が終了しました。ログサーバーはサービスプログラムで実行していますので、Windows PC が起動している間は常にログメッセージを収集して内部で保管しています。

プログラムメニューから "All Blue System" -> "ログコンソール" を選択してコンソールプログラムを起動します。ログコンソールプログラムを起動すると、現在ログサーバーで取得中のメッセージを画面に表示することができます。



（ログコンソールプログラムを実行している様子）

ログコンソールプログラムから、ログサーバーで保存している過去のログファイルを参照することもできます。また、現在ログサーバー中でメモリ中にキャッシュされているログメッセージを強制的にファイル出力させる機能もあります。ログコンソールプログラムを終了してもログサーバープログラムは常にサービスプログラムとして実行中ですので、ログメッセージが失われることはありません。詳しい機能は後述します。

5.5 ログサービス(ABLogServer)機能

オールブルーシステム製の abs_agent, MGCP, DeviceServer等で出力した全てのログ情報を記録します。
クライアントプログラムの幾つかは直接ログサービスにイベントを記録するものがあり、これらも同様に記録します。

ログサービスは ABS-9000 LogServer をインストールしたWindows コンピュータ上のサービスプログラムとして動作しています。ユーザーが直接このサービスプログラムを操作することはありません。アプリケーション(abs_agent, MGCP, DeviceServer等)で検出したエラーやイベント、クライアントアクセスの記録、ユーザーが作成したスクリプトからのログ出力を、集中管理して記録しています。

ログサービスは記録動作を高速に行うために、通常はメモリ中にログ情報を保管していて、定期的にファイルに書き出す方法を採用しています。定期的に出力されたログファイルは、いつでもユーザーが参照できます。後述のログコンソールプログラムを起動することで、リアルタイムにログ出力を画面に表示することや、ログサービスに対してメモリ中に保管中のログを強制的に、ファイル出力するように指示することもできます。

Windows サービス名	ABLogService
プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABLogServer.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogServer.exe
設定ファイル名	C:¥Program Files¥AllBlueSystem¥ABLogService.ini Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogService.in
バージョン番号の確認方法	プログラムファイルのプロパティを開いて、バージョン情報タブ内のファイルバージョンを確認する
ログファイル出力フォルダ [Folder]	C:¥Program Files¥AllBlueSystem¥Logs Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥Logs
ログファイル名	“abs” + *YYYYMMDDHHMMSS* + “.log” ファイル作成時のタイムスタンプをファイル名に利用する。
ログファイル切り替えタイマー間隔 [Interval]	約 6 時間（設定ファイルの単位は秒で指定される） ログコンソールプログラムから強制的にファイルを切り替えることもできる。またログサービス再起動時も自動的に切り替えが行われる。
ログイベント受信ポート(UDP) [ReceivePort]	2056
ログイベント送信ポート(UDP) [RepeatPort]	2057 ログコンソールプログラムでリアルタイムにログ情報を表示するために使用する。
ログファイル保管期間 [LogKeepDays]	30 日 保管期間を過ぎたログファイルは自動的に削除される

上記の設定項目の幾つかは、設定ファイル(ini ファイル)を修正することで、設定値を変更することができます。”[xxxx]” 部分が対応する設定ファイルのタグ名です。詳しくは設定ファイルの内容を参照してください。（ただし、ログイベントポート番号の変更は行わないで下さい。サーバーからのログを受信できなくなります）設定を変えた場合はログコンソールプログラムからログサービスを再起動するか、PC を再起動することで新しい設定内容が有効になります。

設定ファイルの内容(例)

```
[ABLogService]
ReceivePort=2056
RepeatPort=2057
Interval=21600
Folder=C:¥Program Files¥AllBlueSystem¥Logs
LogKeepDays=3
```

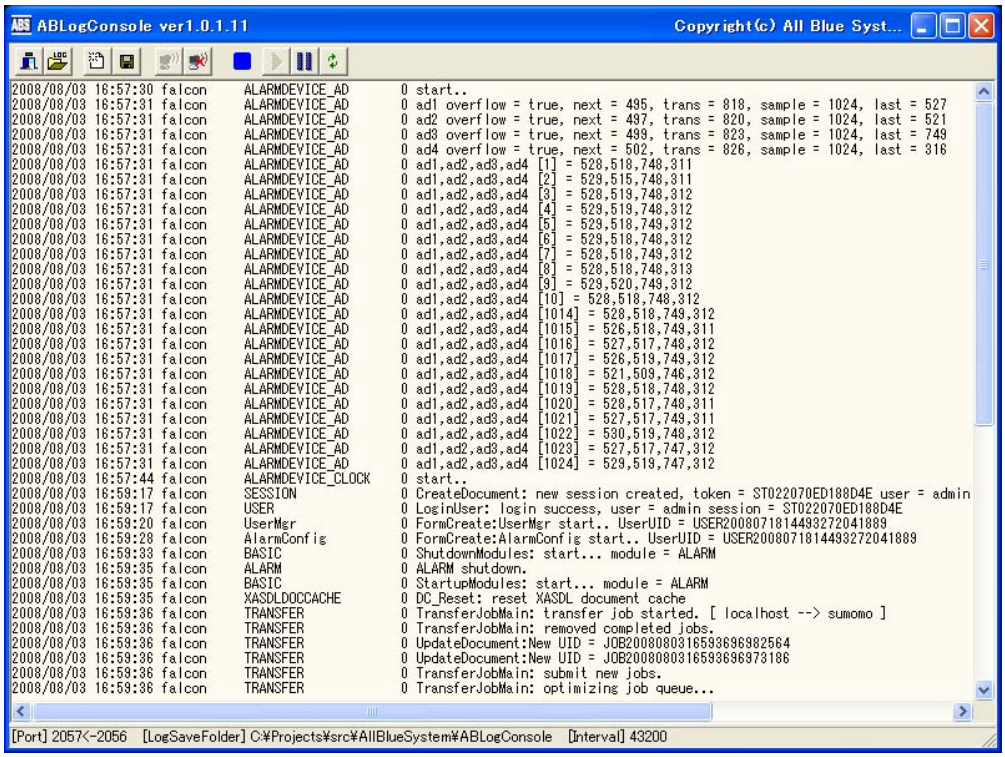
5.6 ログ管理コンソール(LogConsole)機能

ABS-9000 LogServer の動作するPC で起動させて、ログサービス(前述) に記録しているイベントを画面上でリアルタイムに確認することができます。

ログ管理コンソールは何時でも起動することが可能で、そのときに発生しているログイベントをログサービスから取得して表示します。ログ管理コンソールを終了してもログイベントは常にログサーバーで収集・保管していますので、ログが失われることはありません。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABLogConsole.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLogConsole.exe
バージョン番号の確認方法	プログラム起動時のメインフォームタイトルに表示される

プログラムを起動すると自動的にログサーバーで収集している現在のログメッセージを画面に表示し続けます。



各ツールボタンの機能を以下に説明します。

ツールボタン	説明
アプリケーション終了 	ログ管理コンソールプログラムを終了する。 ログイベントの記録はログサービスで行われるので、ログが失われることはない。
ログフォルダを開く 	ログサービスで保管されたログファイルを参照する。 ボタンを押すとエキスプローラでフォルダが展開される。過去に保存されたログをここから選択してワードパッドやメモ帳などで表示してください。
メッセージクリア	画面に表示されたログメッセージをクリアする。

	
メッセージセーブ 	画面に表示されたログメッセージをファイルに保存する。 ログサービスで自動保管されるログファイルとは別に、ログ管理コンソールの画面上にあるメッセージのみをユーザーの指定したファイルに保存するために使用する。
ネットワーク接続 	ログ管理コンソールでログメッセージを受け付ける状態にする。 （起動時は接続済のため、選択できないようになっています）
ネットワーク切断 	ログ管理コンソールでログメッセージを受け付けない状態にする。 ログ管理コンソールの画面上のログ更新を一時的に停止したい場合に使用する。切断中のログは破棄されるので、その間のログを後から確認したい場合は、ログサービスで自動的に保管されたログファイルを参照してください。
ログサービス開始 	ログサービスを開始する。 （通常は、サービスは常に動作中のため選択できないようになっています）
ログサービス停止 	ログサービスを停止する。 ログサービスの設定ファイルを修正したい場合等に、サービスを停止するために使用します。
ログファイル切り替え （ログファイルフラッシュ） 	ログサービスの出力ファイルを強制的に切り替える。 このボタンを押すと、ログサービスで予め設定されたログファイル切り替えタイマー間隔(デフォルトで 6 時間) を無視して、強制的にファイル切り替えが行われます。 現在メモリ中にあるログメッセージを直ぐに確認したい場合に使用します。

6 アンインストール・アップデート

6.1 abs_agent アップデート

abs_agent をアップデートする場合は動作中の abs_agent を停止した後、最新のインストールキットで実行ファイルやスクリプトを上書きします。ただし、既存のスクリプトやライブラリファイルを単に上書き更新すると、現在の動作環境を壊してしまうため、最新のインストールキットを展開する前に下記のスクリプトについては手動で更新前の準備を行う必要があります。

(上書き更新時に事前に準備作業が必要なファイル)

* ユーザーがインストールキットに含まれるデフォルトスクリプトファイルに更新を加えたもの
* ユーザー自身が作成したユーザースクリプトやプリロードライブラリ
* プリロードライブラリ内にあるディレクトリでアンダースコアから始まるフォルダ名の "_" 部分を削除して有効にしている部分

上記のユーザー更新部分については、最新のインストールキットで展開したファイルにこれらの変更部分を適切に反映させる必要があります。これらの準備・更新作業を簡単に行うために abs_agent にはインストールキットで展開したディレクトリとは別に、動作環境ディレクトリを作成・参照する機能があります。また、abs_agent 動作環境で使用(参照)しているファイルと、インストールキットで展開した最新ファイルを比較して、変更点があるかどうかを調べるためのクライアントプログラム "agent_copycfg" を使用できます。これらの機能やプログラムを使用するこ

とで、アップデート作業を半自動的に実行することができます。これらの機能とプログラムについての詳しい説明は、"abs_agentインストール" 章中の "abs_agent 動作環境を別ディレクトリに作成"の項と、"クライアントプログラム" 章中の "agent_copycg" 項の使用例を参照してください。

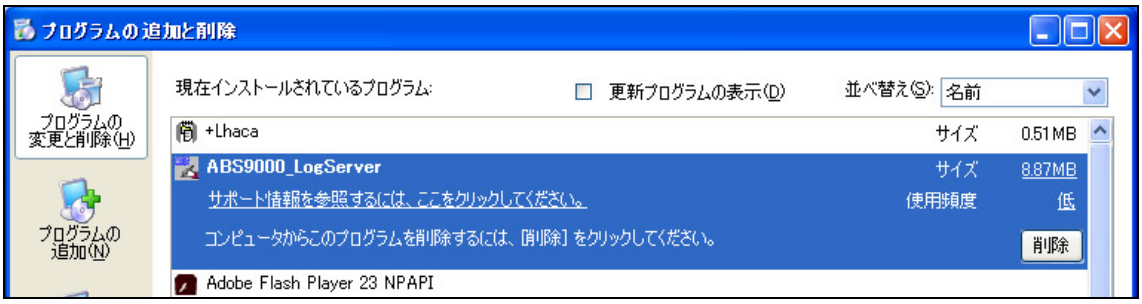
6.2 abs_agent アンインストール

abs_agent のアンインストールは、インストールキットを展開したディレクトリごと削除するだけで完了します。自動起動の設定を /etc/rc.local に設定している場合には abs_agent 起動部分の記述も同時に削除して下さい。

6.3 ログサーバーアンインストール

ログサーバーを削除する場合には、設置した Windows PC 上のコントロールパネルから、"プログラムの追加と削除" で "ABS-9000 LogServer" を指定して削除してください。アンインストーラでは、ログサーバーのサービスプログラムも自動的に削除します。

Windows のシステム管理者権限のユーザー(Administrator等)でWindows にログインした後、Windows コントロールパネルの"プログラムの追加と削除"から "ABS9000_LogServer" を選択して "削除" を押します。



6.4 手動でコピーした XASDLCMD.DLL アンインストール

後の章の "リモートクライアント設定" で説明する様に、手動で DLLライブラリファイル(XASDLCMD.DLL) をインストールした場合には、このファイルを手動で削除します。ABS-9000 LogServer インストールキットを使用してインストール・アンインストールした場合には自動で削除されますのでこの作業は必要ありません。

下記のファイルを削除して下さい

削除する DLL ファイル
Windows システムディレクトリ
WindowsXP,Windows2003 の場合は **"C:¥Windows¥System32¥XASDLCMD.DLL"**
Windows7(64bit) の場合には **"C:¥Windows¥SysWOW64¥XASDLCMD.DLL"**

7 動作確認と簡単な使い方

ここからは abs_agent をインストールした CPUボード (Raspberry Pi) に、周辺機器(LED, タクトスイッチ)を接続して abs_agent の簡単な使い方を説明します。Raspberry Pi 以外のハードウエアで動作させている場合でも、ユー

ザースクリプトの実行やイベントハンドラの使い方について参考にして下さい。

シリアルポートにデバイスを接続したり、MQTT ブローカとの通信を行う方法については以降の章で説明します。

WebAPI を利用したアプリケーション例については、"HTTPサーバー&WebAPI" の章中の "WebAPI使用例"の項と、"Web アプリケーション" の章で説明している集計アプリとそのソースファイル(インストールキットに含まれています) を参照してください。WebSocket サーバー使用例については "WebSocketサーバー"の章中の "WebSocket使用例"の項をご覧ください。

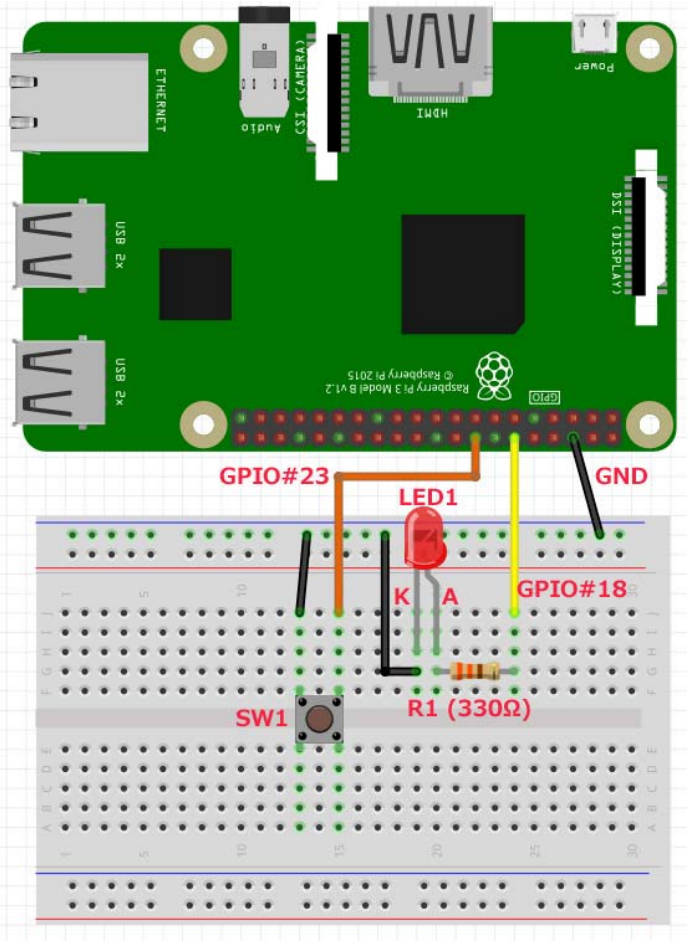
⚠ **配線やスクリプトの内容によっては、回復不能なダメージをコンピュータに与える可能性があります**

ここでは Raspberry Pi の拡張ポートを利用して、外部に簡単なハードウエア回路を接続します。電子回路の基本的な知識がある方が操作されることを前提に説明しています。回路の配線時にはコンピュータの電源を切ってから作業することをお勧めします。配線時に電源ラインと GND 間をショートしたり、決められた耐圧以上の信号をポートに入力すると Raspberry Pi が壊れる可能性がありますので、慎重に作業を行って下さい。

7.1 Raspberry Pi に LED とスイッチを接続

Raspberry Pi ver3 にスイッチと LED を接続して、スクリプトから LED を操作したり スイッチ入力に対応して処理を行う例を説明します。Raspberry Pi ver3 に接続する場合の例を説明しますが、他のバージョン Raspberry Pi を使用する場合には、拡張ポートに配線する GPIO ポート番号と GND を合わせて接続してください。

下記が配線図になります。LED には電流制限抵抗を付けて GPIO#18 に接続しています。タクトスイッチは押したときに GPIO#23 が GND 間と短絡するようにします。

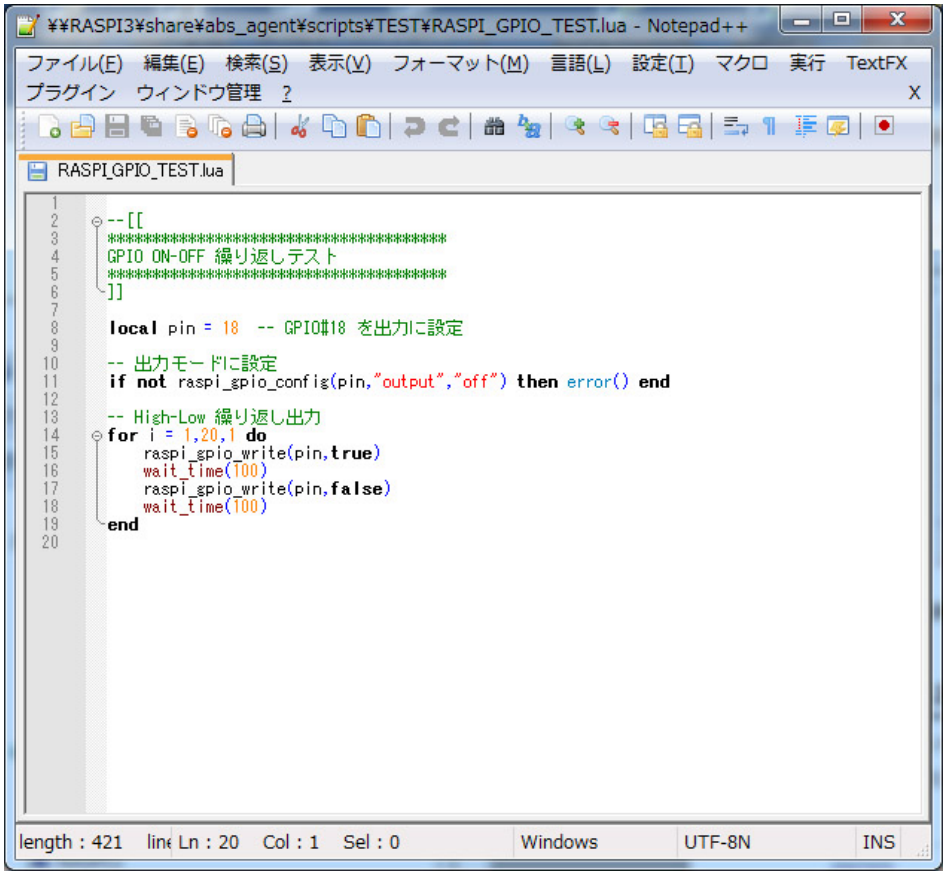


(配線例です。抵抗値は値が多少増減しても問題ありません)

7.2 GPIO に接続した LED 点滅

abs_agent のスクリプトから LED の点滅を行ってみます。使用するスクリプトはインストールキット中に含まれる TEST/RASPI_GPIO_TEST を使用します。実際のスクリプトファイルは abs_agent をインストールしたディレクトリ以下の scripts ディレクトリの中にファイル拡張子(.lua)付きで格納されています。この例では /home/pi/abs_agent/scripts/TEST/RASPI_GPIO_TEST.lua になっています。

スクリプトの内容の様になっています。



最初に、下記のライブラリコールでLED が接続されている GPIO#18 を出力モードで内部プルアップを無効にしています。

```
if not raspi_gpio_config(pin,"output","off") then error() end
```

次の "for" 文で 20 回の繰り返しを指定しています。

```
for i = 1,20,1 do
        raspi_gpio_write(pin,true)
        wait_time(100)
        raspi_gpio_write(pin,false)
        wait_time(100)
end
```

raspi_gpio_write() ライブラリ関数を使用して GPIOポートに値を出力します。第二パラメータに true を指定すると High(3.3V) が出力され、false を指定すると Low(0V) が出力されます。wait_time(100) は 100ms のスリープを指定しています。

スクリプトを実行してみます。abs_agent をインストールしたディレクトリに移動して、スクリプト実行を行うクライアントプログラム "agent_script" を実行します。

```
./agent_script -t -s TEST/RASPI_GPIO_TEST
```

LED が点滅するのを確認できると思います。agent_script コマンドで "-t" オプションをつけているのは、スクリプト実行の終了を待たずにコンソールにプロンプトを返すためです。このときサーバー側(abs_agent側)では、別スレッドで TEST/RASPI_GPIO_TEST スクリプトが実行されています。"-t" オプションを指定しない場合には、スクリプト実行が全て終了(20回の LED の点滅繰り返しが完了)するまでプロンプトは戻りません。

スクリプト中で使用しているライブラリ関数詳細は、機能毎のライブラリ関数APIの章を参照してください。

7.3 スクリプト編集作業

ここからは、コンソール上でスクリプトの内容を変更して LED の点滅を変化させます。abs_agent が動作しているディレクトリを Windows PC 等からファイル共有している場合には、Windows 上で動作しているエディタ・プログラムを使用してスクリプトファイルを編集することもできます。Windows から編集する場合には、共有ディレクトリ経由で、abs_agent が動作しているコンピュータ側の /home/pi/abs_agent/scripts/TEST/RASPI_GPIO_TEST.lua ファイルを直接オープンして編集してください。
このマニュアルでは Linux のコンソールを使用して作業する例を説明します。

作業の流れとしては、スクリプトの内容を直接編集するのではなく一旦ダウンロード（コピー)してワークファイルを作成します。このワークファイルを編集した後、abs_agent にアップロードして既存のスクリプトファイルを更新します。このようにすることで、スクリプトのデバッグから更新までの作業を簡単に行えます。また、リモート側のスクリプトを編集する場合でも、同様の作業手順で行えます。ワークファイルを使用せずに直接エディタでスクリプトファイル /home/pi/abs_agent/scripts/TEST/RASPI_GPIO_TEST.lua を編集してももちろん構いません。

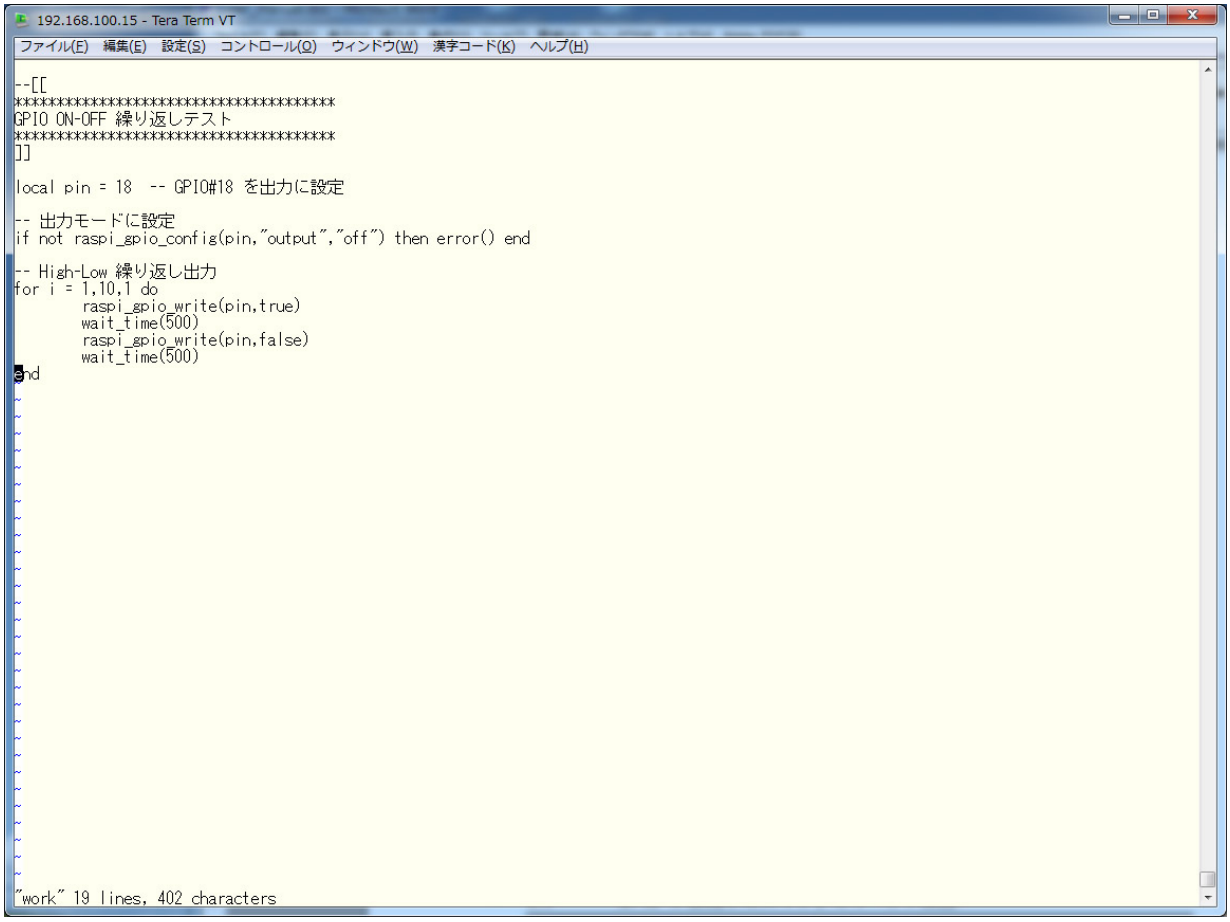
最初に TEST/RASPI_GPIO_TEST スクリプトを agent_get コマンドでダウンロードして "work" の名前でファイルを作成します。"-s" オプションでダウンロード対象のスクリプト名を指定します。また、"-f" オプションではローカルに作成するファイル名を指定します。

```
./agent_get -s TEST/RASPI_GPIO_TEST -f work
```

work の名前でファイルが作成されますので、これを vi エディタ等で編集します。今回は wait_time(100) 部分のスリープ期間を変更して 500ms にします。また、全体の繰り返し回数を 10 回にしてみます。

```
vi work
```

編集している様子は下記の様になります。



編集が終了したら work ファイルを更新してエディタを終了します。

次に、work ファイルを abs_agent のスクリプトファイルとしてアップロード(更新)します。更新時には agent_put コマンドを使用します。"-f" でアップロードするファイルを指定します。

```
./agent_put -s TEST/RASPI_GPIO_TEST -f work
```

"-s" で指定するスクリプト名はダウンロード時に指定したスクリプト名と同一にします。これによって、既存のスクリプトが更新されます。変更後のスクリプトを agent_script コマンドで実行します。

```
./agent_script -t -s TEST/RASPI_GPIO_TEST
```

LED 点滅間隔と回数が変化したのを確認できると思います。

ここで繰り返し回数を 100 に変更してみます。同時に wait_time() ライブラリコールをコメントアウトして高速で LED の点滅させます。この時、先ほど編集したワークファイルがローカルに残っている筈ですのでこれを編集します。

```
vi work
```

編集中の画面は下記の様になります。wait_time() 文の前に "--" でコメントを指定しています。

192.168.100.15 - Tera Term VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)

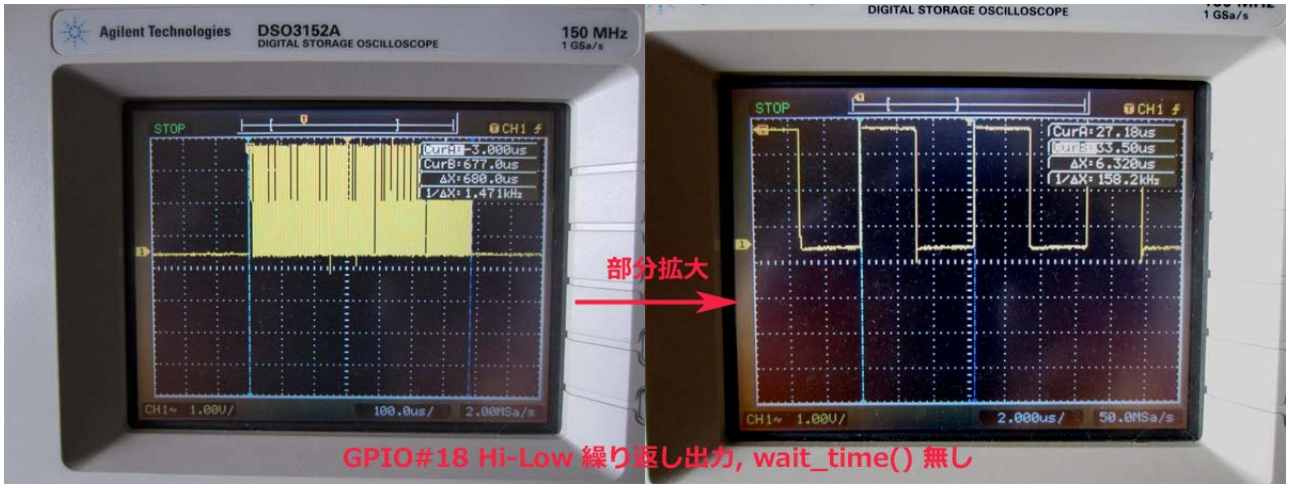
```
--[[
**************************************
GPIO ON-OFF 繰り返しテスト
**************************************
]]

local pin = 18  -- GPIO#18 を出力に設定

-- 出力モードに設定
if not raspi_gpio_config(pin,"output","off") then error() end

-- High-Low 繰り返し出力
for i = 1,100,1 do
        raspi_gpio_write(pin,true)
        -- wait_time(500)
        raspi_gpio_write(pin,false)
        -- wait_time(500)
end
~
~
~
~
```

先ほどと同様に、アップロードしてスクリプトファイルを更新して実行します。

```
./agent_put -s TEST/RASPI_GPIO_TEST -f work

./agent_script -t -s TEST/RASPI_GPIO_TEST
```

実行できましたか？ただし、目では LED の点滅を確認できないと思います。

abs_agent(Raspberry Pi用のビルド版）では ハードウエア機能をアクセスするライブラリ関数は直接 CPU のレジスタを操作していて、非常に高速に動作します。このため、単純な GPIO 出力の繰り返し文だと実行速度が早すぎて目では確認できません。下記は、この時の GPIO#18 出力をオシロスコープ（ストレージタイプ)で観察した様子です。



100 回の繰り返しは約 680μs で完了します。また、High -> Low 繰り返しの周期は約 150KHz に達します。

（上記の波形は Raspberry Pi ver3 で実行したときの一例です。Raspberry Pi ver1 では約３倍時間がかかります。また Linux OS の性質上、常にスレッドの実行が継続されるとは限りません。割り込みや他の優先度の高いプロセスやスレッド実行の為に、途中で意図しないウエイトが入ることがあります）

7.4 GPIO に接続したスイッチでイベント検出

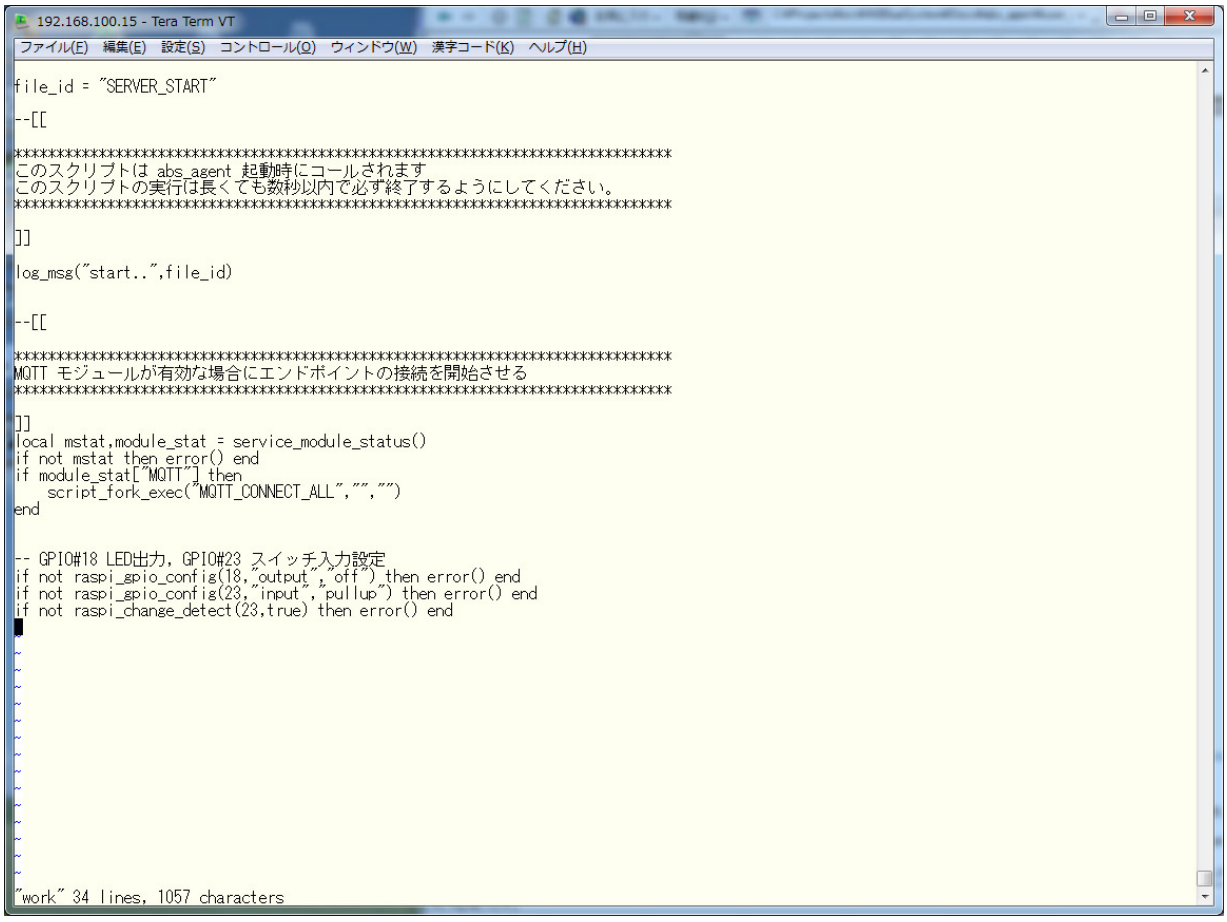
次は、GPIO#23 に接続したスイッチ入力を検出して LED の点灯と消灯を切り替えるようにしてみます。GPIO の入出力モードやプルアップ設定は、前項の LED の点滅スクリプトに記述した raspi_gpio_config() ライブラリ関数で行います。今回は、サーバー起動時に設定することで常にスイッチ入力を監視できるようにします。abs_agent サーバー起動時には SERVER_START イベントハンドラが実行されますので、ここにライブラリ関数をコールする内容を追加します。前項と同様に agent_get クライアントプログラムでスクリプトをダウンロードして編集します。イベントハンドラのスクリプトは、イベント名と同じスクリプトファイルがインストール時に設定されていますのでこれを編集します。

```
./agent_get -s SERVER_START -f work
vi work
```

スクリプトの最後の部分に下記のステートメントを追加します。raspi_change_detect() ライブラリ関数をコールして、指定した GPIO の値が変化した時に RASPI_CHANGE_DETECT イベントハンドラを実行する様に指定します。

```
-- GPIO#18 LED出力, GPIO#23 スイッチ入力設定
if not raspi_gpio_config(18,"output","off") then error() end
if not raspi_gpio_config(23,"input","pullup") then error() end
if not raspi_change_detect(23,true) then error() end
```

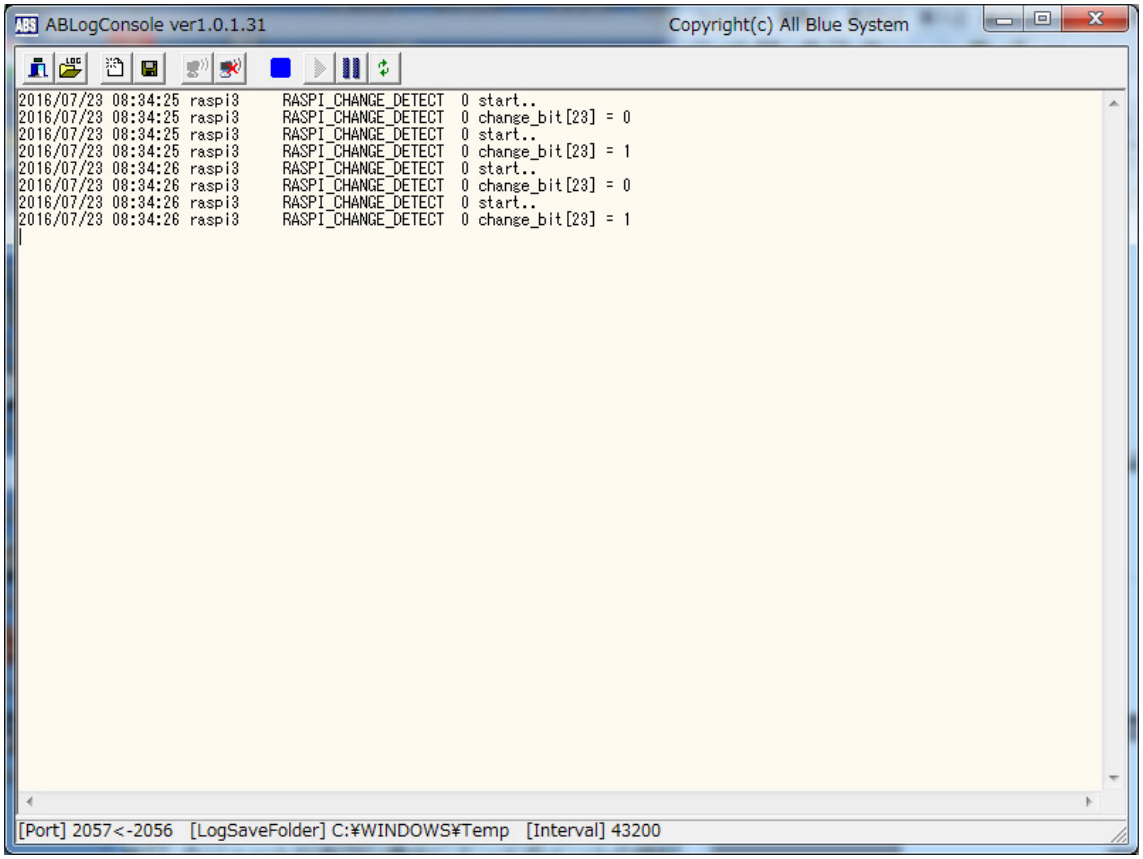
下記はコンソールで編集中の様子です。



編集が完了したら、agent_put コマンドで既存の SERVER_START スクリプトを更新します。その後、abs_agent を再起動させて SERVER_STRAT スクリプトの実行を行います。agent_再起動時には agent_shutdown で一旦サーバーを停止コマンドを実行して完全にサーバーが停止するまで 10 秒程度待った後に、abs_agent を起動しています。

```
./agent_put -s SERVER_START -f work
./agent_shutdown
sudo ./abs_agent -l 192.168.100.45
```

ここで、ログサーバーを設置している場合にはログコンソールを開いてください。Raspberry Pi に接続したタクトスイッチを操作すると下記の様なログが表示されます。



インストール直後の RASPI_CHANGE_DETECT イベントハンドラでは変化した GPIO ポートの値を単にログ出力するだけの内容が記述されています。次に、このイベントハンドラを編集して LED の ON-OFF を切り替えるようにします。

先ほどと同様に、agent_get コマンドで RASPI_CHANGE_DETECT スクリプトをダウンロードして編集します。

```
./agent_get -s RASPI_CHANGE_DETECT -f work
vi work
```

スクリプトの最後の部分に下記のようなステートメントを追加します。

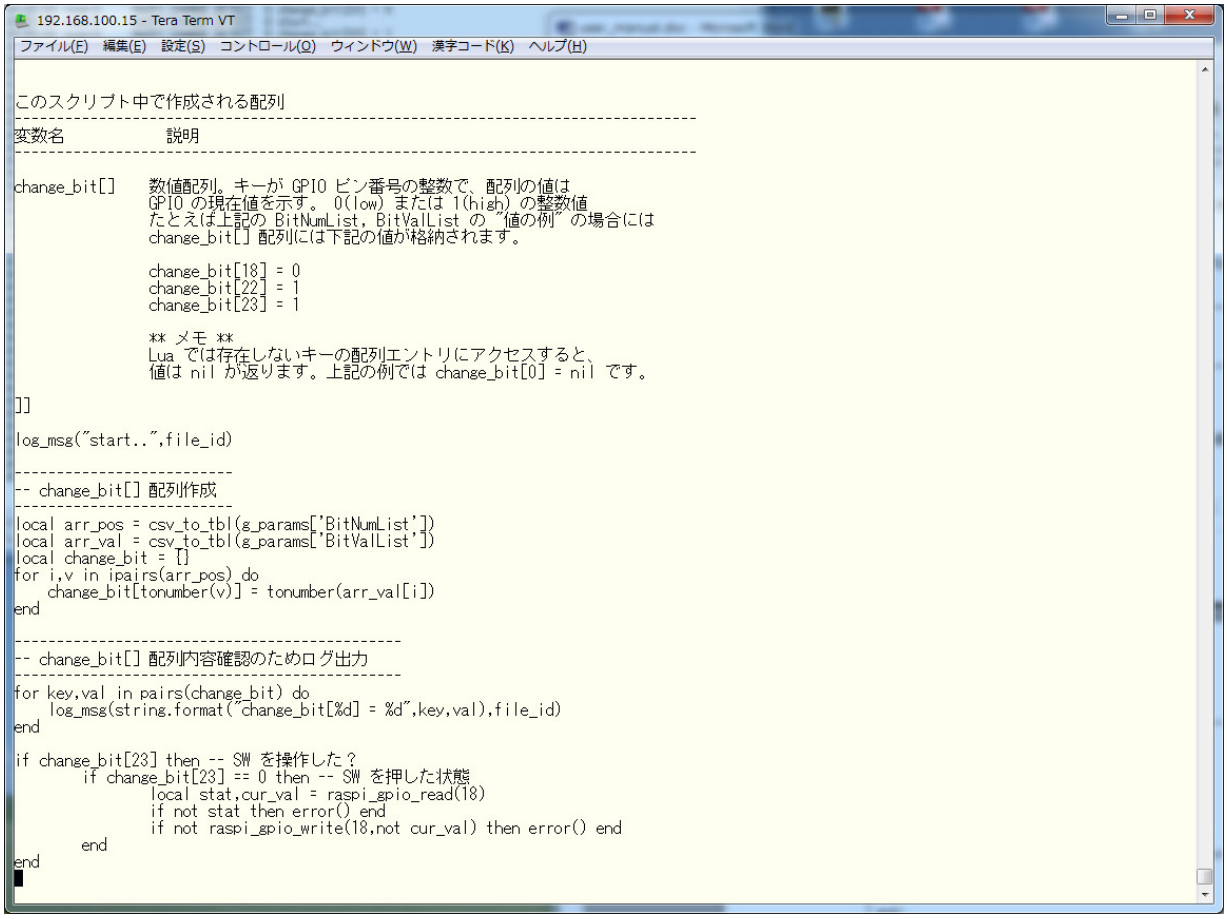
```
if change_bit[23] then -- SW を操作した？
  if change_bit[23] == 0 then -- SW を押した状態
    local stat,cur_val = raspi_gpio_read(18)
    if not stat then error() end
    if not raspi_gpio_write(18,not cur_val) then error() end
  end
end
```

最初の "if change_bit[23] then" 部分はイベントハンドラがコールされたときに、GPIO#23 が変化していたかどうかをチェックしています。RASPI_CHANGE_DETECT イベントハンドラは raspi_change_detect() ライブラリ関数で指

定された GPIO が変化した場合に、共通してコールされますのでこの文によって処理対象を絞り込みます。

次の "if change_bit[23] == 0 then" 部分では、スイッチを押し込んでいるタイミングを判断しています。タクトスイッチを操作すると、押したときと離したときの両方のタイミングでこのスクリプトがコールされます。この"if 文" によって押したときだけ続きの処理を行う様にしています。

raspi_gpio_read(18) で現在の LED(GPIO#18) の状態を読み込んでいます。その後、その値を反転させて LED(GPIO#18) に出力します。編集中の画面は下記の様になります。



編集が完了したら、agent_put コマンドで既存の RASPI_CHANGE_DETECCT スクリプトを更新します。SERVER_START とは違ってこのイベントハンドラスクリプトを有効にさせるための abs_agent 再起動は必要ありません。

```
./agent_put -s RASPI_CHANGE_DETECT -f work
```

これで設定が完了しました。タクトスイッチを押すたびに LED の ON, OFF が切り替わるのを確認できると思います。タクトスイッチ等で発生するチャタリングは、RASPI_CHANGE_DETECT イベントを検出する abs_agent 内部のロジックでフィルタリングされています。詳しくは "イベント" の章をご覧ください。

RASPI_CHANGE_DETECT イベントハンドラの内容を下記の様に変更すると、タクトスイッチを押したときに Raspberry

Pi をシャットダウンするようにできます、是非お試し下さい。

```
if change_bit[23] then -- SW を操作した？
  if change_bit[23] == 0 then -- SW を押した状態
    os.execute('/sbin/shutdown -h now &')
  end
end
```

8 リモートクライアント設定

abs_agent サーバーは別コンピュータ(Linux, Windows)に設置したクライアントプログラムからアクセスすることができます。

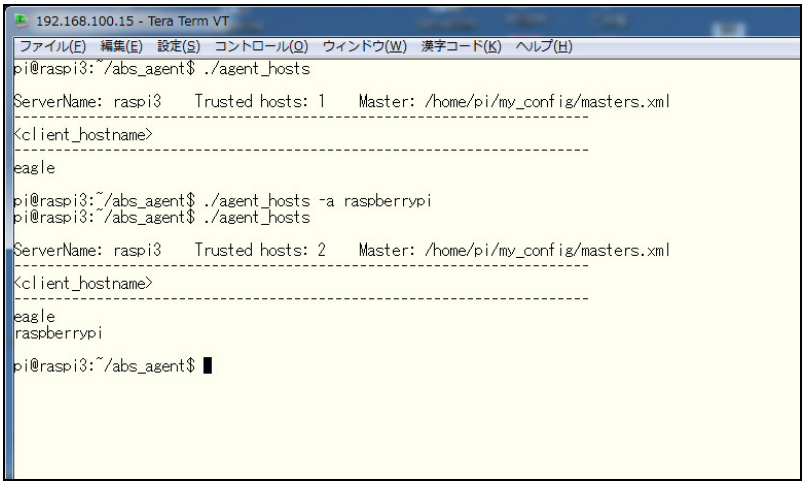
8.1 リモートクライアント設定 (Linux)

Linux 環境からリモートアクセスする場合は、abs_agent インストールキットを使用してサーバーインストール時と同様にアーカイブファイルを展開するだけで完了します。abs_agent クライアントプログラム(agent_xxxxx のファイル名の実行プログラム) のみを使用する場合には、コンフィギュレーションの設定は必要なく直ぐに使用できます。

複数のコンピュータに abs_agent サーバーを設置して、互いのサーバーにアクセスすることもできます。abs_agent のスクリプトからライブラリ関数を使用して、リモート側の abs_agent サーバーのスクリプトを実行したり、グローバル共有変数の参照や更新を行えます。リモート側アクセスする側の abs_agent の設置方法と、アクセスされる側の abs_agent サーバーのインストール作業は全く同じです。

セキュリティ確保の為、リモートアクセスされる側の abs_agent ではクライアント側(アクセスする側)のホスト名を予め登録しておく必要があります。"agent_hosts -a <hostname>" コマンドを abs_agent が動作しているコンピュータで実行すると、<hostname> で指定したコンピュータからのリモートアクセスを許可します。クライアント側のホスト名を知りたい場合は、クライアント側で Linux コマンド "hostname" を実行して下さい。別の方法として、クライアント側で"agent_stat -c" コマンドを実行すると、ホスト名と MAC アドレスを表示することができます。

"agent_hosts" コマンドではホスト名の代わりに MAC アドレスをアクセス許可に使用することもできます。MAC アドレスを認証に使用することで、アクセス許可対象のハードウエアを厳密に限定することが可能になります。詳しい "agent_hosts" コマンドの使用方法については "クライアントプログラム" の章を参照して下さい。



(ホスト名 "raspi3" で動作中の abs_agent に対して、リモートホスト名 "raspberrypi" からのリモートアクセスを許可するように agent_hosts コマンドを使用する例)

8.2 リモートクライアント設定 (Windows)

Windows PC 上で動作させている Win32 アプリケーションやエクセル VBA 等のプログラムから abs_agent にリモートアクセスすることができます。

Windows PCから abs_agent にアクセスする場合には、クライアント側の Windows PC に XASDLCMD.DLLライブラリファイルを設置しておきます。Windows PC に ABS-9000 LogServer のインストールキットを使用してインストール作業を行った場合には、XASDLCMD.DLL ライブラリファイルが自動で配置されています。

XASDLCMD.DLL ライブラリファイルを設置することでエクセル VBA や Win32プログラムから abs_agent にリモートアクセスすることができます。abs_agent のインストールキットに同梱されている XASDLCMD.DLLライブラリのバージョンは、ABS-9000 LogServer インストールキット同梱のものよりも新しいため、下記の手順で最新バージョンの XASDLCMD.DLLファイルを手動でインストール(上書きコピー)することをお勧めします。

ABS-9000 LogServer がインストールされていない Windowsコンピュータからリモートアクセスしたい場合や、最新バージョンの XASDLCMD.DLL を使用する場合には手動で XASDLCMD.DLL ファイルのインストール作業を行います。

abs_agent のインストールキットに含まれる XASDLCMD.DLL ライブラリファイルを Windows PC に手動でインストール(コピー)して下さい。abs_agent インストールキットを展開したディレクトリ "abs_agent/contrib/dll" 内にある XASDLCMD.DLL ファイルを Windows PC の下記のフォルダにネットワーク経由やその他の方法でコピー(または上書き)して下さい。Windows 上で tar + gzip形式の abs_agent インストールキット・アーカイブファイルを直接展開して XASDLCMD.DLL ファイルのみをコピー(または上書き)しても構いません。

コピーを行うファイル (abs_agent キット内のファイル名)	クライアント Windows PC のコピー先
abs_agent/contrib/DLL/XASDLCMD.dll	Windows システムディレクトリ WindowsXP,Windows2003 の場合は "C:¥Windows¥System32¥"

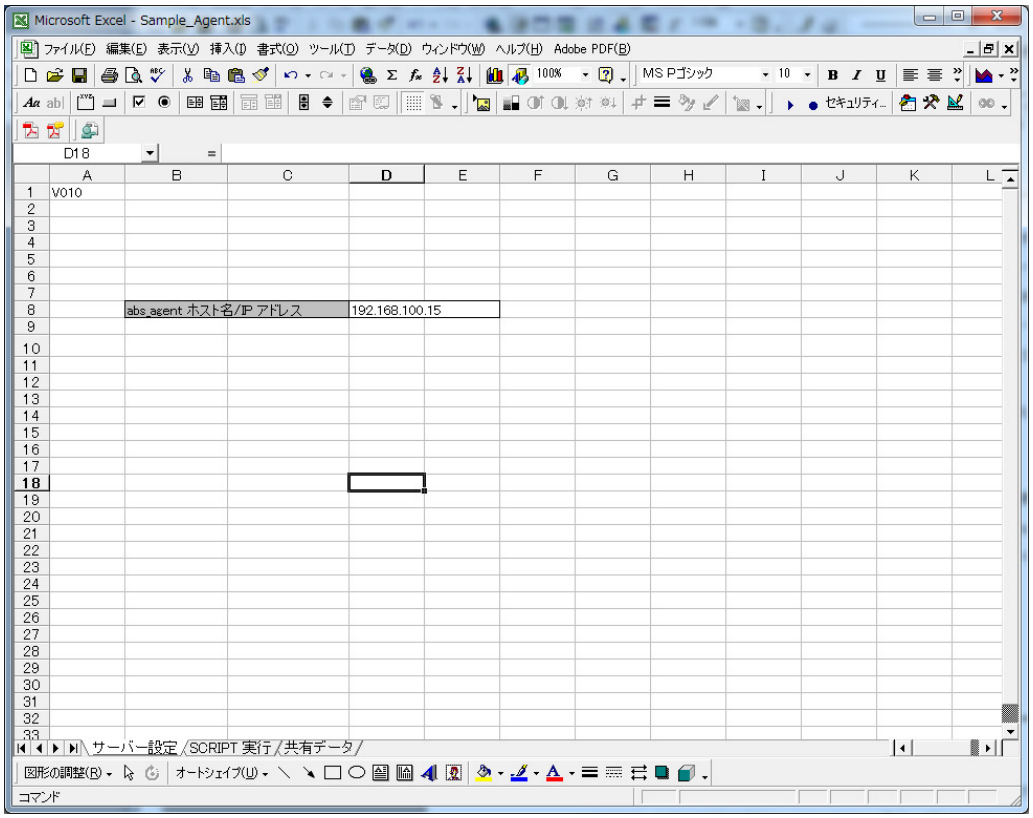
	Windows7(64bit) の場合には **"C:¥Windows¥SysWOW64"**

セキュリティ確保の為、リモートアクセスされるサーバー側の abs_agent では、クライアント WindowsPCのホスト名を予め登録しておく必要があります。abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行すると、<hostname> で指定したコンピュータからのリモートアクセスを許可します。Windows のホスト名は DOS プロンプト(コマンドプロンプト)から "hostname" コマンドで得られます。ABS-9000 LogServer をインストールしている場合には、プログラムメニューから "All Blue System" -> "CPU 情報表示" でホスト名と MACアドレスを表示することができます。

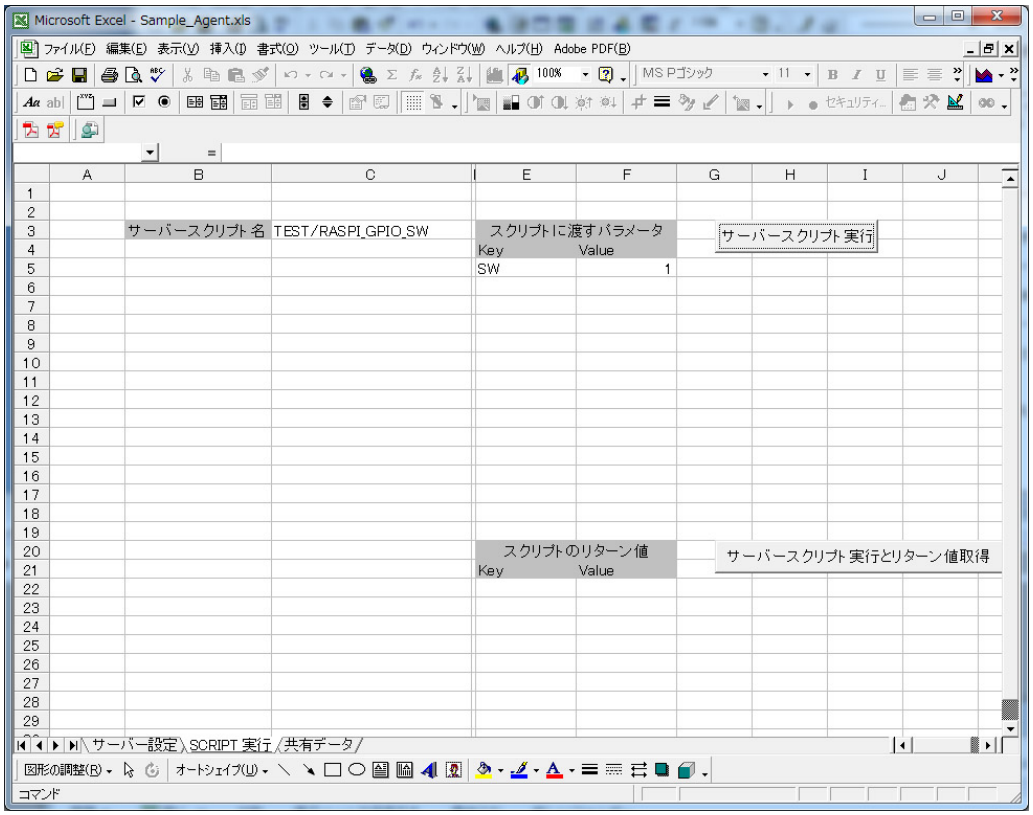
ホスト名の他に、MAC アドレスをアクセス許可に使用することもできます。詳しいコマンドの使用方法については "クライアントプログラム" の章を参照して下さい。

下記の例は、Windows PC 上のエクセルから、Raspberry Pi の GPIO#18 に接続した LED を ON または OFF に設定している例です。下記のサンプルエクセルファイル(Sample_Agent.xls) も abs_agent インストールキットに含まれていますので、Windows PC にコピーして使用できます。アーカイブ中のファイル名は "abs_agent/contrib/Excel/Sample_Agent.xls"です。Raspberry Pi 側の LED 接続方法については、"動作確認と簡単な使い方"の章に記載していますので参照して下さい。

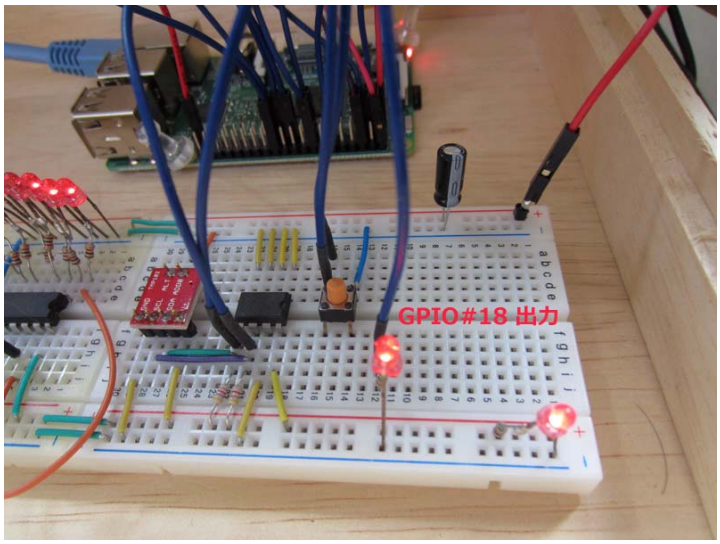
エクセルを起動して Sample_Agent.xls ファイルをオープンします。このときマクロを有効にしてください。
ワークシート画面の "サーバー設定" タブページを表示して、表示されたセルに abs_agent の IP アドレスを入力します。



"SCRIPT実行" タブを押して サーバースクリプト名欄に "TEST/RASPI_GPIO_SW" を入力します。スクリプトに渡すパラメータ欄には Key："SW", Value： "1" を入力します。その後、"サーバースクリプト実行" ボタンを押すと LED が光ります。 Value："0" にすると LED は消灯します。



Windows PC上のエクセルから操作している、Raspberry Pi GPIO#18 に接続した LED です。



リモート側 abs_agent (Raspberry Pi)で実行されるスクリプト "TEST/RASPI_GPIO_SW" の内容です。

¥¥RASPI3¥share¥my_config¥scripts¥TEST¥RASPI_GPIO_SW.lua - Notepad++

ファイル(F) 編集(E) 検索(S) 表示(V) フォーマット(M) 言語(L) 設定(T) マクロ 実行 TextFX プラグイン ウィンドウ管理 ?

RASPI_GPIO_SW.lua

```
--[[
●機能概要
GPIO に接続した LED を ON または OFF に設定する
●リクエストパラメータ
------------------------------------------------------------------------------
キー値          値                                          値の例
------------------------------------------------------------------------------
SW              出力するGPIO ポート値                       "1"
                "0" または "1"を指定
●リターンパラメータ
------------------------------------------------------------------------------
キー値          値                                          値の例
------------------------------------------------------------------------------
なし
●備考
●変更履歴
2016/07/18    初版作成
ABS-9000 DeviceServer        copyright(c) All Blue System
]]

local pin = 18  -- GPIO#18 を出力に設定

-- "SW" パラメータが指定されていない場合にはエラー
if not g_params["SW"] then error() end

-- 出力モードに設定。できれば SERVER_START スクリプト中に下記の設定を記述して一回だけ実行する方が好ましい
if not raspi_gpio_config(pin,"output","off") then error() end

if g_params["SW"] == "1" then
    raspi_gpio_write(pin,true)
else
    raspi_gpio_write(pin,false)
end
```

Lua source File | length : 1403 | lines : 47 | Ln : 1 | Col : 1 | Sel : 0 | Windows | UTF-8N | INS

9 シリアルデバイス接続

abs_agent ではシリアルデバイスから入力されたデータ毎にイベントハンドラで処理を行うことや、スクリプトやイベントハンドラ中からシリアルデバイスにデータを簡単に書き込むことができます。

abs_agent のシリアルデバイスは、設定したデバイスタイプ毎に予め決められたパケット処理を自動で行います。現在指定可能なデバイスタイプは、文字列を扱う STRING と Arduino との通信プロトコルで使用される FIRMATA、XBee 802.15.4 や XBee-ZB Zigbee 無線モジュールの API プロトコルです。

シリアルデバイスは abs_agent のサーバー設定ファイル(abs_agent.xml) に OS(Linux) 上のデバイスファイル毎にデバイスタイプや通信条件などを書き込むことで簡単に作成することができます。複数のシリアルデバイスを abs_agent に設定することができますが、指定可能な数はライセンスによって制限されます。

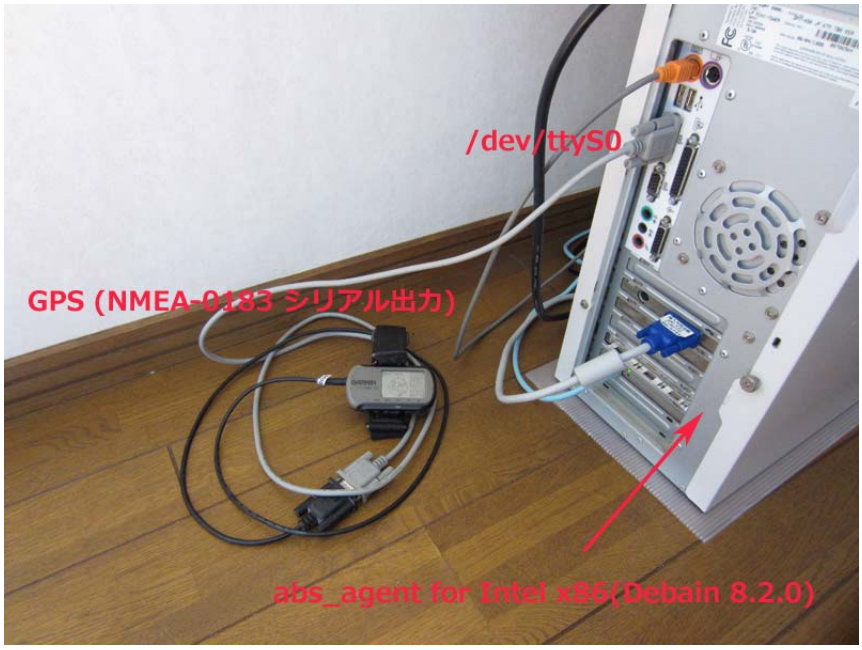
この章では GPS モジュールをシリアルデバイス経由で abs_agent に接続する例を説明します。その他のシリアルデバイスについてもデバイス登録の方法は全て同じです。デバイスタイプをデバイスの通信プロトコルに合わせたものを選択して、通信条件を設定ファイルに追加するだけで使用可能になります。

XBee 802.15.4, XBee-ZB デバイスも同様にシリアルデバイス経由で接続しますが、リモート側無線モジュールの設定やリモートデバイス・マスターの構築が必要です。詳しくは後の "XBee 802.15.4デバイス接続" や "XBee-ZB Zigbee デバイス接続" の章を参照してください。

9.1 シリアルデバイスをサーバー設定ファイルに登録

シリアルデバイスを abs_agent に接続するときには abs_agent のサーバー設定ファイル (abs_agent.xml) に、シリアルデバイス毎にエントリを登録します。シリアルポートで使用する Linux デバイスファイル名や通信条件を XML ファイルのタグとして設定ファイルに書き込みます。

ここではシリアル接続の GPS レシーバを abs_agent に登録する場合を説明します。abs_agent プログラムは Intel x86 用にビルドされたものを使用しています。これを PC/AT 互換機上で動作している Debian GNU/Linux 8.2 上で動作させます。Raspberry Pi に接続する場合も設定方法は同じですが、接続するシリアルポートのデバイスファイル名は OS 上で設定されたものに合わせてください。例えばコンソールポートに接続する場合には /dev/ttyAMA0、USB・シリアル変換器経由の場合には /dev/ttyUSB0 等になります。



(GPSレシーバとPCを接続した様子)

接続する GPS レシーバ(Garmin Foretrex101)の通信条件は下記の様になります。

GPS レシーバ通信条件	
通信フォーマット	NMEA-0183 (ASCII 文字列, CR-LF 終端、80文字以内)
通信インターフェース	RS-232C
接続するデバイスファイル	/dev/ttyS0
ボーレート	4800
ビット長	8
ストップビット	1
パリティ	無し
フロー制御	無し

サーバー設定ファイルをエディタで開いて上記のシリアルデバイスを登録します。デフォルトのサーバー設定ファイルは abs_agent をインストールしたディレクトリにファイル名 abs_agent.xml で格納されています。

```
vi abs_agent.xml
```

サーバー設定ファイル中の下記の部分を

```
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList/>
  </Serial>
```

以下の様に変更します。<DeviceList/> の部分を <DeviceList><Item> ... </Item></DeviceList>の様に変更して、<Item> .. </Item> 内にシリアルデバイス設定項目を格納します。サーバー設定ファイル中の各タグの詳細については、"サーバープログラム・設定ファイル" の章を参照してください。

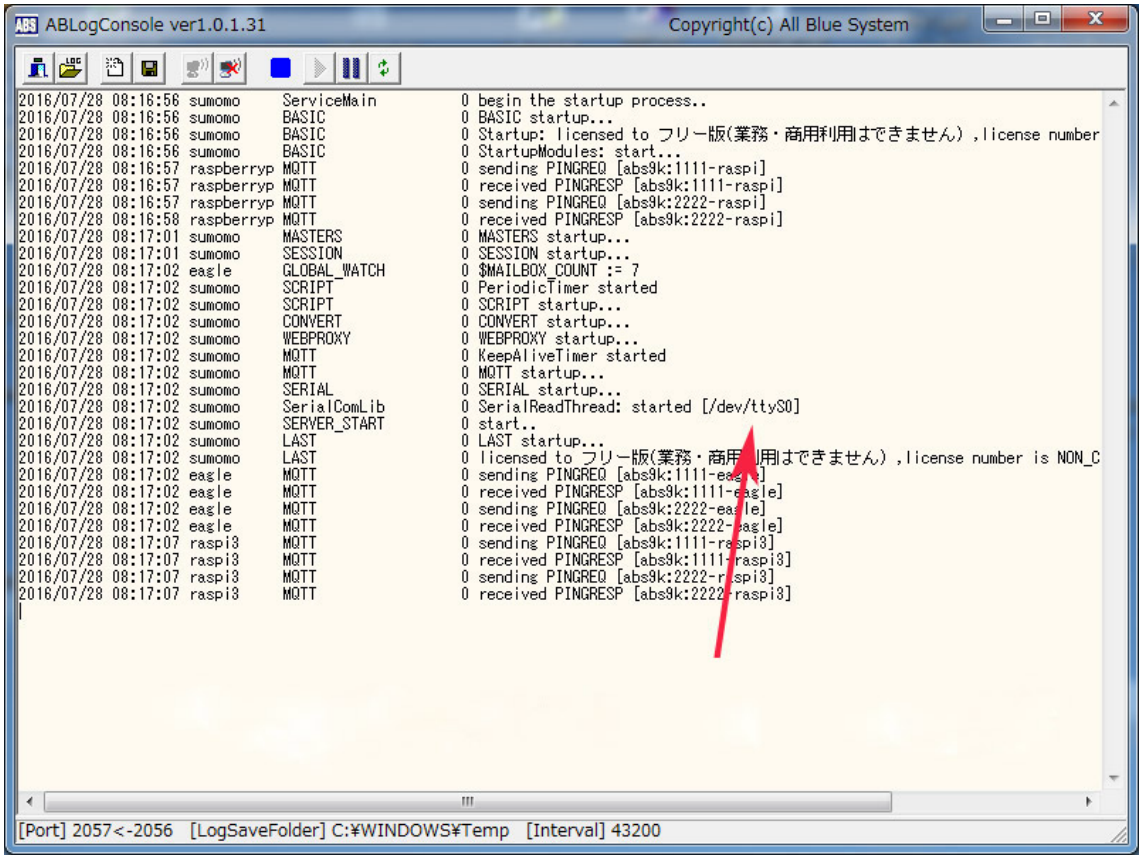
```
<Serial>
  <AutoOnline type="boolean">True</AutoOnline>
  <DeviceList>
    <Item>
      <Title>GarminGPS</Title>
      <COMPort>/dev/ttyS0</COMPort>
      <Type>STRING</Type>
      <BufferedMode>False</BufferedMode>
      <BaudRate>4800</BaudRate>
      <DataBits>8</DataBits>
      <StopBits>1</StopBits>
      <ParityBit>NONE</ParityBit>
      <FlowControl>NONE</FlowControl>
    </Item>
  </DeviceList>
</Serial>
```

XML フォーマットにエラーがあると abs_agent 起動時にエラーになりますので間違えないようにしてください。XML ファイルを編集したときには、ファイルをWebブラウザ等で読み込ませると簡単にフォーマットのエラーをチェックすることができます。

サーバー設定ファイルの編集が完了したら abs_agent を再起動させて新しいサーバー設定を有効にします。agent_shutdown 停止コマンドを実行して、完全にサーバーが停止するまで 10 秒程度待った後に abs_agent を起動しています。もし、abs_agent の自動起動を /etc/rc.local 等に設定している場合には OS を再起動する方法でも構いません。

```
./agent_shutdown
sudo ./abs_agent -I 192.168.100.45
```

ログコンソールを起動していると、サーバー起動時に登録したシリアルデバイス毎のスレッド起動メッセージが下記の様に表示されます。



9.2 GPS レシーバ接続

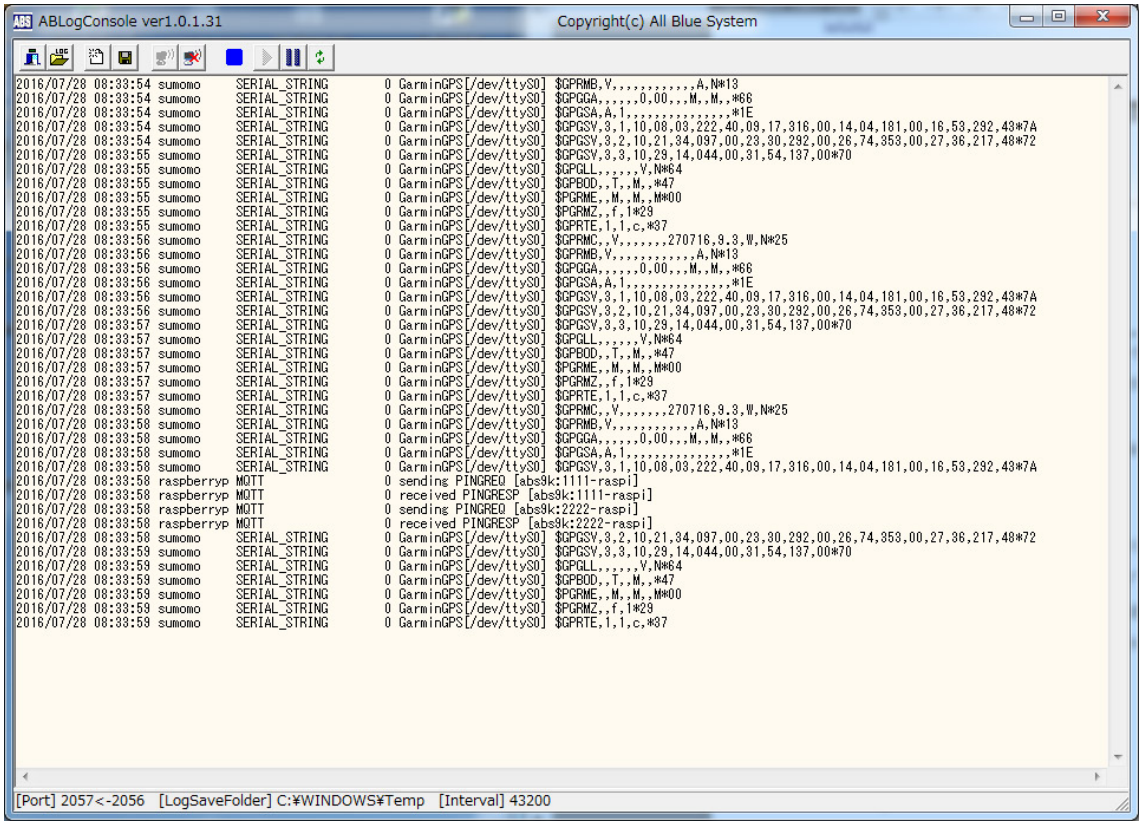
GPSレシーバや汎用のGPS モジュールは、測位データを定期的にシリアルポートに出力する機能が付いていますので、これを abs_agent のシリアルデバイスで取り込みます。測位データは NMEA-0183 フォーマットで出力されます。これは CR-LF 終端の文字列データなので簡単に abs_agent に取り込むことができます。

```
$GPRMC,044924,A,4254.1695,N,14135.6248,E,0.0,0.0,240716,9.3,W,A*0C
$GPRMB,A,,,,,,,,,,,,A,A*0B
$GPGGA,044924,4254.1695,N,14135.6248,E,1,03,3.8,10.7,M,30.7,M,,*71
$GPGSA,A,2,01,,07,,,,,,,,,30,3.9,3.8,1.0*36
$GPGSV,3,1,12,01,76,238,42,03,18,172,00,07,19,228,44,08,53,079,00*7E
$GPGSV,3,2,12,10,13,038,00,11,83,346,00,16,01,144,00,17,07,284,00*71
$GPGSV,3,3,12,22,35,154,00,27,19,093,00,28,37,313,00,30,27,255,44*70
$GPGLL,4254.1695,N,14135.6248,E,044924,A,A*4E
$GPBOD,,T,,M,,*47
$PGRME,49.2,M,50.0,M,70.1,M*12
$PGRMZ,35,f,2*2C
$GPRTE,1,1,c,*37
```

(GPS レシーバから送信されるカンマ区切りの測位データ例で、第一カラムが NMEAセンテンス種別($xxxxx の文字列)です。GPS レシーバーの仕様で設定された間隔(1Hzなど)で、複数のセンテンス種別の測位データがまとめて送ら

れてきます)

GPS レシーバの電源を入れるとシリアルデバイスから読み込んだ GPS データがログコンソールに出力されます。



サーバー設定ファイル中のシリアルデバイス設定部分で "<Type>STRING</Type>" と "<BufferedMode>False</BufferedMode>" を指定しましたので、シリアルデバイスから入力されたデータは文字列として文字終端(Null, CR, CR-LF の何れか)まで読み込んだ後、１文字列ごとに SERIAL_STRING イベントハンドラがコールされます。このとき、デフォルトの SERIAL_STRING イベントハンドラ中では文字列を単にログ出力するように設定されています。

送信されてくる NMEA-0183 フォーマットのデータは、カンマ区切り形式の文字列データで、１文字列に１つのセンテンスデータが格納されます。センテンスデータ中の第一カラムが、測位データの種類を表すセンテンス種別('$' + TalkerID + SentenceID) です。今回は、このセンテンス種別をキー名としてグローバル共有変数に測位データを格納します。上記のデータ例では、合計１２センテンス分の測位データを、１２個のグローバル共有変数に分けて格納します。GPSレシーバからは最新のセンテンスデータが繰り返し送信されますので、グローバル共有変数に格納される GPS データは常に最新の状態で更新されます。

SERAIL_STRING イベントハンドラを修正します。イベントハンドラスクリプトを作業用の "work" ファイルにダウンロードした後、エディタで編集します。

```
./agent_get -s SERIAL_STRING -f work
vi work
```

SERIAL_STRING イベントハンドラの最後の部分にステートメントを追加して下記の様に変更します。

```
file_id = "SERIAL_STRING"
--[[
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
SERIAL_STRING スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値          値                                              値の例
-------------------------------------------------------------------------------
COMPort         イベントを送信したシリアルデバイスのデバイスファイル名  "/dev/ttyUSB0"
Title           イベントを送信したシリアルデバイスのタイトル名          "コントローラ#1"
                タイトルが設定されていない場合にはこのパラメータは
                設定されません
STRING          シリアルデバイスから入力された文字列                    "string data"
                文字列は、下記の何れかのデータで終端されたものです。
                ヌル文字(0x00),CR(0x0D),LF(0x0A),CR-LF(0x0D,0x0A)
                STRING パラメータには、終端文字を含まない文字列部分が格納されています
]]
-------------------------------------------------
-- シリアルデバイスから読み込んだ文字列をログに出力
-------------------------------------------------
log_msg(g_params["Title"] .. "[" .. g_params["COMPort"] .. "] " .. g_params["STRING"],file_id)


-- NMEA-0183 メッセージをグローバル共有変数に格納する。 第一カラムの NMEA センテンス名をキー名とする
if string.sub(g_params["STRING"],1,1) == '$' then
          local arr = csv_to_tbl(g_params["STRING"])
          if not set_shared_data(arr[1],g_params["STRING"]) then error() end
end
```

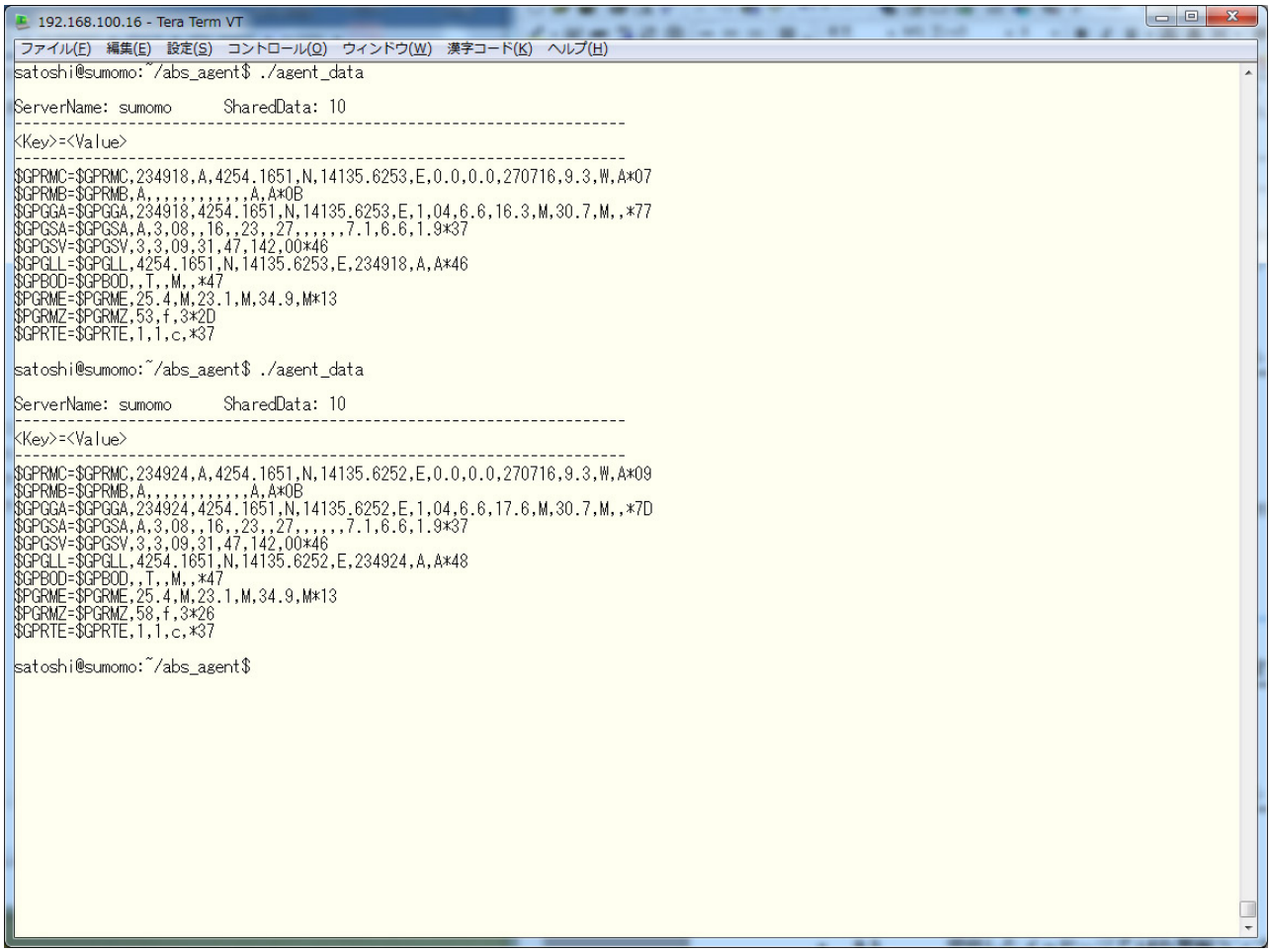
編集が完了したら、既存の SERIAL_STRING イベントハンドラを更新します。

```
./agent_put -s SERIAL_STRING -f work
```

これで GPS データからセンテンスを受信すると最新の GPS センテンスがグローバル共有変数に格納されるようになりました。abs_agent では、グローバル共有変数に格納された GPS データはユーザースクリプトやイベントハンドラから常に使用できます。agent_data クライアントプログラムを使用するとグローバル共有変数の内容を簡単に参照したり変更することができます。下記のプログラムを実行してみてください。

```
./agent_data
```

コンソールには最新の NMEA-0183 センテンスが格納されたグローバル共有変数が表示されます。後の項では、この GPS データから現在地の座標データと取得日時を得るための簡単なユーザースクリプトも作成してみます。



9.3 最新のGPS データから座標を計算

ここまでの設定で、GPS レシーバーから送信される最新の測位データをグローバル共有変数で取り出せるようになりました。このデータを元に座標データを取り出す簡単なスクリプトを作成したいと思います。

NMEA-0183 センテンスで出力される座標データは DMM 形式のフォーマットになっています。このデータ形式を Web API 等で用いられる一般的な Degree(度) フォーマットに変換するためのスクリプトを作成します。同時に GPS の時刻データが UTCフォーマットのため使い勝手が悪いので、これをローカル日時(YYYY/MM/DD HH:MM:SS)に変換します。

スクリプトは下記の様になります。このスクリプトはインストールキットに同梱されていますので作成する必要はありません。スクリプト名は TEST/GPS_REPORT です。

```
--[[
●機能概要
グローバル共有変数に格納されている GPS NMEA-0183 センテンスデータから
現在地の緯度と経度、データ取得時刻を求める。
●参照するグローバル共有データ
--------------------------------------------------------------------------------
キー値                          値/値の例
--------------------------------------------------------------------------------
$GPRMC     GPSレシーバから受信した最新の NMEA-0183 RMC センテンスが格納されている
           "$GPRMC,050246,A,4254.1621,N,14135.6146,E,0.0,30.6,240716,9.3,W,A*31"
$GPGGA     GPSレシーバから受信した最新の NMEA-0183 GGA センテンスが格納されている
           "$GPGGA,050246,4254.1621,N,14135.6146,E,1,05,2.7,30.6,M,30.7,M,,*72"
●リターンパラメータ
--------------------------------------------------------------------------------
キー値                          値                              値の例
--------------------------------------------------------------------------------
status     処理が成功した場合には "OK" が格納される
           エラーが発生した場合にはエラーメッセージが格納される "OK"
lat        Degree 形式の緯度を設定                             "35.680743"
lon        Degree 形式の経度を設定                             "139.768914"
timestamp  GPS データ中のローカル日時を設定                    "2016/02/01 23:22:10"
]]
--------------------------------------------------------------------
-- GPSレシーバから受信した NMEA-0183 センテンスが保存済みか
--------------------------------------------------------------------
local stat,gps_rmc,gps_gga
stat,gps_rmc = get_shared_data("$GPRMC")
stat,gps_gga = get_shared_data("$GPGGA")
if (gps_rmc == "") or (gps_gga == "") then
  script_result(g_taskid,"status","GPSデータが保存されていません")
  return
end
--------------------------------------------------------------------
-- NMEA センテンス中の使用するカラムをローカル変数にコピーする
--------------------------------------------------------------------
```

```
local rmc_arr = csv_to_tbl(gps_rmc)
local gga_arr = csv_to_tbl(gps_gga)
local rmc_time = rmc_arr[2]
local rmc_stat = rmc_arr[3]
local rmc_lat = rmc_arr[4]
local rmc_lat_compass = rmc_arr[5]
local rmc_lon = rmc_arr[6]
local rmc_lon_compass = rmc_arr[7]
local rmc_date = rmc_arr[10]
local gga_alt = gga_arr[10]
--------------------------------------------------------------------------
-- GPS レシーバーステータスが有効な場合のみ処理を行う
--------------------------------------------------------------------------
if rmc_stat ~= "A" then
  script_result(g_taskid,"status","レシーバステータスが有効ではありません")
  return
end
--------------------------------------------------------------------------
-- GPSデータからローカル日時を計算
--------------------------------------------------------------------------
local local_date,local_time
stat, local_date, local_time = gps_utc_to_local(rmc_date,rmc_time)
if not stat then
  script_result(g_taskid,"status","GPSタイムスタンプ変換エラー")
  return
end
script_result(g_taskid,"timestamp",local_date .. " " .. local_time)
--------------------------------------------------------------------------
-- GPSデータのローカル時刻はサーバーと比べて10分以上ずれていないか
--------------------------------------------------------------------------
local now = os.date "*t"
local seconds, gps_year, gps_month, gps_day, gps_hour, gps_min, gps_sec
stat, gps_year, gps_month, gps_day, gps_hour, gps_min, gps_sec = str_to_datetime(local_date .. " " ..
local_time)
stat, seconds = seconds_between(now["year"],now["month"],now["day"],now["hour"],now["min"],now["sec"],
          gps_year, gps_month, gps_day, gps_hour, gps_min, gps_sec)
if math.abs(seconds) > 600 then
  script_result(g_taskid,"status","最新のGPSデータを受信していません")
  return
```

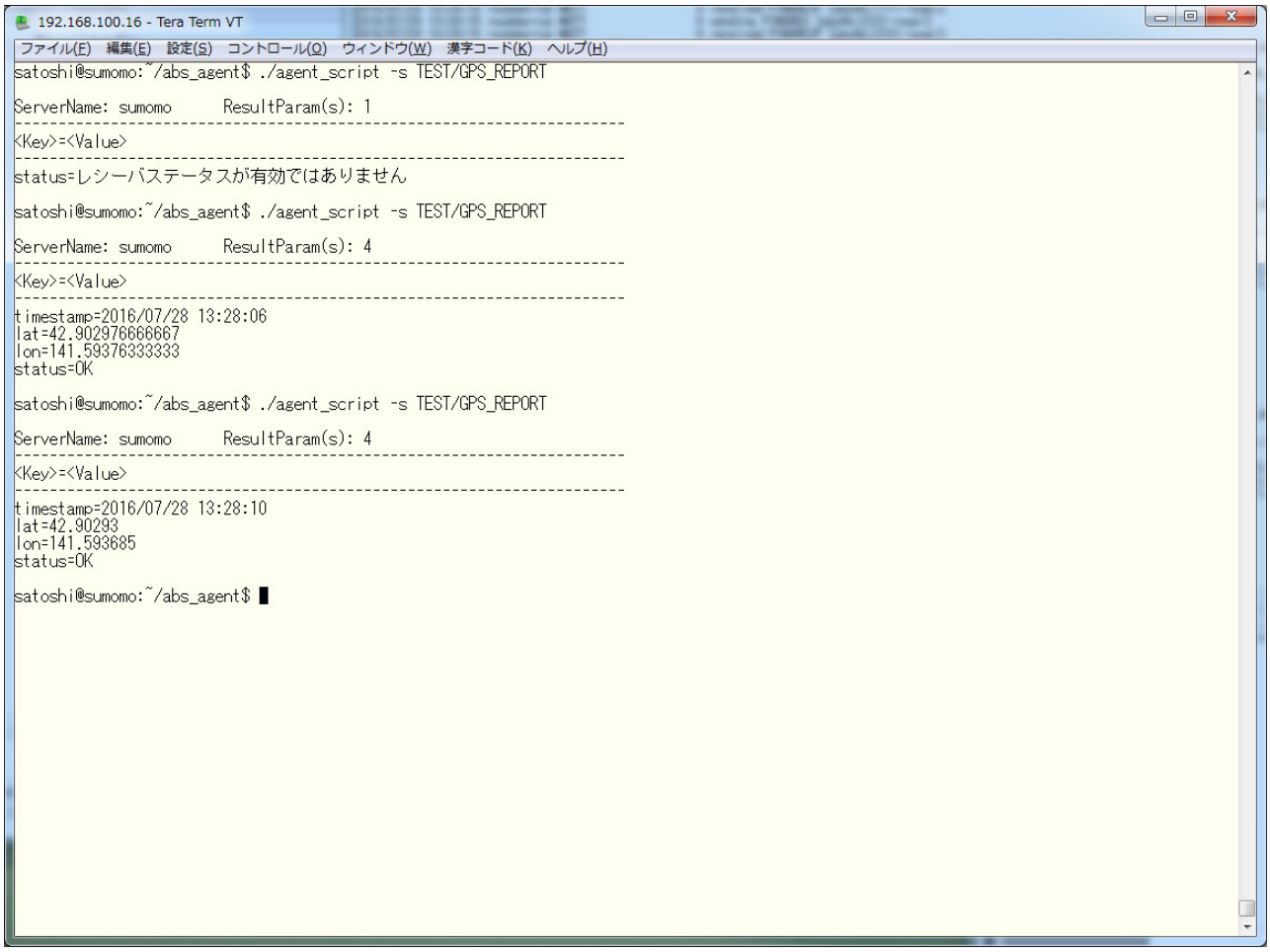
```
end
--------------------------------------------------------------------------
-- GPSデータの座標単位を Degree形式に変換する
--------------------------------------------------------------------------
local lat_deg,lon_deg
stat,lat_deg,lon_deg = gps_coordinate_deg("dmm",rmc_lat,rmc_lon,rmc_lat_compass,rmc_lon_compass)
if not stat then
  script_result(g_taskid,"status","座標単位変換エラー")
  return
end
script_result(g_taskid,"lat",lat_deg)
script_result(g_taskid,"lon",lon_deg)
script_result(g_taskid,"status","OK")
```

TEST/GPS_REPORT スクリプトではグローバル共有変数に格納されている "$GPGGA" と "$GPRMC" の２つのセンテンス中のデータを使用しています。データの内容をチェックしたのち、座標データと時刻データ部分を変換してリターンパラメータに格納しています。処理の詳細はスクリプトをご覧ください。

上記のスクリプトをクライアントプログラムから実行します。

```
./agent_script -s TEST/GPS_REPORT
```

実行結果は下記の様になります。レシーバの電源を入れた直後から、電波を受信して測位するまでの間はエラーメッセージを返します。測位後はスクリプトリターンパラメータに最新の座標データとGPS日時が得られます。



10 MQTT ブローカ接続設定

abs_agent では LAN やインターネット上に設置された MQTT ブローカに接続する機能を提供しています。

MQTT ブローカに対してスクリプトやイベントハンドラ中からセンサーデータ等のメッセージを送信することができます。また、MQTT ブローカに購読リクエストを送信して、メッセージの配信を受けることができます。MQTT ブローカから配信メッセージを受信すると、abs_agent 側で MQTT_PUBLISH イベントハンドラが実行されます。ユーザーはこのイベントハンドラ(Lua スクリプト)をカストマイズしてメッセージを処理することができます。

この章では MQTT ブローカ自身のセットアップ方法については説明していません。MQTT ブローカはインターネット上で公開されているフリーや商用のものが利用できます。また、オープンソースで開発されている Mosquitto[7] を使用すると、abs_agent が動作している Raspberry Pi 上に設置できますので便利です。詳しい設置方法は MQTT ブローカ提供元のドキュメントをご覧ください。

abs_agent では MQTT ブローカへの接続はエンドポイントと呼ばれる単位で管理しています。エンドポイントには MQTT ブローカに接続するためのホスト名やポート番号、認証に使用するユーザー名やパスワードなどを定義します。

[7] Mosquitto (https://mosquitto.org/)

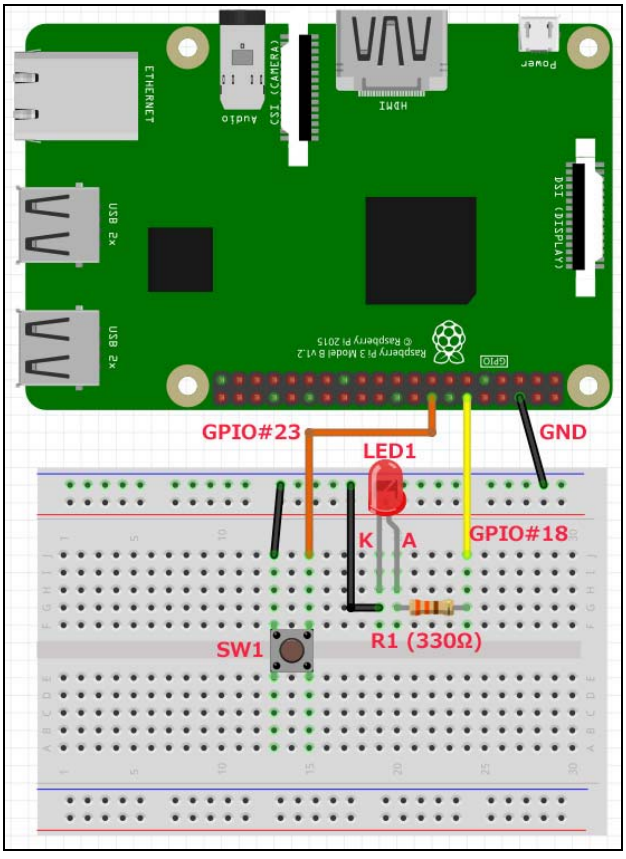
エンドポイントを設定すると、abs_agent は起動時に自動的に MQTT ブローカに接続します。また、エンドポイントに設定した KeepAliveTimer（秒）ごとに、ブローカへ PINGREQ メッセージの送信を自動で行います。MQTT ブローカ間のソケット通信時エラーを検出すると、再接続動作が自動で実行されます。このため、ユーザーは MQTT への配信と受信時のスクリプトやイベントハンドラを記述するだけで簡単に MQTT ブローカを利用することが出来ます。

MQTT エンドポイントは abs_agent のサーバー設定ファイル(abs_agent.xml）にテキストエディタを使用して簡単に作成することができます。複数のエンドポイントを abs_agent に設定することができますが、指定可能な数はライセンスによって制限されます。

10.1 MQTT エンドポイントをサーバー設定ファイルに登録

MQTT エンドポイント abs_agent に接続するときには abs_agent のサーバー設定ファイル（abs_agent.xml）に、接続毎にエントリを登録します。１つのエンドポイントで複数のトピックを購読するように設定したり、同一の MQTT ブローカへ接続するためのエンドポイントを複数作成することもできます。MQTT ブローカへ送信する場合のエンドポイントも購読用に作成したエンドポイントを使用したり、送信用に専用のエンドポイントを別に作成することもできます。

このマニュアルでは "動作確認と簡単な使い方"の章で紹介した、LED と スイッチを Raspberry Pi に接続したときの回路を使用します。スイッチを操作したときに MQTT ブローカにメッセージを送信する配信例と、MQTT ブローカで購読しているメッセージを受信して LED を操作する２つの例を紹介します。



(使用する回路。MQTT ブローカに送信するイベント発生用のスイッチと、受信したメッセージを確認する LED として使用します)

今回は MQTT ブローカへ接続するためのエンドポイントを１つ作成します。 abs_agent プログラムは Raspberry Pi 用にビルドされたものを使用します。また、MQTT ブローカーは abs_agent が動作している Raspberry Pi に Mosquitto をインストールしています。

MQTT ブローカ (Mosquitto) の動作パラメータは以下になります。使用する MQTT ブローカに合わせて下記のパラメータを適宜読み替えて下さい。

MQTT ブローカ(mosquitto) 動作パラメータ	
ホスト名または IP アドレス	192.168.100.14
ポート番号	1883
MQTT 接続時ユーザー名	なし
MQTT 接続時パスワード	なし

上記の MQTT ブローカに接続用の abs_agent 側エンドポイントの設定は下記になります。

MQTT エンドポイント設定	
エンドポイントタイトル(任意の文字列)	試験用MQTT接続
ClientID(同一MQTTブローカ内でユニークな文字列)	test_endpoint_12345
MQTT ブローカホスト名または IP アドレス	192.168.100.14
MQTT ブローカポート番号	1883
KeepAliveTimer (秒)	60
起動時に自動で購読するトピック	/+/switch
起動時に自動で購読するトピックのQoS	0
CONNECT ユーザー名	なし
CONNECT パスワード	なし
Will トピック	なし
Will メッセージ	なし
Will QoS	0 (使用しませんが便宜上 0 を指定します)
Will Retain	False
受信バッファ初期値	0
詳細ログ出力	True

これらのMQTT エンドポイント設定は XML ファイルのタグとして abs_agent サーバー設定ファイルに書き込みます。サーバー設定ファイルをエディタで開いて上記のエンドポイントを登録します。デフォルトのサーバー設定ファイルは abs_agent をインストールしたディレクトリにファイル名 abs_agent.xml で格納されています。

```
vi abs_agent.xml
```

サーバー設定ファイル中の下記の部分を

```
<MQTT>
  <AutoOnline type="boolean"></AutoOnline>
  <KeepAliveTimer type="integer"></KeepAliveTimer>
  <EndPointList/>
</MQTT>
```

以下の様に変更します。<EndPointList/> の部分を <EndPointList><Item> ... </Item></EndPointList>の様に変更して、<Item> .. </Item> 内にエンドポイント設定項目を格納します。サーバー設定ファイル中の各タグの詳細については、"サーバープログラム" の章を参照してください。

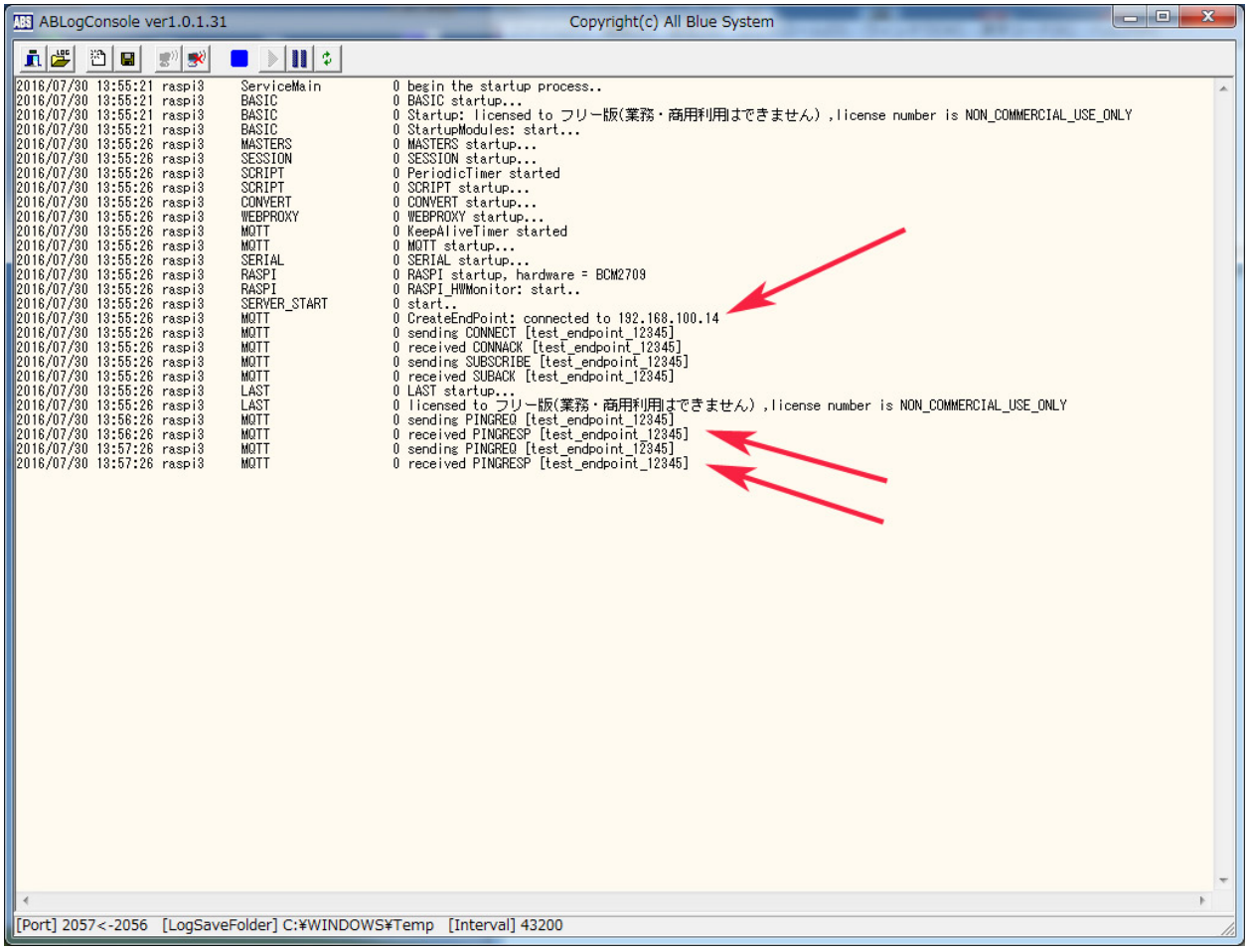
```
<MQTT>
  <AutoOnline type="boolean">True</AutoOnline>
  <KeepAliveTimer type="integer">60</KeepAliveTimer>
  <EndPointList>
    <Item>
      <Title>試験用MQTT接続</Title>
      <ClientID>test_endpoint_12345</ClientID>
      <BrokerHostName>192.168.100.14</BrokerHostName>
      <PortNumber>1883</PortNumber>
      <AutoSubscribeTopicList>/+/switch</AutoSubscribeTopicList>
      <AutoSubscribeQoSList>0</AutoSubscribeQoSList>
      <UserName/>
      <Password/>
      <WillTopic/>
      <WillMessage/>
      <WillQoS>0</WillQoS>
      <WillRetain>False</WillRetain>
      <RecvBuffInit>0</RecvBuffInit>
      <DetailLog>True</DetailLog>
    </Item>
  </EndPointList>
</MQTT>
```

XML フォーマットにエラーがあると abs_agent 起動時にエラーになりますので間違えないようにしてください。XML ファイルを編集したときには、ファイルをWebブラウザ等で読み込ませると簡単にフォーマットのエラーをチェックすることができます。

サーバー設定ファイルの編集が完了したら abs_agent を再起動させて新しいサーバー設定を有効にします。agent_shutdown 停止コマンドを実行して、完全にサーバーが停止するまで 10 秒程度待った後に abs_agent を起動しています。もし、abs_agent の自動起動を /etc/rc.local 等に設定している場合には OS を再起動する方法でも構いません。

```
./agent_shutdown
sudo ./abs_agent -l 192.168.100.45
```

ログコンソールを起動していると、サーバー起動時に登録したエンドポイント毎に MQTT ブローカに接続したときのログが記録されます。また、KeepAliveTimer で設定した間隔(60秒)毎に PINGREQ メッセージを MQTT ブローカに送信している様子も確認できます。



10.2 スイッチ操作でデータ送信

スイッチを操作したときに、その状態を MQTT ブローカにメッセージとして送信する例を作成します。"動作確認と簡単な使い方"の章で説明した様に、abs_agent サーバー起動時にスイッチ入力と LED 出力用に Raspberry Pi の GPIO モードを予め設定する必要があります。SERVER_START イベントハンドラに下記の様な記述がされていることを確認してください。詳しい設定方法は前の章の説明部分をご覧ください。

```
-- GPIO#18 LED出力, GPIO#23 スイッチ入力設定
if not raspi_gpio_config(18,"output","off") then error() end
if not raspi_gpio_config(23,"input","pullup") then error() end
if not raspi_change_detect(23,true) then error() end
```

(SERVER_START イベントハンドラ中に記述したスイッチとLED接続用の GPIO 設定例)

次に、スイッチを操作して GPIO値が変化したときに実行される RASPI_CHANGE_DETECT イベントハンドラを編集してスイッチ操作時に MQTT ブローカにスイッチの状態をメッセージとして送信するようにします。agent_get コマンドで RASPI_CHANGE_DETECT スクリプトをダウンロードして編集します。

```
./agent_get -s RASPI_CHANGE_DETECT -f work
vi work
```

スクリプト後半部分に下記の様な構文を追加します。先の章で LED 出力のための記述がされていた場合には一部内容を編集だけで簡単に記述できます。

```
if change_bit[23] then -- SW を操作した？
  local msg
  local clientid = "試験用MQTT接続"
  local topic = "/" .. g_hostname .. "/switch"
  if change_bit[23] == 0 then
    msg = "ON"
  else
    msg = "OFF"
  end
  if not mqtt_publish(clientid,topic,msg,0) then error() end -- MQTTブローカに SW の状態を送信
end
```

最初の "if change_bit[23] then" 部分はイベントハンドラがコールされたときに、GPIO#23 が変化していたかどうかをチェックしています。

次の "msg" はMQTT ブローカに送信するメッセージを格納する変数です。ここにスイッチが押した状態の場合には "ON" が入り、離した状態のときには "OFF" が入ります。

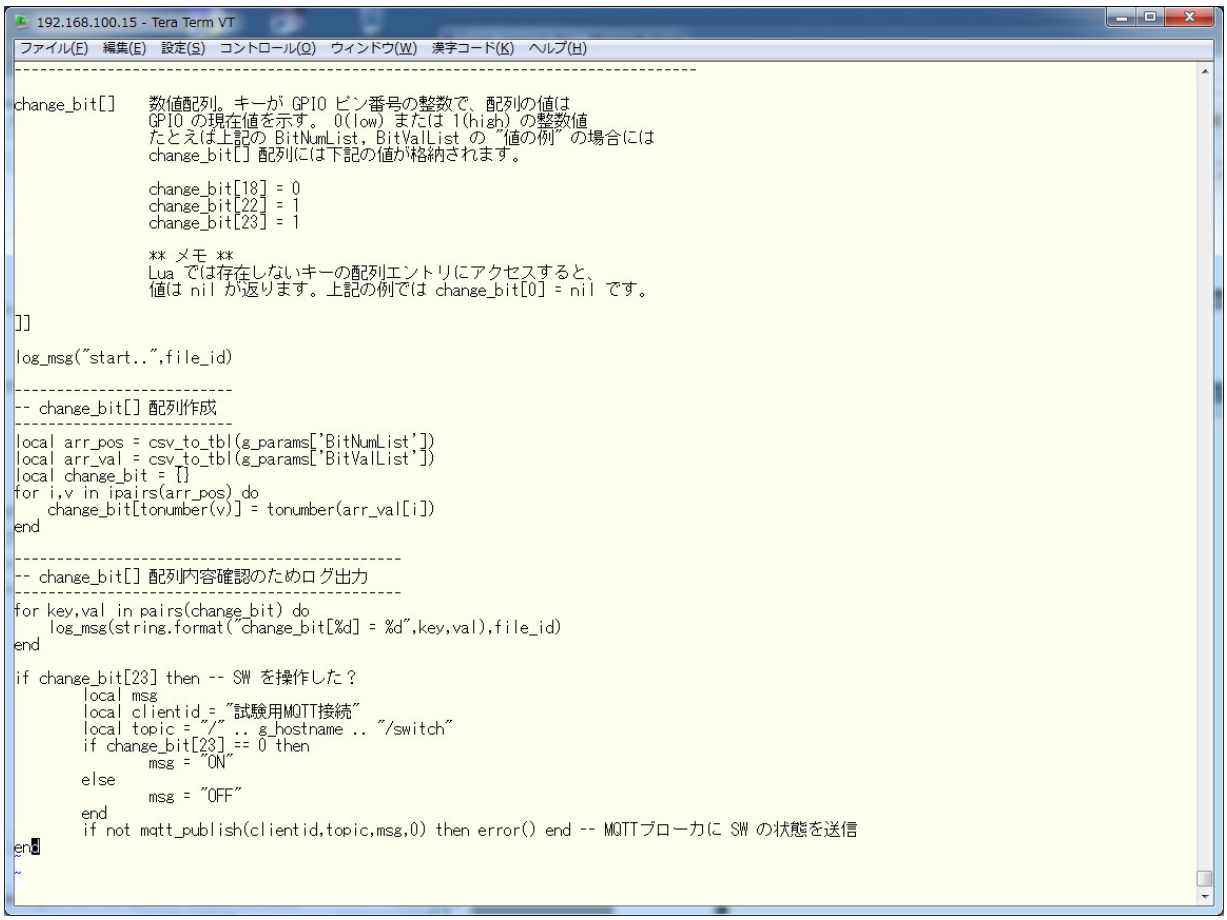
"clientid" はサーバー設定ファイル中に指定した MQTTエンドポイントの <ClientID> または <Title> のどちらかの内容を指定します。ここでは <Title> 項目に設定した "試験用MQTT接続"を指定しています。

"topic" はメッセージを送信するときのトピック文字列です。今回は "/<hostname>/switch" の様な文字列を指定し

ます。<hostname> 部分にはMQTT クライアント側ホスト名、つまりは abs_agent が動作しているホスト名を入れます。

mqtt_publish() ライブラリ関数をコールすると、指定したエンドポイント経由で MQTT ブローカにメッセージを送信します。フリー版ライセンスで指定可能な QoS は 0 のみですので第四パラメータには "0" を指定します。

編集中の画面は下記の様になります。



編集が完了したら、agent_put コマンドで既存の RASPI_CHANGE_DETECCT スクリプトを更新します。このとき、abs_agent 再起動は必要はありません。

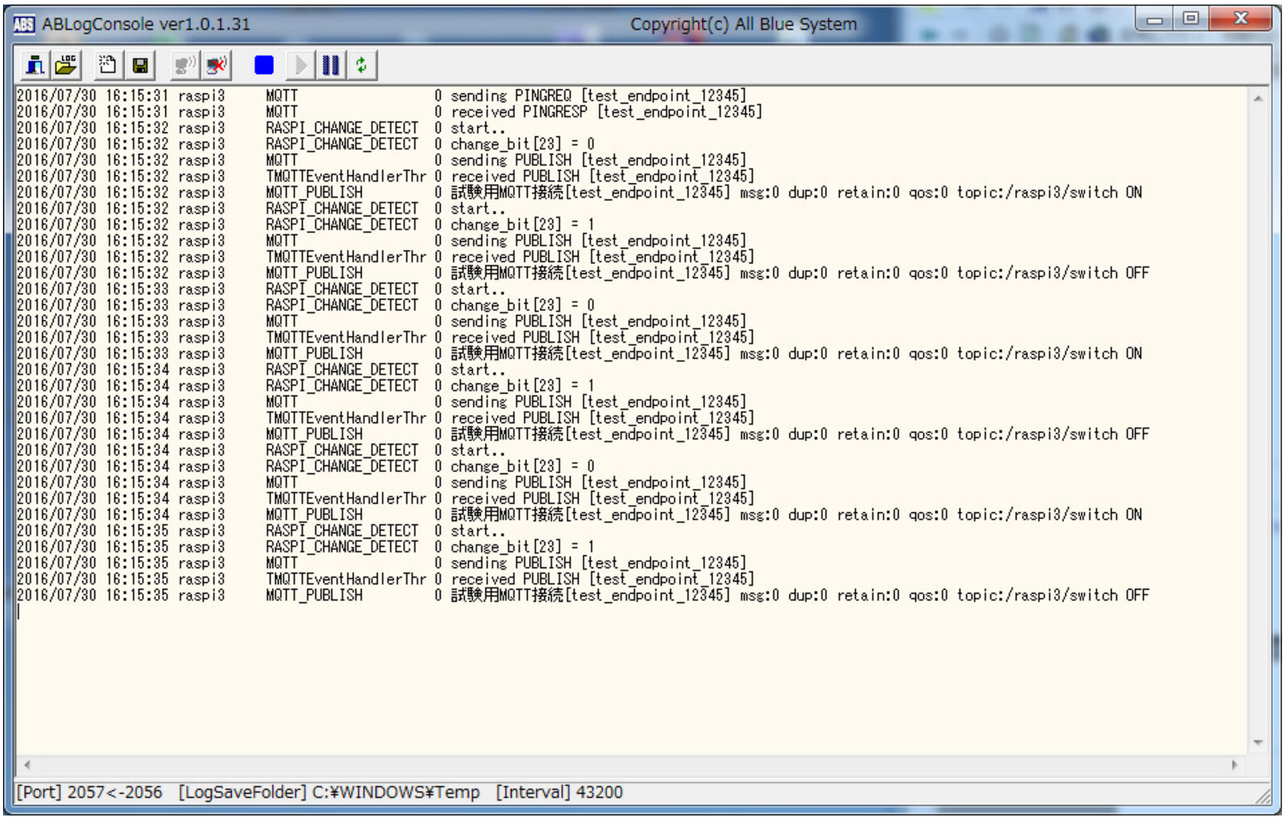
```
./agent_put -s RASPI_CHANGE_DETECT -f work
```

スイッチを操作してみてください。このときログコンソールには下記の様なメッセージが記録されます。スイッチを押した瞬間と離した瞬間それぞれで、MQTT ブローカにメッセージを送信している様子が判ります。

サーバー設定ファイルに MQTT エンドポイント設定を記述した時に、MQTTブローカへの接続と同時に "/<任意の文字列>/switch" のトピックを購読するように指定しています。このため、MQTT ブローカにメッセージを送信した直後に、MQTT ブローカからは同一のメッセージが配信されてきて、ログにはこの様子も記録されています。MQTTブロ

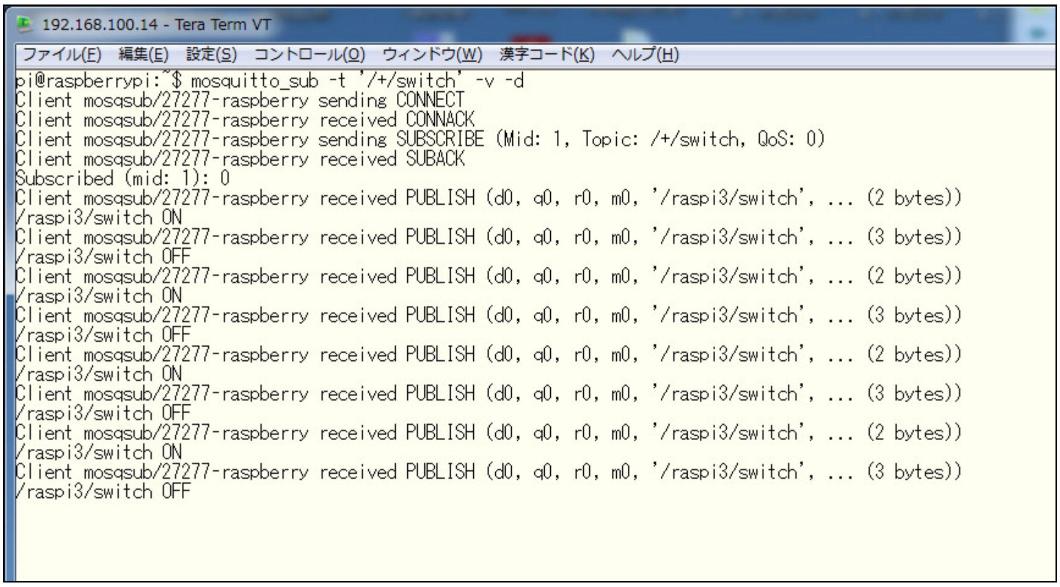
一カからメッセージを受信（配信を受ける）と MQTT_PUBLISH イベントハンドラが実行されますが、デフォルトでは

パケットデータを単にログに出力する内容になっています。



この時、別のコンソール端末から mosquitto_sub クライアントプログラムを使用して配信メッセージを受信してい

ると下記の様に表示されます。



10.3　受信したメッセージでLED点灯

次は、MQTT ブローカから配信されてくるスイッチの状態を知らせるメッセージを解析して、Raspberry Pi に接続した LED を点灯または消灯させます。

MQTT ブローカへの購読リクエストは、MQTTエンドポイント設定中の下記項目で設定しています。

```
<AutoSubscribeTopicList>/+/switch</AutoSubscribeTopicList>
<AutoSubscribeQoSList>0</AutoSubscribeQoSList>
```

MQTT ブローカへ接続したときに "/+/switch" トピックの購読リクエストも自動で送信しています。このトピック名は先のスイッチ操作を行ったときに MQTT ブローカにメッセージを送信するときに使用したトピック名 "/<hostname>/switch" にマッチするように指定します。購読リクエストのトピック名中の "+" 部分は任意の文字列に対応しますので、複数のコンピュータから送信されたスイッチ入力のメッセージを全て購読することができます。

abs_agent では MQTT ブローカから購読中のトピックに対応するメッセージ(MQTT-PUBLISHパケット)を受信すると、自動的に MQTT_PUBLISH イベントハンドラが実行されます。今回はこのイベントハンドラを編集して LED の ON, OFF を切り替えるようにします。agent_get コマンドで MQTT_PUBLISH スクリプトをダウンロードして編集します。

```
./agent_get -s MQTT_PUBLISH -f work
vi work
```

スクリプト後半部分に下記の様な構文を追加します。先の章で LED 出力のための記述をした場合には、内容を一部編集するだけで簡単に完了します。

```
-- トピック名が "/<sender_hostname>/switch" でメッセージ内容が "ON" の場合に LED 点灯、それ以外の場合には消灯させる
if string.match(g_params["PublishTopic"],"/.+/(.+)") == "switch" then
  if PublishString == "ON" then
    if not raspi_gpio_write(18,true) then error() end
  else
    if not raspi_gpio_write(18,false) then error() end
  end
end
```

MQTT_PUBLISH イベントハンドラがコールされると、リクエストパラメータ "PublishTopic" にはメッセージのトピック名が入ります。イベントハンドラは全ての MQTT エンドポイント(今回は "ClientID＝test_endpoint_12345" のみ)で購読している全てのトピック(今回は "/+/switch" のみ) で共通してコールされます。そのため最初に、処理対象とするメッセージをトピック名などで選別しています。

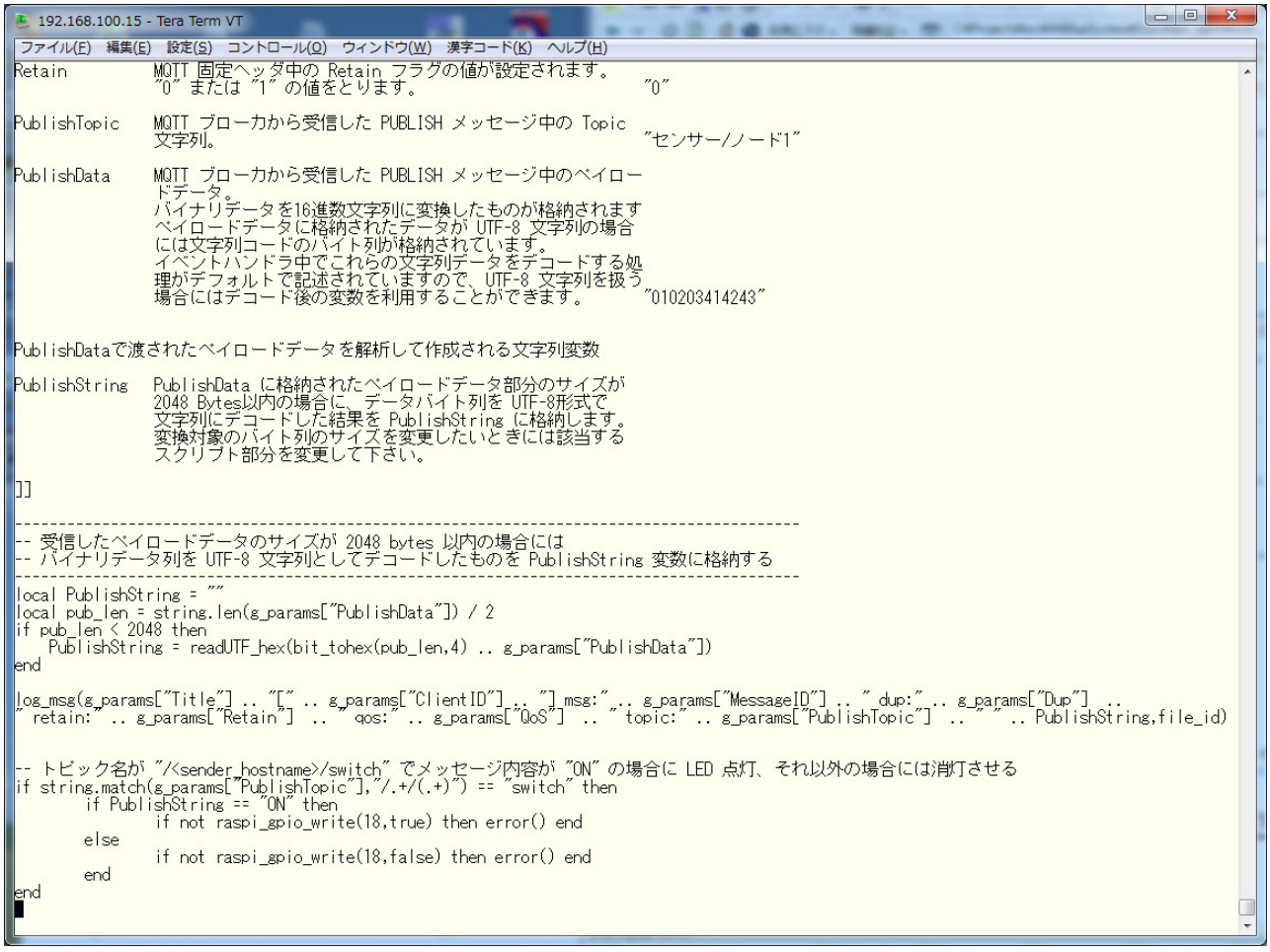
"if string.match(g_params["PublishTopic"],"/.+/(.+)") == "switch" then " 文では、受信したメッセージのト

ピック名が "/<任意の文字列>/switch" のパターンに一致しているかを調べています。string.match() ライブラリ関数は Lua 標準ライブラリの１つで正規表現を利用したパターンマッチ機能を使用できます。

PublishString 変数はこのイベントハンドラの先頭部分で作成しているローカル変数で、MQTT ブローカから受信したバイナリデータを UTF-8 文字列に変換したものです。この文字列にはスイッチを操作したときにMQTT ブローカに送信した "ON" または "OFF" の文字列が入ってきます。

メッセージ内容が "ON" の場合には、LED が接続された GPIO#18 を High にして、それ以外のメッセージ("OFF")の場合には GPIO#18 を Low にします。

編集中の様子は下記の様になります。



編集が完了したら、agent_put コマンドで既存の MQTT_PUBLISH スクリプトを更新します。このとき abs_agent の再起動は必要ありません。

```
./agent_put -s MQTT_PUBLISH -f work
```

スイッチを操作してみてください。スイッチを押すと同時に LED が点灯して離すと LED が消灯します。

ローカルコンピュータ間で MQTT ブローカ経由でメッセージをやり取りしているのですが、見た目は単に LED と スイッチを直列に接続した様な動作になります。

もう少しメッセージの仕組みを反映したデモも簡単にできます。この場合には Raspberry Pi をもう一台用意して、全く同じ様に abs_agent と周辺回路をセットアップします。ただし、abs_agent サーバー設定ファイル中の MQTT エンドポイントに設定する <ClientID> 部分だけは違う文字列にしてください。また、MQTT ブローカは既存の１つだけ動作させておいて、追加の１台の abs_agent で設定する MQTTエンドポイントは共通の MQTT ブローカに接続します。例えば追加する側の abs_agent のサーバー設定ファイル中の MQTTエンドポイントの設定項目は下記の様にします。

```
<MQTT>
  <AutoOnline type="boolean">True</AutoOnline>
  <KeepAliveTimer type="integer">60</KeepAliveTimer>
  <EndPointList>
    <Item>
      <Title>試験用MQTT接続</Title>
      <ClientID>test_endpoint_67890</ClientID>
      <BrokerHostName>192.168.100.14</BrokerHostName>
      <PortNumber>1883</PortNumber>
      <AutoSubscribeTopicList>/+/switch</AutoSubscribeTopicList>
      <AutoSubscribeQoSList>0</AutoSubscribeQoSList>
      <UserName/>
      <Password/>
      <WillTopic/>
      <WillMessage/>
      <WillQoS>0</WillQoS>
      <WillRetain>False</WillRetain>
      <RecvBuffInit>0</RecvBuffInit>
      <DetailLog>True</DetailLog>
    </Item>
  </EndPointList>
</MQTT>
```

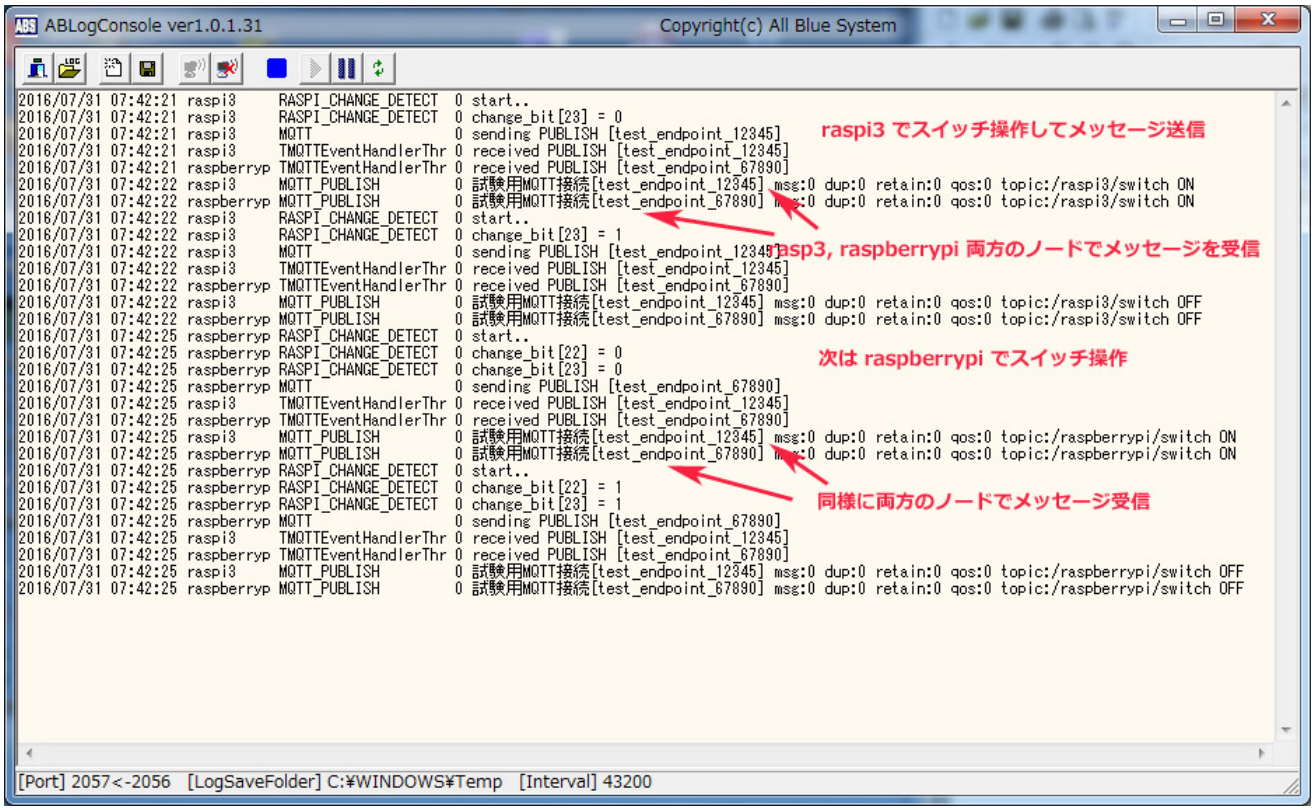
設定が完了したら、追加の Raspberry Pi と abs_agent を起動してください。もちろん、ネットワークにも接続している必要があります。

この状態で片方の Raspberry Pi のスイッチを操作すると、両方の Raspberry Pi の LED が同時に点灯、消灯します。２つの MQTT クライアントが動作している Raspberry Pi は完全に対等な機能なのでどちらを操作しても構いません。また、１台の Raspberry Pi をシャットダウンしても、別の Raspberry Pi のスイッチとLED の操作に影響を

与えない特徴があります。Raspberry Pi を再起動させると何事も無かったように再び連動して LED の操作ができるようになります。ただし、MQTT ブローカが動作しているコンピュータや Raspberry Pi はシャットダウンしないでください。

インターネット上に設置している MQTT ブローカを利用すると、同一LAN 内や別 LAN 内に設置した Raspberry Pi 同士でメッセージのやり取りを簡単にできるのが判ると思います。MQTT ブローカへのソケット接続は常に abs_agent から MQTT ブローカへの一方向なので、ルータ装置での Raspberry Pi 側のローカルアドレスのマッピングや、ドメイン・ホスト名またはグローバルアドレスの付与などの作業は必要はありません。ただし、MQTT ブローカ自身は双方の Raspberry Pi からアクセス可能なアドレスやドメイン・ホスト名を付与されている必要があります。

下記は、２ つの abs_agent にスイッチと LED を接続して操作したときのログコンソールの様子です。スイッチを押した瞬間と離した瞬間それぞれで MQTT ブローカにメッセージを送信して、２つの abs_agent に配信している様子が判ります。



11 サーバープログラム・設定ファイル

マニュアル中のコマンド実行形式部分の記述について

- コマンド実行時には abs_agent キットをインストールしたディレクトリ中のフルパス名、またはシェルの PATH 環境変数からの相対パス名を指定します。
- 実行時には必ず sudo コマンドを併用するなど、root 特権付きで実行してください。
- コマンドパラメータが *[-l <log_server>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパ

ラメータは省略可能です。

11.1 abs_agent (サーバープログラム)

- **機能説明**

  abs_agent サーバープログラム。通常は root 権限を付与してデーモンプロセスとして実行します

- **コマンド実行形式**

  abs_agent *[-f] [-l <log_server>] [-c <config_file>]*

- **コマンドパラメータ**

  -f

  abs_agent プログラムをフォアグランドで実行します。

  "-f"オプションを指定しない場合には abs_agent はデーモンプロセスとしてバックグランドで実行されます。"-f" オプションを指定すると abs_agent はフォアグランドで実行されて、このときログメッセージはコンソールにも出力されますので、別途ログサーバーを使用することなく簡単に動作状態を確認できます。ただし、全てのログメッセージをコンソールに出力しながら abs_agent が動作しますので、デーモンプロセスとして動作させているときよりもパフォーマンスが悪くなります。

  -l <log_server>

  abs_agent プログラムのログメッセージを <log_server> で指定したホストに送信します。このパラメータを指定しない場合には "localhost" にログメッセージを送信します。
  送信には UDP/IP パケットを使用しますので、ログサーバーが動作していない場合でも abs_agent の動作に影響はありません。

  ログサーバーは ABS-9000 LogServer インストールキットに含まれる ABLogServer プログラムを使用します。またリアルタイムにログを確認する場合には同じく ABS-9000 LogServer の LogConsole プログラムを使用してください。

  -c <config_file>

  abs_agent プログラムのサーバー設定ファイルを <config_file> で指定したファイルから読み込みます。

  このパラメータを指定しない場合には abs_agent プログラムが配置された同一ディレクトリにある abs_agent.xml ファイルを使用します。もし設定ファイルが存在しない場合には、デフォルト設定値が格納された新規の設定ファイルが自動的に作成されます。

  <config_file> を指定する時は必ず**絶対パス名で指定してください**。相対パス名で指定すること

はできません。

config_file> の詳しい記述フォーマットについては、"abs_agent.xml" の項を参照して下さい。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています

通常は abs_agent プログラムをデーモンプロセスとして起動しますので、リターン値の取得は行いません

- **環境変数**

LANG　共有変数やマスターデータ等プログラム内で扱うデータで日本語を使用する場合に備えて "ja_JP.UTF-8" を指定します。
コンソールから abs_agent を起動する場合には、実行中のシェル環境変数が "ja_JP.UTF-8" になっている事を確認してください。

- **説明**

abs_agent サーバープログラムです。通常はデーモンプロセスとしてバックグランドで動作させます。

Raspberry Pi 用にビルドされた abs_agent では GPIO, SPI, I2C 機能を使用する場合に、プロセッサのハードウエア機能に直接アクセスしています。このため abs_agent 起動時に root 特権が必要になります。もし特権が付いていない場合にはログに下記の様なメッセージが記録されて、エラーになったハードウエア機能は使用できません。

```
RASPI 0 *ERROR* failed to open /dev/mem, use "sudo" command
```

スクリプト中に、実行するために root 特権が必要な os.execute('/sbin/shutdown -h now &') の様な文を記述する場合や、パーミッションが設定されたシリアルデバイスファイルにアクセスする場合にも同様に abs_agent 起動時に root 特権が必要になります。これらに該当する場合には、コンソールからプログラムを起動する場合には必ず sudo コマンドを使用してください。

/etc/rc.local シェルスクリプト中に abs_agent を起動する記述をする場合には、sudo コマンドの併用は必要ありません。

- **使用例**

● コンソールから abs_agent をデーモンプロセスとして起動する。

ログサーバーを配置していない場合（この場合にはログの詳細を確認することができません）

```
$ sudo ./abs_agent
```

● コンソールから abs_agent をデーモンプロセスとして起動する。

ログサーバーが 192.168.100.45 に設置されている場合

```
$ sudo ./abs_agent -l 192.168.100.45
```

● デーモンとして同一PC で起動中の abs_agent プログラムの状態を確認後、abs_agent をシャットダウンする

```
$ ./agent_stat
--------------------------------------------------------------------------------
Program name       : ABSAgent
Version            : 1.00
Program folder     : /home/pi/abs_agent
Hardware type      : RASPI
Server hostname    : raspberrypi
Server MAC address : B8-27-EB-A7-19-9A
Licensed user      : 開発用ライセンス
License time limit :
Service modules    : MASTERS SESSION SCRIPT CONVERT WEBPROXY MQTT SERIAL RASPI LAST
Client hostname    : raspberrypi
Client MAC address : B8-27-EB-A7-19-9A
--------------------------------------------------------------------------------

$ ./agent_shutdown
$
```

● フォアグランドで abs_agent プログラムを起動して、全てのログ情報をコンソールに出力する。

```
$ sudo ./abs_agent -f
ServiceMain          begin the startup process..
TXASDLServerThread: listener has been created
TXASDLServerThread: socket successfully initialized
TXASDLServerThread: bind on 27101
BASIC                BASIC startup...
BASIC                Startup: licensed to フリー版(業務・商用利用はできません) ,license number is
NON_COMMERCIAL_USE_ONLY
BASIC                StartupModules: start...
MASTERS              MASTERS startup...
SESSION              SESSION startup...
```

```
SCRIPT              PeriodicTimer started
SCRIPT              SCRIPT startup...
CONVERT             CONVERT startup...
WEBPROXY            WEBPROXY startup...
MQTT                KeepAliveTimer started
MQTT                MQTT startup...
SERIAL              SERIAL startup...
RASPI               RASPI startup, hardware = BCM2709
RASPI               RASPI_HWMonitor: start..
SERVER_START        start..
LAST                LAST startup...
LAST                licensed to フリー版(業務・商用利用はできません) ,license number is
NON_COMMERCIAL_USE_ONLY
SAMPLE              start..
SAMPLE              モジュール開始時刻 2016/06/15 11:45:59 ホスト名 raspi3 リクエスト元 raspi3
SAMPLE              スクリプトに渡されたパラメーター覧
SAMPLE              g_params[Key1] = Val1
ServiceMain         begin the shutdown process..
SERVER_STOP         start..
LAST                LAST shutdown.
RASPI               RASPI_HWMonitor: finished
RASPI               RASPI shutdown.
SERIAL              SERIAL shutdown.
MQTT                MQTT shutdown.
WEBPROXY            WEBPROXY shutdown.
CONVERT             CONVERT shutdown.
SCRIPT              SCRIPT shutdown.
SESSION             SESSION shutdown.
MASTERS             MASTERS shutdown.
BASIC               BASIC shutdown.
TXASDLServerThread: listener has been deleted
ServiceMain         service listener shutdown
ServiceMain         service monitor stopped
$
```

フォアグランドで実行中のコンソールとは別に、同一 PC にコンソールを接続した状態でサーバーのステータス確認(agent_stat)と SAMPLE スクリプトの実行(agent_script)を行います。その後、サーバーを停止させています(agent_shutdown)。これらのログがフォアグランドで abs_agent を実行中のコンソールに出力されます。

abs_agent プログラムをフォアグランドで実行しているときには、実行中のコンソールに <Ctrl-C> を入力してプロセスを強制的に終了できます。ただし、この場合 abs_agent のサービスモジュール終了処理を実行しないため、 abs_agent がファイルアクセスしていた時等に不具合が発生する可能性があります。これらをを防ぐためにこの例の様に、別コンソールから agent_shutdown プログラムで停止させるようにしてください。

```
$ ./agent_stat

------------------------------------------------------------------------------
Program name       : ABSAgent
Version            : 1.00
Program folder     : /home/pi/abs_agent
Device type        : RASPI
Server hostname    : raspi3
Server MAC address : B8-27-EB-D8-7C-2D
Licensed user      : フリー版(業務・商用利用はできません)
License time limit :
Service modules    : MASTERS SESSION SCRIPT CONVERT WEBPROXY MQTT SERIAL RASPI LAST
Client hostname    : raspi3
Client MAC address : B8-27-EB-D8-7C-2D
------------------------------------------------------------------------------

$ ./agent_script -s SAMPLE -k Key1 -v Val1

ServerName: raspi3      ResultParam(s): 0
----------------------------------------------------------------------
<Key>=<Value>
----------------------------------------------------------------------

$ ./agent_shutdown
$
```

● /home/pi/abs_agent ディレクトリに abs_agent プログラムを配置して、デーモンプロセスとして自動起動を行う様に /etc/rc.local に設定する。

/etc/rc.local 等の起動スクリプト中から abs_agent を自動起動する場合には絶対パス名で abs_agent プログラムを指定します。

abs_agent プログラム中の共有変数やリスト、マスターファイル等で日本語を使用する場合に備えて、LANG 環

境変数を ja_JP.UTF-8 に設定した状態でプログラムを起動します。

この例では abs_agent プログラムの設定はデフォルトのファイルを使用しないで、/home/pi/my_config ディレクトリに格納した abs_agent.xml ファイルを指定しています。

ログサーバーが 192.168.100.45 に設置されている場合

[/etc/rc.local の内容]

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s¥n" "$_IP"
fi

#################################
## start abs_agent daemon
#################################
LANG=ja_JP.UTF-8
export LANG
/home/pi/abs_agent/abs_agent -l 192.168.100.45 -c /home/pi/my_config/abs_agent.xml

exit 0
```

- **参照**

agent_stat	abs_agent 動作ステータス表示用コマンド
agent_shutdown	abs_agent シャットダウンコマンド
abs_agent.xml	abs_agent サーバー設定ファイル

11.2 abs_agent.xml (サーバー設定ファイル)

- **機能説明**

  abs_agent サーバープログラムのコンフィギュレーションファイル

- **ファイル**

  <abs_agentをインストールしたディレクトリ>/abs_agent.xml または、abs_agent プログラム起動時に "-c <config_file>" パラメータで指定したファイル。

  abs_agent からアクセスするスクリプトディレクトリやマスターファイル、HTTP Webサーバー公開ディレクトリは、設定ファイル中に記述されているファイルパスを使用します。デフォルトでは使用する設定ファイルが配置されているパスを元に決定されます。

- **設定項目**

  XML ドキュメントの DocumentElement の各チャイルドノードに設定項目が記述されています。設定項目はサービスクラス毎にタグが分かれていて、この章の後半部分に各サービスクラスの詳細設定項目を記載しています。

  設定項目が /xxxx/xxxx/xxxx の様なパス名形式で記述されている場合には、コンフィギュレーションファイル中の XML タグのツリー構造をディレクトリパスの様に表現しています。例えば、コンフィギュレーション項目が "/Document/Class/Script/DetailLog" の場合にはコンフィギュレーションファイル中の <Document> タグ内の <Class> チャイルドタグ内の <Script> チャイルドタグ内の <DetailLog>タグの値またはそのチャイルドノードを全体を表しています。

  設定項目の XML タグ中にある "type" 属性値はタグの値を記述するときの型を示しています。XML ファイル中では常に文字列として編集しますが、abs_agent 内部ではここで指定されている型に変換してから使用されます。abs_agent ではコンフィギュレーションファイル中の type 属性値は使用していません。これは、ユーザーがエディタ等で直接編集するときに正しい値の型を確認できるようにする目的で存在します。そのためこれらの属性値はデフォルト値のままにしておいて下さい。

  設定項目毎の表中にあるタグ名に (*) が付いた項目は、設定データ文字列中に XML リザーブ文字をエンコードした形式で含めることが出来ます。(詳しくは後述)

- **補足**

  abs_agent 実行中に abs_agent.xml ファイルをエディタ等で編集することができますが、最新の設定項目を有効にするためには abs_agent プログラムを再起動する必要があります。

  手動でファイル内容を編集する場合には、正しいXML フォーマット（整形式のXML文書）になっていることを確

認して下さい。XML フォーマットにエラーがあると abs_agent プログラムを起動できません。特にタグ名のタイプミスや文字列以外の制御コードなどを含まないようにして下さい。

設定ファイルの文字コードは必ず UTF-8N(BOM無し)で保存してください。Shift_JIS や その他の文字コード形式には対応していません。

改行コードは Unixタイプ(LFのみ)または DOSタイプ(CR-LF) のどちらを使用しても構いません。Webブラウザプログラム等で abs_agent.xml ファイルをロードしてみると、簡単に XML フォーマットの整合性をチェックすることができます。

インストールキットに含まれている abs_agent.xml ファイルは、abs_agent プログラムのデフォルト設定値から変更する部分の項目のみが初期値として記述されています。インストール直後に最初に abs_agent プログラムが起動されると、abs_agent.xml ファイル中の未設定の項目(XMLタグの内容が空の項目)の設定値にはデフォルト値が書き込まれます。例えば、abs_agentで公開する HTTP サーバーのポート番号を最初の起動時から 80 で使用したい場合には、インストール直後にエディタで abs_agent.xml ファイル中の XMLタグ "/Document/Class/WebProxy/HTTPServerPort" に "80" を書き込んでから abs_agent を起動します。

- **サーバー設定ファイル中に XMLリザーブ文字を使用する場合**

abs_agent.xml ファイル中に XML でリザーブされた下記の文字を使用したい場合には、予め予約されたエンコード文字列で置き換えて記述してください。下記の表の左側が本来記述したい文字列で、右側は実際に abs_agent.xml ファイル中に書き込む文字列になります。

これらの置き換えられた文字列は abs_agent プログラム内で本来の文字列に直されてから使用されます。ただし、これらの置き換え文字列が使用可能な設定項目はタグごとに予め決められています。詳しくは設定項目ごとの説明をご覧ください。

abs_agent クライアントプログラムからリザーブ文字を使用する場合には、クライアントプログラム内部でこれらのエンコード処理が自動で行われますので本来の文字列をそのまま使用することができます。

設定ファイル中で表現したい文字列	実際に設定ファイル中に記述するエンコード文字列 (セミコロンまでを含めた文字列を指定します)
&	@#38;
<	@#60;
>	@#62;
[	@#91;
]	@#93;
%	@#37;
*	@#42;
タブ文字 (0x09)	@#09;

改行文字 (0x13, 0x10)	@#13;@#10;

- **サーバー設定ファイル例**

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
  <Description>ABSAgent コンフィギュレーション</Description>
  <ServiceMain>
    <PortNumber type="integer">27101</PortNumber>
    <DefaultRemoteHost type="string">127.0.0.1</DefaultRemoteHost>
    <TimeStampMargin type="integer">0</TimeStampMargin>
    <AllowFileUpload type="boolean">True</AllowFileUpload>
    <UseMACProtection type="boolean">False</UseMACProtection>
  </ServiceMain>
  <Class>
    <Basic>
      <ServerKey type="string"></ServerKey>
      <LicenseKey type="string">azB1hcyiO6dtImSau1 .... nffLhU/Y4DOStBc9C745zOCOS/LGad</LicenseKey>
      <AllowFileOperation type="boolean">False</AllowFileOperation>
    </Basic>
    <Convert>
      <AutoOnline type="boolean">True</AutoOnline>
    </Convert>
    <Masters>
      <AutoOnline type="boolean">True</AutoOnline>
      <MasterFile type="string">/home/pi/abs_agent/masters.xml</MasterFile>
      <XMLSessionPool type="integer">4</XMLSessionPool>
    </Masters>
    <Session>
      <AutoOnline type="boolean">True</AutoOnline>
    </Session>
    <Script>
      <AutoOnline type="boolean">True</AutoOnline>
      <ScriptFolder type="string">/home/pi/abs_agent/scripts</ScriptFolder>
      <SessionPool type="integer">16</SessionPool>
      <UsePeriodicTimer type="boolean">True</UsePeriodicTimer>
    </Script>
    <WebProxy>
      <AutoOnline type="boolean">True</AutoOnline>
```

```
    <UseHTTPServer type="boolean">True</UseHTTPServer>
    <DetailLog type="boolean">True</DetailLog>
    <HTTPServerPort type="integer">8080</HTTPServerPort>
    <PubRoot type="string">/home/pi/abs_agent/webroot</PubRoot>
  </WebProxy>
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList>
      <Item>
        <COMPort>/dev/ttyUSB1</COMPort>
        <Type>STRING</Type>
        <BufferedMode>True</BufferedMode>
        <BaudRate>9600</BaudRate>
        <DataBits>8</DataBits>
        <StopBits>1</StopBits>
        <ParityBit>NONE</ParityBit>
        <FlowControl>NONE</FlowControl>
        <Title>ループバックデバイス</Title>
      </Item>
      <Item>
        <COMPort>/dev/ttyUSB0</COMPort>
        <Type>FIRMATA</Type>
        <BufferedMode>False</BufferedMode>
        <BaudRate>57600</BaudRate>
        <DataBits>8</DataBits>
        <StopBits>1</StopBits>
        <ParityBit>NONE</ParityBit>
        <FlowControl>NONE</FlowControl>
        <Title>Arduino実験ボード#1</Title>
      </Item>
      <Item>
        <COMPort>/dev/ttyUSBError</COMPort>
        <Type>FIRMATA</Type>
        <BufferedMode>False</BufferedMode>
        <BaudRate>19200</BaudRate>
        <DataBits>8</DataBits>
        <StopBits>1</StopBits>
        <ParityBit>NONE</ParityBit>
        <FlowControl>HARD</FlowControl>
```

```
            <Title>エラー発生シリアルデバイス</Title>
          </Item>
        </DeviceList>
      </Serial>
      <MQTT>
        <AutoOnline type="boolean">True</AutoOnline>
        <KeepAliveTimer type="integer">60</KeepAliveTimer>
        <EndPointList>
          <Item>
            <Title>センサーデータ登録</Title>
            <ClientID>abs9k:1111-raspi</ClientID>
            <BrokerHostName>192.168.100.14</BrokerHostName>
            <PortNumber>1883</PortNumber>
            <AutoSubscribeTopicList>hello/world,/sensor/#,abc/+/データ</AutoSubscribeTopicList>
            <AutoSubscribeQoSList>2,2,1</AutoSubscribeQoSList>
            <UserName>my_user</UserName>
            <Password>my_password</Password>
            <WillTopic>my_will</WillTopic>
            <WillMessage>これは試験　Willです。。@#60;@#60;@#91;@#91;??##@#38;@#38;試験
@#38;@#38;##??@#93;@#93;@#62;@#62;</WillMessage>
            <WillQoS>0</WillQoS>
            <WillRetain>True</WillRetain>
            <RecvBuffInit>10000</RecvBuffInit>
            <DetailLog>True</DetailLog>
          </Item>
          <Item>
            <Title>title2</Title>
            <ClientID>abs9k:2222-raspi</ClientID>
            <BrokerHostName>192.168.100.14</BrokerHostName>
            <PortNumber>1883</PortNumber>
            <AutoSubscribeTopicList/>
            <AutoSubscribeQoSList/>
            <UserName/>
            <Password/>
            <WillTopic>my_will</WillTopic>
            <WillMessage>さようなら</WillMessage>
            <WillQoS>0</WillQoS>
            <WillRetain>False</WillRetain>
            <RecvBuffInit>2000</RecvBuffInit>
```

```
    <DetailLog>True</DetailLog>
  </Item>
  <Item>
    <Title>エラー発生エンドポイント</Title>
    <ClientID>abs9k:444-raspi</ClientID>
    <BrokerHostName>192.168.100.14</BrokerHostName>
    <PortNumber>1883</PortNumber>
    <AutoSubscribeTopicList/>
    <AutoSubscribeQoSList/>
    <UserName>sss</UserName>
    <Password>sss</Password>
    <WillTopic/>
    <WillMessage/>
    <WillQoS>0</WillQoS>
    <WillRetain>False</WillRetain>
    <RecvBuffInit>0</RecvBuffInit>
    <DetailLog>True</DetailLog>
  </Item>
  <Item>
    <Title>センサーデータ取得</Title>
    <ClientID>abs9k:93501-raspi</ClientID>
    <BrokerHostName>192.168.100.14</BrokerHostName>
    <PortNumber>1883</PortNumber>
    <AutoSubscribeTopicList>/+/+/io,/+/+/tdcp,/twe/+/Samp_Monitor</AutoSubscribeTopicList>
    <AutoSubscribeQoSList>1,1,1</AutoSubscribeQoSList>
    <UserName/>
    <Password/>
    <WillTopic/>
    <WillMessage/>
    <WillQoS>0</WillQoS>
    <WillRetain>False</WillRetain>
    <RecvBuffInit>2048</RecvBuffInit>
    <DetailLog>False</DetailLog>
  </Item>
  <Item>
    <Title>接続エラーエンドポイント</Title>
    <ClientID>abs9k:888-raspi</ClientID>
    <BrokerHostName>192.168.100.144</BrokerHostName>
    <PortNumber>1883</PortNumber>
```

```
            <AutoSubscribeTopicList/>
            <AutoSubscribeQoSList/>
            <UserName>sss</UserName>
            <Password>sss</Password>
            <WillTopic/>
            <WillMessage/>
            <WillQoS>0</WillQoS>
            <WillRetain>False</WillRetain>
            <RecvBuffInit>0</RecvBuffInit>
            <DetailLog>True</DetailLog>
          </Item>
        </EndPointList>
      </MQTT>
      <RASPI>
        <AutoOnline type="boolean">True</AutoOnline>
      </RASPI>
    </Class>
</Document>
```

- **参照**

  agent_stat　　abs_agent 動作ステータス表示用コマンド

  abs_agent　　abs_agent サーバープログラム

  以下はサービスクラス毎に分けられたタグ中の設定項目の詳細です。

11.2.1 ServiceMain

/Document/ServiceMain: abs_agent プログラム動作全般の設定項目

XML ドキュメントタグ名	説明
PortNumber	abs_agent クライアントプログラムと通信するための TCP/IP ポート番号 整数を指定 デフォルト値: 27101（この値は変更しないでください）
DefaultRemoteHost	リモートアクセス API 関数のデフォルトホスト名 abs_agent で提供する Lua API ライブラリ関数で、リモート側の abs_agent にアクセスする場合のホスト名を省略した場合に使用されるホスト名。ホスト名文字列または IP アドレスを指定する。 デフォルト値: 127.0.0.1
TimeStampMargin	abs_agent が動作しているコンピュータの時刻と、リモート側からアクセスしたコンピュータの時刻のずれをどれだけ許容するかを指定する。秒単位の整数

	を指定する。 0 を指定すると時刻の違いを検出しない。 デフォルト値: 0
AllowFileUpload	abs_agent が動作しているコンピュータにファイルのアップロードを許可する。agent_put コマンドでスクリプトファイルをアップロードさせる場合には True を指定します。True または False を指定する。 デフォルト値: True
UseMACProtection	リモートからのアクセス許可を設定する agent_hosts コマンドで、ホスト名の代わりに MAC アドレスを使用する。MAC アドレスは "XX-XX-XX-XX-XX-XX" で表現される１６進数文字列とハイフンの文字列。 True または False を指定する。 デフォルト値: False

11.2.2 Basic

/Document/Class/Basic: 基本機能の設定項目

XML ドキュメントタグ名	説明
ServerKey	現在未使用 デフォルト値: ""
LicenseKey	abs_agent プログラムのライセンス文字列を設定します。 "agent_stat -l <license_file>"コマンドを使用すると、この部分にライセンスキー文字列を格納することができます。 デフォルト値: ""
AllowFileOperation	現在未使用 デフォルト値: False

11.2.3 Convert

/Document/Class/Convert: 変換機能やベクトル・行列ライブラリの設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
UseMatrix	ベクトル･行列ライブラリを使用するかどうかを指定する。True または False を指定する。 デフォルト値:False
CBLAS_Library	BLAS パッケージの共有ライブラリ名。 デフォルト値:libblas.so

LAPACKE_Library	LAPACKEパッケージの共有ライブラリ名。 デフォルト値:liblapacke.so
DGEEV_AutoScale	LAPACKE_dgeev 計算時に固有ベクトルを見易くするために補正(定数倍)する デフォルト値:True

11.2.4 Masters

/Document/Class/Masters: マスターファイル機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
MasterFile	abs_agent のマスター機能で使用するマスターファイル名 マスターファイル名には絶対パス名の文字列を設定します。ファイル名の中に $CONFIGDIR$ を記述するとその部分が、コンフィギュレーションファイルが配置されているディレクトリ名(絶対パス名)に置き換えられて解釈されます。 デフォルト値: $CONFIGDIR$/masters.xml
XMLSessionPool	マスターファイルのキャッシュを保持するインスタンス数 整数を指定 デフォルト値:4

11.2.5 Session

/Document/Class/Session: グローバル共有変数、グローバル共有文字列リスト機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
UseGlobalWatch	イベントハンドラ GLOBAL_WATCH を使用するかどうかを指定する。True または False を指定する。 デフォルト値:False

11.2.6 Script

/Document/Class/Script: イベントハンドラ、ユーザースクリプト機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にす

	る。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
ScriptFolder	abs_agent のイベントハンドラやユーザースクリプトで使用する Lua スクリプトファイルを格納しているディレクトリ。絶対パス名の文字列を設定します。ディレクトリ名の中に $CONFIGDIR$ を記述するとその部分が、コンフィギュレーションファイルが配置されているディレクトリ名(絶対パス名)に置き換えられて解釈されます。 デフォルト値: $CONFIGDIR$/scripts
SessionPool	スクリプト実行時に使用する Lua スクリプトエンジンのインスタンス数。ここで設定された値まで、ユーザースクリプトやイベントハンドラが同時に実行することができます。この値はライセンスによって設定可能な上限があります。整数を指定 デフォルト値:16
APISearchOffset	Lua APIライブラリ関数からスクリプト実行時に検索する未使用のスクリプトインスタンスを、ここで指定したエントリ番号以降に制限する。0よりも大きな整数を指定することで未使用状態のエントリを確保して、後で優先度の高いスクリプトが実行可能な状態に置く事ができる。イベントハンドラやクライアントプログラムによるスクリプト実行は、この設定に関わらず常に 0番目のエントリから検索する。 デフォルト値:2
UsePeriodicTimer	イベントハンドラ PERIODIC_TIMER, TICK_TIMER を使用するかどうかを指定する。True または False を指定する。 デフォルト値:True

11.2.7 WebProxy

/Document/Class/WebProxy: HTTPサーバー、Web API機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
DetailLog	HTTP サーバーや Web API,WebSocketサーバーで発生する詳細なログメッセージを、ログサーバーに出力するかどうかを指定する。True または False を指定する。 デフォルト値:True
UseHTTPServer	HTTP サーバー機能と Web API 機能を使用するかどうかを指定する。True または False を指定する。 デフォルト値:True

HTTPServerPort	HTTPサーバーとWeb APIで使用する TCP/IP ポート番号。整数を指定 デフォルト値:8080
PubRoot	HTTPサーバーで公開するファイルを格納するトップディレクトリ。絶対パス名の文字列を設定します。ディレクトリ名内に $CONFIGDIR$ を記述するとその部分が、コンフィギュレーションファイルが配置されているディレクトリ名(絶対パス名)に置き換えられて解釈されます。 デフォルト値: $CONFIGDIR$/webroot
UseWebSocketServer	WebSocketサーバー機能を使用するかどうかを指定する。True または False を指定する。 デフォルト値:True
WebSocketServerPort	WebSocketサーバーで使用する TCP/IP ポート番号。整数を指定 デフォルト値:9090
MaxWaitToPreventOverwrite	WebSocket サーバーから接続中のクライアントにフレームデータを送信するときに、前回設定したデータが送信済みになっていないときの最大待ち時間(ms)。詳しくは "WebSocketサーバーAPI" の章を参照してください。 デフォルト値:20
SendCloseResponse	WebSoscket コントロールフレームの "Close"(opcode =8) をクライアント側から受信した時に、レスポンスとしてサーバー側から "Close" コントロールフレームを送信する。RFC6455 によると送信すべきであるが、一部のクライアント接続時に不具合が発生する場合があるため、デフォルト設定では "Close" コントロールフレームは送信しない。 デフォルト値:False

11.2.8 Serial

/Document/Class/Serial: シリアルデバイス機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True (この値は変更しないでください)
DeviceList	abs_agent から使用するシリアルデバイスリストを定義する DevliceList 中に <Item> ... </Item> タグで囲まれたシリアルデバイスを複数格納できます。<Item> タグ内には、各シリアルデバイス毎の設定が格納されます。詳しくは 下記の (8.1) を参照してください。 リスト中に格納可能な <Item> の個数はライセンスによって上限があります。
MaxBufferRecordCount	BufferedMode を True に設定したシリアルデバイスにおいて、文字列バッファに保存可能な最大データレコード(文字列)数。この数を超えた場合には、最初に登録したデータエントリから順に削除されます。文字列バッファはシリア

	ルデバイス毎に独立して作成されます。1以上の整数を指定する。 デフォルト値: 100

/Document/Class/Serial/DeviceList/Item 中に格納する設定項目

XML ドキュメントタグ名	説明
Enabled	このシリアルデバイスエントリを有効にするかどうかを指定する。このタグを省略した場合には True が指定されます。一時的にデバイスを無効にする場合に False を設定して、abs_agent 起動時にこのエントリを無視させることができます。True または False を指定する。 デフォルト値:True
COMPort	シリアルポートにアサインされたデバイスファイル名を指定します。USB-シリアル変換器経由で接続している場合には、/dev/USBxx 等のデバイスファイル名になります。CPU ボードに予め用意されたシリアルインターフェイスポートに接続する場合には /dev/ttyACMxx, /dev/ttySxx 等になります。 指定するデバイスファイルは他のプロセスで使用中でないことを確認してください。ログインポートとして使用可能な場合には、予め "getty" 等のプロセスからデバイスファイルを使用しないように OS の設定を変更して下さい。 デフォルト値:""
Type	シリアルポートで入出力するデータを扱うときに、デバイスタイプを指定することで予め決められたフォーマットでデータを扱うことができます。 指定可能なデバイスタイプは以下になります。 STRING 文字列形式でシリアルデータを扱います。文字列の終端を自動で判別して、シリアルデバイスから受信した文字列ごとにイベントを発生させたり、文字列データをバッファから取り出すことができます) 文字列の終端は、 Null(0x00), LF(0x0A), CR(0x0D)または CR-LF(0x0D,0x0A)の何れかのデータとして判断します。 FIRMATA 主に Arduino デバイスとの通信で使用されるプロトコルを扱います。Arduinoでは FIRMATA ライブラリとして標準で用意されていて、バイナリデータやコマンド、文字列データ等を予め決められたプロトコルでやりとりできます。 abs_agent では FIRMATA 用のライブラリ関数を用意していますので、簡単にスクリプトから FIRMATAを扱えます。

	XBEE XBee 802.15.4 (S1)モジュールの API Mode1 で使用されるプロトコルを扱います。XBee, XBee Pro(S1) を USBアダプタやシリアルポートに接続することで、abs_agent からリモートに設置した複数のXBee モジュールをコントロールできます。ローカル(abs_agent に接続した XBee) やリモート側の XBee に対してリモートAT コマンドを実行することもできます。 abs_agent では データ送信やATコマンド実行用のライブラリ関数を用意していますので、簡単にスクリプトからリモートXBeeを操作できます。 リモート XBee からデータを受信した場合には SERIAL_XBEE イベントハンドラが起動されますので、そのイベントハンドラ内にデータ処理スクリプトを記述できます。 ZB XBee-ZB Zigbee (S2)モジュールの API Mode1 で使用されるプロトコルを扱います。XBee-ZB, XBee-ZB Pro(S2) を USBアダプタやシリアルポートに接続することで、abs_agent からリモートに設置した複数のXBee-ZB モジュールをコントロールできます。ローカル(abs_agent に接続した XBee-ZB) やリモート側の XBee-ZB に対してリモートAT コマンドを実行することもできます。 ローカルに接続可能な XBee-ZB のデバイスタイプは "coordinator" と "router" です。リモート側 XBee-ZB には "end device" デバイスタイプを含む全てのデバイスタイプに対してアクセスできます。 abs_agent では データ送信やATコマンド実行用のライブラリ関数を用意していますので、簡単にスクリプトからリモートXBee-ZBを操作できます。 リモート XBee-ZB からデータを受信した場合には SERIAL_ZB イベントハンドラが起動されますので、そのイベントハンドラ内にデータ処理スクリプトを記述できます。 RAW シリアルポートから受信したバイナリデータをそのままの形で扱います。デバイスドライバ経由で受信したデータブロック毎にイベントが発生します。 デフォルト値:""
BufferedMode	シリアルポートからデータを受信したときに、イベントを発生させるかまたは内部のバッファにデータを格納して、後からライブラリ関数を使用してデータ

	を取り出すかを設定します。デバイスタイプ "Type" が STRING, FIRMATA, RAW の場合のみ有効なタグで、それ以外のデバイスタイプではこのタグは無視されて常に BufferedMode = False となります。 True に設定すると serial_readln() ライブラリ関数を使用して、受信したデータを手動で取り出すことができます。このとき、Type に設定したデバイスタイプ毎に決められた１パケット分のデータをまとめて取り出せます。 False に設定すると、Type に設定したデバイスタイプ毎の１パケット分のデータを受信した時に、それぞれのデバイスタイプ毎に予め決められたイベントハンドラスクリプトが自動的に実行されます。False に設定したデバイスに対して serial_readln() を実行してもデータを取得することはできません。 シリアルデバイスに対して、専用の読み込みループを作成してデータを処理する場合には True を指定します。シリアルデバイスから受信したデータパケットをリアルタイムに処理・応答するようなアプリケーションを作成する場合には False を指定する方がスクリプトが簡単になります。 True または False を指定する。 デフォルト値:""
BaudRate	シリアルポートの通信ボーレートを指定します。 指定可能な値は下記から選択してください。 300 1200 2400 4800 9600 14400 19200 38400 57600 115200 デフォルト値:""
DataBits	シリアルポートのデータビット数を指定します。 指定可能な値は下記から選択してください。 5 6 7 8

	デフォルト値:""
StopBits	シリアルポートのストップビット数を指定します。 指定可能な値は下記から選択してください。 1 1.5 2 デフォルト値:""
ParityBit	シリアルポートのパリティビットを指定します。 指定可能な値は下記から選択してください。 NONE　パリティ無し EVEN　偶数パリティ ODD　奇数パリティ デフォルト値:""
FlowControl	シリアルポート通信時のフロー制御を指定します。 指定可能な値は下記から選択してください。 NONE　フロー制御無し。 HARD　RTS, CTS ラインを使用したフロー制御を行います SOFT　XON, XOFF データによるフロー制御を行います デフォルト値:""
(*)Title	シリアルデバイスに任意のタイトル文字列を設定することができます。 タイトルを指定すると、シリアルポートを操作するライブラリ関数のパラメータで、デバイスファイル名の代わりに、ここで設定したタイトル文字列を指定することができます。また、イベントハンドラが実行されるときのパラメータに "Title" が追加されて、ここで設定したタイトル文字列が格納されるようになります。 デフォルト値:""

11.2.9 MQTT

/Document/Class/MQTT: MQTTクライアント機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
KeepAliveTimer	MQTT ブローカ間のソケット接続確認間隔を秒単位で指定する。 MQTT ブローカに CONNECT メッセージを送信するときに、KeepAliveTimer フィールドにここで設定した値が格納される。また、設定した秒数の間隔で abs_agent 側の MQTT_KEEP_ALIVE_TIMER イベントハンドラが定期的に実行される。

	デフォルトの MQTT_KEEP_ALIVE_TIMER イベントハンドラでは MQTT ブローカに接続中の全てのエンドポイントで PINGREQ メッセージを送信するように記述されています。もし、ブローカとのソケット接続でエラーを検出した場合には、自動的に再接続を行います。詳しくは MQTT_KEEP_ALIVE_TIMER イベントの項を参照して下さい。整数を指定 デフォルト値:60
EndPointList	abs_agent から使用するMQTTエンドポイントリストを定義する EndPointList 中に <Item> ... </Item> タグで囲まれたMQTT クライアントエンドポイントを複数格納できます。<Item> タグ内には、各エンドポイントの設定が格納されます。詳しくは 下記の (9.1) を参照してください。 リスト中に格納可能な <Item> の個数はライセンスによって上限があります。

/Document/Class/MQTT/EndPointList/Item 中に格納する設定項目

XML ドキュメントタグ名	説明
Enabled	この MQTT エンドポイントを有効にするかどうかを指定する。このタグを省略した場合には True が指定されます。一時的にエンドポイントを無効にする場合に False を設定して、abs_agent 起動時にこのエントリを無視させることができます。True または False を指定する。 デフォルト値:True
(*)Title	MQTT エンドポイントに任意のタイトル文字列を設定することができます。タイトルを設定すると、MQTT ブローカにリクエストを送信するライブラリ関数のパラメータに ClientID の代わりにタイトル文字列を指定することができます。MQTT ブローカから PUBLISH メッセージを受信したときに abs_agent 側で実行される MQTT_PUBLISH イベントハンドラの "Title" パラメータにここで設定したタイトル文字列が入ります。 デフォルト値:""
ClientID	MQTT ブローカに接続するときに使用する ClientID 文字列を設定します。 任意の文字列を指定できますが、MQTT ブローカに接続する全てのエンドポイント間でユニークな文字列を指定する必要があります。 デフォルト値:""
BrokerHostName	MQTT ブローカのホスト名または IP アドレス デフォルト値:""
PortNumber	MQTT ブローカのポート番号。通常は 1883 を指定します。整数を指定 デフォルト値:""
AutoSubscribeTopicList	abs_agent 起動時に MQTT ブローカに対してここで指定したトピックを購読するように、自動で SUBSCRIBE リクエストメッセージを送信します。 カンマ区切りで複数のトピックを指定することができます。この設定項目でトピックを自動購読しなくても、後から mqtt_subscribe() ライブラリ関数を使

	用して任意のトピックを購読することもできます。カンマ区切りのトピック文字列を指定 トピック文字列中に $HOSTNAME$ を記述すると自身のホスト名に置き換えます。この場合、MQTT エンドポイント接続開始時に、ブローカ側に送信されるトピック名の該当部分が abs_agent の動作しているホスト名に置き換えられます。 デフォルト値：””
AutoSubscribeQoSList	AutoSubscribeTopicList 設定項目で指定したトピックの QoS を指定します。カンマ区切りで複数の QoS を指定する場合には必ず AutoSubscribeTopicList 設定項目で指定したトピック数と並びを合わせてください。"0","1","2" の何れかを文字列形式で設定します。カンマ区切りのQoS文字列を指定 デフォルト値:””
UserName	MQTT ブローカに接続するときのユーザー名を指定します Password項目と共に設定を ”” 空文字列にした場合には、MQTT ブローカに CONNECT コマンドを送信するときに、ユーザー名とパスワードを省略します。 デフォルト値:””
Password	MQTT ブローカに接続するときのパスワードを指定します UserName 項目と共に設定を ”” 空文字列にした場合には、MQTT ブローカに CONNECT コマンドを送信するときに、ユーザー名とパスワードを省略します。 デフォルト値:””
WillTopic	MQTT ブローカに接続するときの Will Topic文字列を設定します。 WillMessage 設定項目と共に設定を ”” 空文字列にした場合には、MQTT ブローカにはWillトピック・Will メッセージを省略して接続します。 デフォルト値:””
(*)WillMessage	MQTT ブローカに接続するときの Will Message文字列を設定します。 WillTopic 設定項目と共に設定を ”” 空文字列にした場合には、MQTT ブローカにはWillトピック・Will メッセージを省略して接続します。 デフォルト値:””
WillQoS	MQTT ブローカに接続するときの Will QoS文字列を設定します。"0","1","2" の何れかを設定します。 デフォルト値:””
WillRetain	MQTT に接続するときの WillRetain フラグの値を設定します。True または False を指定する。 デフォルト値:””
RecvBuffInit	ブローカからMQTT メッセージパケットを受信するときに利用する、一時的なデータ保存領域のサイズ初期値を指定します。 ここで設定した値よりも大きなデータをブローカから受信した場合には、自動でサイズが拡張されます。受信するデータサイズを予測できる場合には、バッ

	ファ初期値を予めここで設定できます。このようにすると、データサイズ拡張時のオーバヘッドが少なくなりますので、受信時のレスポンスが早くなります。 例えば、1000000 を指定すると、1MBytesまでのデータサイズのパケットを受信するときに最適な値となります。整数を指定 デフォルト値:""
DetailLog	ログサーバーに MQTT パケットのやり取りに関する詳細メッセージを出力します。True または False を指定する。 デフォルト値:""

11.2.10 RASPI (Raspberry Piのみ)

/Document/Class/RASPI: Raspberry Pi ハードウエア機能の設定項目

このタグは Raspberry Pi 用にビルドされた abs_agent でのみ有効なサービスクラスの設定です。

Raspberry Pi 用の abs_agent では RASPI サービスモジュールを使用してハードウエアに直接アクセスします。このため abs_agent 起動時には必ずスーパーユーザー(root) 特権を付けて実行してください。具体的には /etc/rc.local 等の起動スクリプト中から abs_agent を自動起動させるかまたは、コンソールから "sudo ./abs_agent -l xx.xx.xx.xx" 等のコマンドで起動させてください。

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 "agent_stat" コマンドで得られる "Hardware Type" が RASPI の場合はデフォルト値:True "agent_stat" コマンドで得られる "Hardware Type" が RASPI 以外の場合はデフォルト値:False (この値は変更しないでください)
Hardware	Raspbian OS の /proc/cpuinfo ファイルから取得した Hardware 情報(ハードウエアタイプ文字列)よりも、ここで設定した値を優先して使用する。OS のバージョン違いによるハードウエアタイプ文字列を修正する場合に使用する。空白の場合には /proc/cpuinfo の情報のみを使用する。 BCM2708 , BCM2709 , BCM2710 , BCM2711 の何れかを指定する。 BCM2835 SoC を使用している Raspberry Pi ver1, Zero 等の場合には BCM2708 を指定します。BCM2836, BCM2837 SoC を使用している Raspberry Pi ver2, ver3 等の場合には BCM2709 を指定します。Raspberry Pi Zero 2 W の場合には、BCM2710 を指定します。Raspberry Pi ver4 の場合には BCM2711 を指定します。 デフォルト値:""
HWCheckInterval	RASPI_CHANGE_DETECT イベント検出のための GPIO リード間隔を設定する。 ミリ秒単位の整数を指定する。

	デフォルト値:10
BSCAutoGPIOAltMode	True を設定すると raspi_i2c_xxxx() 関数内で GPIO ピンの "alt"モードが自動設定されます。使用する I2Cバスと GPIO 番号は "Raspberry Piハードウエア API"の章を参照してください。 False を設定すると自動設定されないので、raspi_gpio_config() ライブラリ関数を事前にコールして、I2Cバスで使用する GPIOピンの "alt" モードを設定します。 デフォルト値: True
SPIAutoGPIOAltMode	True を設定すると raspi_spi_config() 関数内で GPIO ピンの "alt"モードが自動設定されます。使用する SPIポートと GPIO 番号は "Raspberry Piハードウエア API"の章を参照してください。 False を設定すると自動設定されないので、raspi_gpio_config() ライブラリ関数を事前にコールして、SPIポートで使用する GPIOピンの "alt" モードを設定します。 デフォルト値: True

11.2.11 NETSERVER

/Document/Class/NETSERVER: 汎用TCPサーバー(ModbusTCP)、汎用TCPクライアント、汎用UDPサーバー機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:True（この値は変更しないでください）
DetailLog	TCPサーバーや TCPクライアントで発生する詳細なログメッセージを、ログサーバーに出力するかどうかを指定する。True または False を指定する。 デフォルト値:True
TCPServerEnabled	汎用TCPサーバー機能を使用するかどうかを指定する。True または False を指定する。NETSERVER の他の設定項目を空白もしくはデフォルト値のまま、この項目を True に設定すると abs_agent は ModbusTCPサーバー(スレーブ)として動作します。 デフォルト値:False
TCPPort	汎用TCPサーバーで使用する TCP/IP ポート番号。TCPクライアントライブラリで、port パラメータ省略時の値としても使用される。整数を指定 (デフォルト設定は ModbusTCPとして使用可能な設定値になっています) デフォルト値:502
TCPFixedLen	TCPサーバーがバイナリデータの固定長部分のフレームを受信するときのバイト数。またこの項目は、TCPクライアントのバイナリモードで受信するリプライデータ長を -1 に指定した時の固定長部分のバイト数にも使用する。

	この項目はソケットからデータリード時に必ず読み込む固定フレーム部分のバイト数を指定している。整数を指定 (デフォルト設定は ModbusTCPとして使用可能な設定値になっています) デフォルト値:6
TCPVarLenIdx0 TCPVarLenIdx1 TCPVarLenIdx2 TCPVarLenIdx3	TCPサーバーがバイナリデータの可変長部分のフレームを受信するときのバイト数が格納されているオフセット位置を指定する。オフセット値は既に読み込み済みの固定長フレーム先頭からのバイト数になる。 TCPVarLenIdx0は可変長サイズ(32bit)の、bit0-7(1byte)部分を指定する TCPVarLenIdx1は可変長サイズ(32bit)の、bit8-15(1byte)部分を指定する TCPVarLenIdx2は可変長サイズ(32bit)の、bit16-23(1byte)部分を指定する TCPVarLenIdx3は可変長サイズ(32bit)の、bit24-31(1byte)部分を指定する TCPVarLenIdx0-3 で設定する項目に -1 を指定すると、このビット位置の可変長サイズ値を 0 として計算する。 TCPVarLenIdx0-3 までの全ての設定値を -1 にすると TCPFixedLen で指定された固定フレームのみを使用する。このとき TCPVarLenAuxSize 設定値は無視される。 また、TCPクライアントのバイナリモードで受信するリプライデータ長を -1 に指定した時の可変長部分のバイト数取得時にも同様に使用される。 整数を指定 (デフォルト設定は ModbusTCPとして使用可能な設定値になっています) デフォルト値: TCPVarLenIdx0:5 TCPVarLenIdx1:4 TCPVarLenIdx2:-1 TCPVarLenIdx3:-1
TCPVarLenAuxSize	TCPVarLenIdx0-3設定で取得した可変長サイズに追加で加えるバイト数を指定できる。整数を指定。 (デフォルト設定は ModbusTCPとして使用可能な設定値になっています) デフォルト値:0
UDPServerEnabled	汎用UDPサーバー機能を使用するかどうかを指定する。True または False を指定する。 デフォルト値:False
UDPPort	汎用UDPサーバーで使用する UDP/IP ポート番号。整数を指定 デフォルト値:8090

11.2.12 STORAGE

/Document/Class/STORAGE: データベース(パーマネントデータ)保存機能の設定項目

XML ドキュメントタグ名	説明
AutoOnline	abs_agent プログラム起動時に自動的にこのサービスモジュールを有効にする。True または False を指定する。 デフォルト値:Fase
DBSessionPool	Firebird RDBMS に接続する同時セッション数 デフォルト値:3
DBHostName	Firebird RDBMS サーバーが動作するコンピュータ名または IP アドレス デフォルト値:localhost
DBPath	データベースファイル名 絶対パス名の文字列を設定します。ディレクトリ名の中に $CONFIGDIR$ を記述するとその部分が、コンフィギュレーションファイルが配置されているディレクトリ名(絶対パス名)に置き換えられて解釈されます。 デフォルト値: $CONFIGDIR$/STOREDB.IB
DBUser	データベース接続時のユーザー名 デフォルト値:sysdba
DBPassword	データベース接続時のパスワード デフォルト値:masterkey

12 クライアントプログラム (CLI コマンドラインインターフェイス)

マニュアル中のコマンド実行形式部分の記述について

● コマンド実行時に指定するクライアントプログラム名には、abs_agent キットをインストールしたディレクトリからの相対パスまたはフルパス名を指定します。コマンド操作を行うシェルの PATH 環境変数に abs_agent インストールディレクトリを追加すると、クライアントプログラム名のみを指定して実行可能になります。

● コマンドパラメータが *[-r <hostname>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパラメータは省略可能です。

12.1 agent_stat (ステータス表示)

- **機能説明**

  abs_agent ステータス表示。

  ライセンス情報のインストールや、マスターファイルをエディタ等で直接編集した場合に、動作中の abs_agent に最新情報を反映させるときにも使用します。

- **コマンド実行形式**

agent_stat *[-l <license_file>] [-m] [-o <message>] [-c] [-r <hostname>]*

- **コマンドパラメータ**

-l <license_file>

<license_file> で指定したファイル中に格納されたライセンス情報を、abs_agent プログラムのサーバー設定ファイル(abs_agent.xml) に保存する。
ライセンスを購入した場合など、新しいライセンス情報を既存のサーバー設定ファイルに反映します。
新しいライセンス情報を有効にするために、abs_agent の再起動を行って下さい。

-m

マスターファイルをライブラリ関数やクライアントプログラムを経由せずに直接エディタで編集した場合に、動作中の abs_agent に最新のマスター情報を反映させます。

-o <message>

指定したメッセージ文字列を abs_agent が使用しているログサーバーに出力する。ログサーバー側で記録されるメッセージ中のモジュール名には "agent_stat(client)" が使用される。

-c

agent_stat コマンドを実行したコンピュータ自身のホスト名と MAC アドレス情報のみを表示する。このオプションは localhost または リモート側の abs_agent にアクセスしません。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバープログラムの動作ステータスを表示します。"-r <hostname>" オプションを指定するとリ

モートコンピュータで動作している abs_agent のステータスを取得することもできます。

agent_stat で得られるサーバー情報は、新規ライセンスを購入するときにも使用します。agent_stat コマンドの出力をコピーしてライセンス購入時の CPU 情報として指定します。agent_stat で表示される"Server MAC address" 項目のみがライセンス発行時に使用されます。詳しくはオールブルーシステムのホームページ (http://www.allbluesystem.com) を参照して下さい。

- **使用例**

使用する abs_agent と agent_stat プログラムのバージョンによって、表示されるステータス項目が増減する場合があります。

● localhost で実行中の abs_agent プログラムステータスを表示。

```
pi@raspi3:~/abs_agent$ ./agent_stat


--------------------------------------------------------------------------------
Program name       : ABSAgent
Version            : 1.00
Program folder     : /home/pi/abs_agent
Hardware type      : RASPI
Server start       : 2016/06/27 06:18:06
Server hostname    : raspi3
Server MAC address : B8-27-EB-D8-7C-2D
Licensed user      : test license
License time limit :
Service modules    : MASTERS SESSION SCRIPT CONVERT WEBPROXY MQTT SERIAL RASPI LAST
Client hostname    : raspi3
Client MAC address : B8-27-EB-D8-7C-2D
--------------------------------------------------------------------------------
```

● リモートで実行中の abs_agent プログラムステータスを表示。(リモートIPアドレス 192.168.100.14)

```
pi@raspi3:~/abs_agent$ ./agent_stat -r 192.168.100.14


--------------------------------------------------------------------------------
Program name       : ABSAgent
Version            : 1.00
Program folder     : /home/pi/fpsrc/abs_agent
```

```
Hardware type      : RASPI
Server start       : 2016/06/27 06:12:53
Server hostname    : raspberrypi
Server MAC address : B8-27-EB-A7-19-9A
Licensed user      : 開発用ライセンス
License time limit :
Service modules    : MASTERS SESSION SCRIPT CONVERT WEBPROXY MQTT SERIAL RASPI LAST
Client hostname    : raspi3
Client MAC address : B8-27-EB-D8-7C-2D
--------------------------------------------------------------------------------
```

● ファイル名 "license.xml" に格納されたライセンス情報を abs_agent サーバー設定ファイルに書き込む

```
pi@raspi3:~/abs_agent$ ./agent_stat -l license.xml

The new license has been installed, please restart the abs_agent program
to reflect the new configurations.

pi@raspi3:~/abs_agent$
```

- **参照**

abs_agent	abs_agent サーバープログラム
agent_hosts	リモートアクセスを許可するホストの管理プログラム

12.2 agent_hosts (リモートアクセス許可)

- **機能説明**

  リモートからアクセス可能なクライアントコンピュータの管理を行います。

- **コマンド実行形式**

  agent_hosts *[*-a <add_client_hostname>*]* *[*-d <delete_client_hostname>*]*

- **コマンドパラメータ**

  -a <add_client_hostname>

  <add_client_hostname> で指定したクライアントコンピュータからのリモートアクセスを許可する。ここで登録するホスト名はクライアントコンピュータ上で "agent_stat -c" コマンドで得られたホスト名を入力します。IP アドレスは指定できません。

-d <delete_client_hostname>

<delete_client_hostname> で指定したクライアントコンピュータを許可リストから削除する。削除対象のホスト名は、"agent_hosts" コマンドをオプション無しで実行したときに得られる許可済みのクライアントリスト中のエントリから指定します。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

リモートからアクセス可能なクライアントを管理します。リモートアクセスは、abs_agent クライアントプログラムを "-r <hostname>" オプション付きで実行した場合や、abs_agent の Lua スクリプト中からリモートアクセス用のライブラリ関数をコールしたときに発生します。

リモートアクセスされた abs_agent サーバー側では、リモートアクセス許可済みであるかをチェックして、もしアクセス許可されていないクライアントの場合にはエラーが発生します。

このコマンドで管理しているクライアントホスト名一覧は、abs_agent プログラムが動作しているコンピュータのマスターファイル "masters.xml" に登録されています。

agent_hosts コマンドは必ずローカルで動作している abs_agent プログラムを対象にします。

パラメータ無しで "agent_hosts" コマンドを実行すると、許可されているクライアント一覧が表示されます。許可するクライアントを追加するには "-a" オプションを使用します。反対に、現在許可済みのクライントをリストから削除する場合には "-d" オプションを使用します。

agent_hosts で指定するクライアントはホスト名または、MAC アドレスを指定できます。デフォルトはホスト名を使用するようになっています。クライアントホスト名はクライアントコンピュータ上に abs_agent プログラムをインストールして、"agent_stat -c" コマンドを実行することで表示させることができます。このとき、クライアントコンピュータの IP アドレスを "agent_hosts" コマンドのパラメータに指定することは出来ません。

Windows PC からリモートアクセスする場合のホスト名は、ABS-9000 LogServer がインストール済みの場合にはプログラムメニューから "AllBlueSystem" -> "CPU 情報表示" で確認できます。ABS-9000 LogServer をインストールしていない場合には、DOS プロンプト(または コマンドプロンプト) から "hostname" コマンドを実行して同様に確認することができます。

クライントを MAC アドレスで指定したい場合には、abs_agent のサーバー設定ファイル abs_agent.xml を変更します。サーバー設定ファイル(XML) 中の "<DocumentElement>/ServiceMain/UseMACProtection"タグの内容をエディタ等で修正して "True" にします。ファイル修正時には必ず UTF-8(BOM無し) でセーブするようにします。この後 abs_agent プログラムを再起動することで設定が有効になります。"agent_hosts" の "-a", "-d" オプションで指定するクライント名部分に "XX-XX-XX-XX-XX-XX" 形式の MAC アドレスを指定します。MAC アドレスはクライアントコンピュータ上で "agent_stat -c" コマンドで表示されたものを指定します。

クライアント認証を MAC アドレスで行う場合に、サーバーと直接 peer 接続している必要はありません

クライアント認証にホスト名ではなく MAC アドレスを使用することで、サーバーにアクセスするクライアントの認証を厳格にすることができます。(ただし、クライアントハードウエア入れ替え時にはサーバー側の設定を変更する必要があります) このとき、クライアントとサーバー間のネットークが、スイッチングハブやルータ装置、ゲートウエイを経由している場合でも、クライアント自身の持つ MAC アドレスを常に認証に使用します。このためPeer 接続している装置の MAC アドレスを確認する必要は一切ありません

フリー版ライセンス使用中の場合に有効なリモートアクセスクライアント数は１になります

スタンダードライセンスを使用している場合には、リモートアクセスを許可するクライアント数に制限はありません。ただし、フリー版ライセンスでは有効なエントリ数は１つのみに制限されます。

- **使用例**

● localhost で実行中の abs_agent プログラムで許可されているクライアント一覧を表示。

```
pi@raspi3:~/abs_agent$ ./agent_hosts

ServerName: raspi3    Trusted hosts: 1    Master: /home/pi/my_config/masters.xml
-------------------------------------------------------------------------
<client_hostname>
-------------------------------------------------------------------------
eagle
```

この例では "eagle" というホスト名からのリモートアクセスのみを受け付けます。

● "falcon" というホスト名からのリモートアクセスを許可する

```
pi@raspi3:~/abs_agent$ ./agent_hosts -a falcon
pi@raspi3:~/abs_agent$ ./agent_hosts

ServerName: raspi3    Trusted hosts: 2    Master: /home/pi/my_config/masters.xml
-------------------------------------------------------------------------
```

```
<client_hostname>
------------------------------------------------------------------------
eagle
falcon
```

● "falcon" というホスト名からのリモートアクセス許可を取り消す

```
pi@raspi3:~/abs_agent$ ./agent_hosts -d falcon
pi@raspi3:~/abs_agent$ ./agent_hosts

ServerName: raspi3    Trusted hosts: 1    Master: /home/pi/my_config/masters.xml
------------------------------------------------------------------------
<client_hostname>
------------------------------------------------------------------------
eagle
```

● リモートホスト "raspberrypi" に abs_agent をインストールして、そこからのリモートアクセスを MAC アドレス形式で許可する。サーバーは上記の例と同様にホスト名 "raspi3" で abs_agent が動作しているものとします。

(1) raspi3上で abs_agent のサーバー設定ファイルを変更して MAC アドレスでリモートアクセス許可を行う様に変更

```
pi@raspi3:~/abs_agent$ ./agent_shutdown    abs_agent シャットダウン
pi@raspi3:~/abs_agent$ vi abs_agent.xml
abs_agent.xml をエディタで編集して UseMACProtection を Trueに変更
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
  <Description>ABSAgent コンフィギュレーション</Description>
  <ServiceMain>
    <PortNumber type="integer">27101</PortNumber>
    <DefaultRemoteHost type="string">127.0.0.1</DefaultRemoteHost>
    <TimeStampMargin type="integer">0</TimeStampMargin>
    <AllowFileUpload type="boolean">True</AllowFileUpload>
    <UseMACProtection type="boolean">True</UseMACProtection>  この部分を True に変更
  </ServiceMain>
..
..
エディタを終了してファイルを更新する
```

```
pi@raspi3:~/abs_agent$ pwd
/home/pi/abs_agent
pi@raspi3:~/abs_agent$ sudo ./abs_agent -l 192.168.100.45      abs_agent 再起動
pi@raspi3:~/abs_agent$
```

(2) リモート側の raspberrypi 上で自身(クライアント側)の MAC アドレスを確認

```
pi@raspberrypi:~/abs_agent$ ./agent_stat -c

--------------------------------------------------------------------------------
Client hostname    : raspberrypi
Client MAC address : B8-27-EB-A7-19-9A
--------------------------------------------------------------------------------

pi@raspberrypi:~/abs_agent$
```

(3) 再び raspi3 上で、リモート側の MAC アドレスをリモートアクセス許可リストに追加する。このとき、(2) で得られた "Client MAC Address" の内容を使用します。

```
pi@raspi3:~/abs_agent$ ./agent_hosts -d eagle    (ホスト名形式のエントリは無効なので削除)
pi@raspi3:~/abs_agent$ ./agent_hosts -a B8-27-EB-A7-19-9A
pi@raspi3:~/abs_agent$ ./agent_hosts

ServerName: raspi3    Trusted hosts: 1    Master: /home/pi/my_config/masters.xml
-----------------------------------------------------------------------
<client_hostname>
-----------------------------------------------------------------------
B8-27-EB-A7-19-9A
```

- **参照**

  abs_agent　　abs_agent サーバープログラム

  agent_stat　　サーバーステータス表示（クライアントのホスト名、MAC アドレス取得）

  agent_shutdown abs_agent サーバープログラムシャットダウン

12.3　　agent_shutdown (サーバーシャットダウン)

- **機能説明**

  abs_agent サーバープログラムを停止させます。

- **コマンド実行形式**

agent_shutdown *[-r <hostname>]*

- **コマンドパラメータ**

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバープログラムのシャットダウンを実行します。このコマンド実行後に abs_agent プログラムでは各サービスモジュールの停止処理が実行されて、完全にシャットダウンが完了するまで数秒から 10秒程度かかります。

abs_agent が完全にシャットダウンするまで、新規の abs_agent サーバーを起動することはできません。abs_agent が現在起動中かどうかを調べるためには agent_stat コマンドを使用してください。abs_agent がシャットダウンしている場合には agent_stat コマンドはエラーを返します。

リモートに設置した abs_agent プログラムをシャットダウンする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります(agent_hosts コマンドの項を参照)。

OS のシャットダウン等、agent_shutdown コマンドを使用しないで強制的に abs_agent サーバープロセスを停止させることも可能です。このとき、abs_agent で実行中のタスク・スレッドは強制終了されます。abs_agent では、ユーザーが作成したスクリプト中からの明示的なファイル操作(マスターファイル書き込み、グローバル共有メモリエリアのファイル出力、Lua 標準 io ライブラリでの書き込み等)を実行している場合を除いて、ファイルシステムへの書き込みを行っていません。このため、abs_agent プロセスを強制終了しても不具合は発生しにくくなっています。

- **使用例**

● localhost で実行中の abs_agent プログラムをシャットダウン。

```
pi@raspi3:~/abs_agent$ ./agent_shutdown
```

バックグランドでデーモンタイプのスクリプトが実行中の場合や、Raspberry Pi H/W を操作しているスクリプトが実行中の場合には、それらのスクリプトを停止させてからシャットダウンします。"agent_task -K" コマンドと組み合わせて下記の様に実行してください。

```
pi@raspi3:~/abs_agent$ ./agent_task -K;./agent_shutdown
```

● リモートコンピュータ(IP アドレス 192.168.100.14) で実行中の abs_agent プログラムをシャットダウン。

```
pi@raspi3:~/abs_agent$ ./agent_shutdown -r 192.168.100.14
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
agent_stat	サーバーステータス表示
agent_hosts	リモートからのアクセス許可

12.4 agent_script (スクリプト実行)

- **機能説明**

  abs_agent サーバー上でスクリプトを実行する。

  スクリプト中でリターンパラメータを設定している場合はその値をコンソールに出力する。

- **コマンド実行形式**

  (スクリプトファイル一覧を表示)

  agent_script *[-f <result_file>] [-r <hostname>]*

  (スクリプトファイル実行)

  agent_script -s <script_name> *[-t] [-p <param_file>]*
  [-k <key#1,key#2,...,key#N> -v <val#1,val#2,...,val#N>]
  [-x <key> -y <val>]
  [-f <result_file>] [-r <hostname>]

- **コマンドパラメータ**

-s <script_name>

実行するスクリプト名を指定します。

指定可能なスクリプト名一覧は agent_script コマンドをパラメータ無しで実行すると得られます。スクリプト名は abs_agent のコンフィギュレーションファイルで指定したスクリプトフォルダのトップディレクトリからの相対パスになります。Lua スクリプトファイル・パス名のファイル拡張子(.lua) 部分を除いて指定します。

-t

別スレッドでスクリプトを実行します。abs_agent プログラムが動作しているコンピュータ上で別スレッドでスクリプトが実行され、agent_script コマンド自身は直ぐに終了します。
スクリプト内で無限ループになる様なデーモンタイプのスクリプトを実行する場合には、必ず "-t" オプションを指定(または API script_fork_exec(),script_fork_rexec()を使用)して別スレッドでスクリプト実行を開始させてください。

このオプションを指定すると、スクリプト中で指定したリターンパラメータは受け取れません。別スレッドで実行するスクリプトから他のスクリプトにデータを渡したい場合にはグローバル共有変数等を利用できます。

現在別スレッドで実行中のスクリプトは agent_task コマンドで確認できます

-p <param_file>

スクリプト実行時に <param_file> で指定した複数のリクエストパラメータを渡します。スクリプト中からは g_params[] テーブルのキーとその値をアクセスすることでパラメータを取得できます。<param_file> はテキストファイル形式で作成します。

-p パラメータは、-k, -v, -x, -z パラメータ等と同時に使用することで、追加のリクエストパラメータをスクリプトに渡すこともできます。

第一カラムが '#' 文字の場合には行全体をコメント行として読み飛ばします。

１行につき１パラメータで、パラメータのキーと値を "=" でつなげた形で記述します。キーや値に日本語を使用する場合は、必ず UTF-8(BOM無し)エンコード形式で保存して下さい。（下記参照）

```
# コメント行
Key#1=Value#1
Key#2=Value#2
..
Key#n=Value#n
```

-p パラメータは、-k, -v, -x, -y パラメータ等と同時に使用することで、追加のリクエストパラメータをスクリプトに渡すこともできます。

-k <key#1,key#2,...,key#N> -v <val#1,val#2,...,val#N>

スクリプトパラメータを指定する。"-k" オプションでキー名を指定して -v オプションで値を指定します。キー名とそれに対応する値をカンマ区切りで複数指定することもできます。この時にはキー名と値の並びは同じにしてください。

<key#n> と <val#n> で指定する文字列中に "," カンマを含めることはできません。カンマはリクエストパラメータ区切りとして解釈されます。キーや値にカンマ文字を含めたい場合には -x, -y パラメータを使用します。

-k, -v パラメータは、-p, -x, -y パラメータ等と同時に使用することで、追加のリクエストパラメータをスクリプトに渡すことができます。

-x <key> -y <val>

スクリプトパラメータを指定する。"-x" オプションでキー名を指定して -y オプションで値を指定します。<key> と <val> で指定する文字列中に "," カンマを含めることができます。

-x, -y パラメータは、-p, -k, -v パラメータ等と同時に使用することで、追加のリクエストパラメータをスクリプトに渡すことができます。

-f <result_file>

スクリプト実行時に、スクリプト中でリターンパラメータを指定している場合にはコンソールにリターンパラメータのキーと値のペア一覧が表示されます。"-f" オプションを指定すると、これらのリターンパラメーター覧を <result_file> で指定したファイルに書き込みます。また、agent_script コマンドを "-s <script_name>" オプション無しで実行したときに表示されるスクリプト名一覧も、同様に <result_file> に出力されます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

パラメータで指定したスクリプトを abs_agent で実行します。

実行可能なスクリプト一覧は agent_script コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

実行するスクリプトの記述が無限ループに入るようなデーモンタイプになっている場合や、スクリプト実行に時間がかかる場合には "-t" オプションを指定することで別スレッドで実行させることができます。このとき、スレッドは abs_agent が動作しているコンピュータ側に作成されて、agent_script コマンドは直ぐに終了します。

実行する前にスクリプトの内容を確認したい場合には "agent_get -s <script_name>" コマンドを実行して、スクリプトの内容をコンソールに出力できます。スクリプトを新規に作成したり内容を修正したい場合には、"agent_get", "agent_put" コマンドを使用して、ローカルまたはリモートコンピュータ上のスクリプトファイルをダウンロード or 新規作成、編集、アップロードすることができます。詳しくは各クライアントプログラムの項を参照してください。

現在 abs_agent サーバーで実行中のスクリプト一覧を表示するためには、"agent_task" コマンドを使用してください。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") で PARAM_ECHO スクリプトを実行する。PARAM_ECHO スクリプトの内容を事前に画面に表示した後、パラメータキー "キー１" パラメータ値 "値１" を指定してスクリプトを実行する。

```
pi@raspi3:~/abs_agent$ ./agent_get -s PARAM_ECHO

file_id = "PARAM_ECHO"

-----------------------------------------------
-- スクリプトパラメータに指定された値をそのまま
```

```
-- スクリプトのリターン値にセットして返す
-----------------------------------------------


log_msg(string.format("モジュール開始時刻 %s ホスト名 %s リクエスト
元 %s",g_startup,g_hostname,g_sender),file_id)


for key,val in pairs(g_params) do
    log_msg(string.format("g_params[%s] = %s",key,val),file_id)
    if not script_result(g_taskid,key,val) then error() end
end


pi@raspi3:~/abs_agent$ ./agent_script -s PARAM_ECHO -k キー１ -v 値１


ServerName: raspi3      ResultParam(s): 1
-------------------------------------------------------------------------
<Key>=<Value>
-------------------------------------------------------------------------
キー１=値１


pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") で PARAM_ECHO スクリプトを実行する。リクエストパラメータを２つ指定する。

```
pi@raspi3:~/abs_agent$ ./agent_script -s PARAM_ECHO -k キー1,キー2 -v 値1,値2


ServerName: raspi3      ResultParam(s): 2
-------------------------------------------------------------------------
<Key>=<Value>
-------------------------------------------------------------------------
キー2=値2
キー1=値1


pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）で PARAM_ECHO スクリプトを実行。リクエストパラメータを２つ指定する。

```
pi@raspi3:~/abs_agent$ cat > my_param << EOF
```

```
> キー1=値1
> キー2=値2
> EOF
pi@raspi3:~/abs_agent$ cat my_param
キー1=値1
キー2=値2
pi@raspi3:~/abs_agent$ ./agent_script -s PARAM_ECHO -p my_param -r 192.168.100.14

ServerName: raspberrypi      ResultParam(s): 2
---------------------------------------------------------------------
<Key>=<Value>
---------------------------------------------------------------------
キー1=値1
キー2=値2

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

● localhost(サーバー名 "raspi3") で無限ループに入る LOOP_TASK スクリプトを実行します。最初にスクリプト一覧から LOOP の名前の付いたスクリプトを検索した後、画面にスクリプトの内容を表示します。その後、別スレッドで LOOP_TASK スクリプトを実行します。タスク一覧コマンド agent_task で実行中の LOOP_TASK スクリプトとアサインされた <TaskID> を確認した後、"agent_task -k <TaskID>" コマンドでスクリプトを強制終了します。

```
pi@raspi3:~/abs_agent$ ./agent_script | grep LOOP
TEST/LOOP_TASK
pi@raspi3:~/abs_agent$ ./agent_get -s TEST/LOOP_TASK

--[[
このスクリプトは無限ループのため終了しません。
必ず別スレッドでスクリプトを起動してください。
スクリプトを終了させる場合には "agent_task -k <TaskID>" または、
script_kill() ライブラリ関数を使用します。
]]

log_msg("start..",g_script)
while true do
        wait_time(100)
```

```
        log_msg("task running..." .. g_taskid,g_script)
end


pi@raspi3:~/abs_agent$ ./agent_script -t -s TEST/LOOP_TASK
pi@raspi3:~/abs_agent$ ./agent_task

ServerName: raspi3      Active task(s): 2   Total: 16
------------------------------------------------------------------------
<start>                  <task_id>           <script_name>
------------------------------------------------------------------------
2016/07/01 17:57:53      LUA041C31E75E3AFE    TEST/LOOP_TASK
2016/06/30 13:42:27      LUA041BCDA9B7BC7F    TEST/RASPI_HC595_COUNTER

pi@raspi3:~/abs_agent$ ./agent_task -k LUA041C31E75E3AFE
pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "debian", IP アドレス 192.168.100.11）で実行中の abs_agent プログラムに格納されたスクリプト SHUTDOWN を使用してリモートコンピュータをシャットダウンします。最初にスクリプト一覧から OS フォルダの名前の付いたスクリプトを検索した後、画面に SHUTDOWN スクリプトの内容を表示します。その後、SHUTDOWN スクリプトを実行してリモートコンピュータを停止させます。

```
pi@raspi3:~/abs_agent$ ./agent_stat -r 192.168.100.11


------------------------------------------------------------------------------------------
Program name       : ABSAgent
Version            : 1.00
Program folder     : /home/satoshi/abs_agent
Hardware type      : X86
Server start       : 2016/07/01 17:29:24
Server hostname    : debian
Server MAC address : 00-0C-29-20-11-7A
Licensed user      : フリー版(業務・商用利用はできません)
License time limit :
Service modules    : MASTERS SESSION SCRIPT CONVERT WEBPROXY MQTT SERIAL LAST
Client hostname    : raspi3
Client MAC address : B8-27-EB-D8-7C-2D
------------------------------------------------------------------------------------------


pi@raspi3:~/abs_agent$ ./agent_script -r 192.168.100.11 | grep OS
```

```
OS/REBOOT
OS/SHUTDOWN
pi@raspi3:~/abs_agent$ ./agent_get -s OS/SHUTDOWN -r 192.168.100.11

file_id = "SHUTDOWN"

--[[
*******************************************************************************
OS (Linux) のシャットダウンを実行する
*******************************************************************************
]]

log_msg("start..",file_id)
os.execute('/sbin/shutdown -h now &')

pi@raspi3:~/abs_agent$ ./agent_script -s OS/SHUTDOWN -r 192.168.100.11

ServerName: debian      ResultParam(s): 0
------------------------------------------------------------------------
<Key>=<Value>
------------------------------------------------------------------------

pi@raspi3:~/abs_agent$ ping 192.168.100.11
PING 192.168.100.11 (192.168.100.11) 56(84) bytes of data.
^C
--- 192.168.100.11 ping statistics ---
19 packets transmitted, 0 received, 100% packet loss, time 18008ms

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
agent_stat	サーバーステータス表示
agent_hosts	リモートからのアクセス許可
agent_get	スクリプトダウンロード
aget_put	スクリプトアップロード
agent_task	実行中のスクリプトタスク一覧表示、スクリプト強制終了

12.5 agent_data (グローバル共有変数操作)

- **機能説明**

abs_agent サーバーのグローバル共有変数参照や更新を行います。

- **コマンド実行形式**

agent_data *[-d] [-D] [-i] [-k <key>] [-v <value>] [-p <data_file>] [-f <result_file>] [-r <hostname>]*

- **コマンドパラメータ**

-d, -D

グローバル共有変数を削除します。削除対象のグローバル共有変数は "-k <key>" オプションで指定します。

"-d" は "-k <key>" で指定したキー名に完全一致するデータ(一つ)を削除します。"-D" を指定すると "-k <key>" で指定したキー名に前方一致するデータ(複数)を全て削除します。

-i

グローバル共有変数を数値に変換して値に1を足した値で更新します。更新後のグローバル共有変数は文字列形式になります。更新対象のグローバル共有変数は "-k <key>" オプションで指定します。グローバル共有変数が見つからないか、既存の値が数値に変換できない場合には "1" が設定されます。abs_agent のライブラリ関数 inc_shared_data(), inc_net_data() と同等の動作になります。

-k <key>

グローバル共有変数のキー名を指定します。"-i", "-d", "-v" オプションを使用するときに、対象となるキー名を指定します。"-D" オプションの時はキー名の(前方一致部分の)部分文字列を指定します。"-k" オプションのみを指定した場合には、現在のグローバル共有変数の値がコンソールに出力されます。

-v <value>

<value> に指定した値でグローバル共有変数の値を更新します。更新対象のグローバル共有変数は "-k" オプションで指定します。

-p <data_file>

複数のグローバル共有変数を更新します。更新対象のキーと値のペアを <data_file> で指定したファイル中に記述します。<data_file> はテキストファイル形式で作成します。１行につき１共有変数で、キーと値を "=" でつなげた形で記述します。

"=" 文字の右側の値を空にして "Key1=" の様に記述すると "Key1" のグローバル共有変数を削

除します。（ -d オプションと同等）

第一カラムが '#' 文字の場合には行全体をコメント行として読み飛ばします。

キーや値に日本語を使用する場合は、必ず UTF-8(BOM無し)エンコード形式で保存して下さい。（下記参照）

```
# コメント行
Key#1=Value#1
Key#2=Value#2
..
Key#n=Value#n
```

更新対象の共有変数が１つの場合には、"-k", "-v" オプションを使用することもできます。

-f <result_file>

グローバル共有変数を表示させるとコンソールにキーと値のペアー覧が表示されます。"-f" オプションを指定すると、これらのグローバル共有変数一覧を <result_file> で指定したファイルに書き込みます。現在使用中のグローバル共有変数を全て取得する場合には agent_data コマンドを "-k <key>" オプション無しで実行します。<result_file> の出力フォーマットは、グローバル変数更新時に指定する "-p<data_file>" と同一のフォーマットです。このため、サーバーで現在使用中の複数のグローバル共有変数を更新したり、他のサーバーにコピーするときにも使用できます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーで使用中のグローバル共有変数を参照したり値を更新することができます。abs_agent のスクリプトやイベントハンドラからグローバル共有変数ライブラリ関数で使用するのと同等の動作です。

agent_data コマンドを使用して、GLOBAL_WATCH イベントで監視対象にしているグローバル共有変数を更新した場合には GLOBAL_WATCH イベントが発生します。ただし、-p<data_file> オプションを使用してグローバル共有変数を更新した場合には、GLOBAL_WATCH イベントは発生しません。

Linux のシェルで特別な解釈が行われる '$'等の文字をパラメータに指定する場合には、"'" シングルコート文字でパラメータ文字列を囲んで下さい。

現在使用中の全てのグローバル共有変数一覧は agent_data コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

グローバル共有変数の値をコンソールに出力する時に、データ値が IEEE-754 64ビット倍精度浮動小数点または 32ビット単精度浮動小数点のバイト列の16進数文字列の場合には、データ値の右側に浮動少数点値に変換した値を "()" 内に表示します。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") のグローバル共有変数一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_data

ServerName: raspi3      SharedData: 2
--------------------------------------------------------------------------
<Key>=<Value>
--------------------------------------------------------------------------
TEST/RASPI_HC595_COUNTER_RUNNING=1
HC595_COUNTER=250

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") のキー名 "MY_DATA" グローバル共有変数に値 "試験データ" を書き込む。

```
pi@raspi3:~/abs_agent$ ./agent_data -k MY_DATA -v 試験データ
```

```
pi@raspi3:~/abs_agent$ ./agent_data -k MY_DATA
MY_DATA=試験データ
pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）のグローバル共有変数一覧をローカルファイル data1 に出力する。その後、data1 ファイルを使用して localhost(サーバー名 "raspi3"）のグローバル共有変数に書き込む。

```
pi@raspi3:~/abs_agent$ ./agent_data -r 192.168.100.14 -f data1
pi@raspi3:~/abs_agent$ cat data1
共有変数キー1=共有変数値1
共有変数キー2=共有変数値2
共有変数キー3=共有変数値3
pi@raspi3:~/abs_agent$ ./agent_data -p data1
pi@raspi3:~/abs_agent$ ./agent_data

ServerName: raspi3      SharedData: 6
------------------------------------------------------------------------
<Key>=<Value>
------------------------------------------------------------------------
TEST/RASPI_HC595_COUNTER_RUNNING=1
MY_DATA=試験データ
HC595_COUNTER=86
共有変数キー1=共有変数値1
共有変数キー2=共有変数値2
共有変数キー3=共有変数値3

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
agent_hosts	リモートからのアクセス許可
BACKUP_RESTORE	グローバル共有データをファイルにバックアップしたりリストアするためのスクリプト

12.6 agent_strlist (グローバル共有文字列リスト操作)

- **機能説明**

abs_agent サーバーのグローバル共有文字列リストの参照や更新を行います。

- **コマンド実行形式**

agent_strlist *[-a] [-c <channel> [-d] [-t] [-b] [-l] [-i <add_data>] [-m <remove_data>]]*
[-p <data_file> [-q]] [-f <result_file>] [-r <hostname>]

- **コマンドパラメータ**

-a

全ての文字列リスト(Channel)に格納されている文字列エントリを全て出力(参照)します。

データ数が 多い場合には "-c <channel>" オプションを使用して文字列リスト(Channel)毎にデータを出力 するようにしてください。使用中の文字列リスト名(Channel名)一覧は agent_strlist をオプション無しで実行すると得られます。

文字列リスト(Channel)内に文字列エントリが１つも登録されていない場合には、該当チャンネルのデータは出力されません。

-c <channel>

文字列リストを操作するチャンネル名を指定します。このオプションのみを指定した場合には、指定したチャンネルに格納されている全ての文字列エントリを出力(参照のみ)します。文字列リストに新規に文字列データを追加したり、データを取り出す等の操作を行う場合には、"-c <channel>" オプションと共に、操作を行いたい機能毎のオプションを指定します。

-t

"-c <channel>" オプションで指定した文字列リストの先頭(最も過去に登録された)データを取り出します。取り出したデータは文字列リストから取り除かれます。

-b

"-c <channel>" オプションで指定した文字列リストの最後(最も最近に登録された)データを取り出します。取り出したデータは文字列リストから取り除かれます。

-d

"-c <channel>" オプションで指定した文字列リスト自身と、その文字列リストに格納されている全データを削除します。

-i <add_data>

"-c <channel>" オプションで指定した文字列リストの最後に <add_data> で指定した文字列を追加します。

-m <remove_data>

"-c <channel>" オプションで指定した文字列リストから、<remove_data> で指定した文字列と一致するデータを取り除きます。文字列リスト中に一致するデータが複数ある場合には、それらを全て削除します。

-p <data_file> [-q]

複数の文字列リストデータを追加します。更新対象の文字列リスト名と追加データのペアを <data_file> で指定したファイル中に記述します。<data_file> はテキストファイル形式で作成します。１行につき１つの文字列リスト名(Channel)と１つのデータを "=" でつなげた形で記述します。

"-q" オプションを "-p"と同時に指定した場合には、データファイル中で指定された全ての文字列リストを abs_agent から削除した後、データファイルからデータがロードされます。"-q" オプションを指定しない場合には、abs_agent の既存の文字列リストにデータに追加する形で、データがロードされます。

文字列リスト(Channel)毎にファイルに指定されたデータが追加されていきます。文字列リスト名(Channel)は <data_file>ファイル中で行を連続して並べる必要はありませんが、文字列リスト毎のデータを追加する順序は、ファイル中で指定したデータ行の順序と一致します。

第一カラムが '#' 文字の場合には行全体をコメント行として読み飛ばします。キーや値に日本語を使用する場合は、必ず UTF-8(BOM無し)エンコード形式で保存して下さい。（下記参照）

```
# コメント行
Channel#1=Data#1
Channel#1=Data#2
..
Channel#2=Data#1
Channel#2=Data#2
```

-f <result_file>

グローバル共有文字列リストのデータを表示させると、コンソールに文字列リスト名(Channel)とデータのペアが表示されます。"-f" オプションを指定すると、これらのグローバル共有文字列リスト中のデーター覧を <result_file> で指定したファイルに書き込みます。

abs_agent に現在登録されている全てのグローバル共有文字列リスト中のデータエントリを取得する場合には、agent_strlist コマンドを "-a" オプション付きで実行します。

指定した文字列リスト(Channel) に登録されたデータを表示する場合には "-c <channel>" オプション付きで実行します。

<result_file> の出力フォーマットは、グローバル共有文字列リストのデータをファイルからロードするときに指定する "-p <data_file>" と同一のフォーマットです。このため、abs_agent で現在使用中のグローバル共有文字列リストを更新したり、他のサーバーに文字列リストをコピーするときにも使用できます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーで使用中のグローバル共有文字列リストの参照や更新を行います。abs_agent のスクリプトやイベントハンドラからグローバル共有文字列リスト用のライブラリ関数で操作するのと同等の機能です。

現在使用中の全てのグローバル共有文字列リストのチャンネル一覧は agent_strlist コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

指定したチャンネルに格納済みのエントリを取得する場合には "-c <channel>" オプションを付けて実行します。

文字列リストのエントリ値をコンソールに出力する時に、エントリ値が IEEE-754(64ビット倍精度浮動小数点)形式の16進数文字列の場合には、エントリ値の右側に浮動少数点値に変換した値を "()" 内に表示します。

Linux のシェルで特別な解釈が行われる '$'等の文字をパラメータに指定する場合には、"'" シングルコート文字でパラメータ文字列を囲んで下さい。例: agent_strlist -c '$GLOBAL_WATCH'

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。

このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") のグローバル共有文字列リスト名(Channel名)一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_strlist

ServerName: raspi3      Channels: 9
------------------------------------------------------------------------
<channel>
------------------------------------------------------------------------
$GLOBAL_WATCH
/xbee/Device4/tdcp
Device4_temperature
Device4_ir_count
/zb/Node1/tdcp
Node1_temperature
Node1_pressure
Node1_humidity
Node1_light

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") で、文字列リスト名 "試験Ch" に "データ1"と "データ2"を追加する。

```
pi@raspi3:~/abs_agent$ ./agent_strlist -c '試験ch' -i 'データ1'
pi@raspi3:~/abs_agent$ ./agent_strlist -c '試験ch' -i 'データ2'
pi@raspi3:~/abs_agent$ ./agent_strlist -c '試験ch'

ServerName: raspi3      Data: 2
------------------------------------------------------------------------
<channel>=<data>
------------------------------------------------------------------------
試験ch=データ1
試験ch=データ2

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") のグローバル共有文字列リストの文字列リスト名 "試験Ch" の全データをローカルファイル list1 に出力する。その後、list1 ファイルを使用してリモートコンピュータ(サーバー名 "raspberrypi", IP アドレス 192.168.100.14)に文字列リスト名 "試験Ch"を登録する。

```
pi@raspi3:~/abs_agent$ ./agent_strlist -c '試験ch' -f list1
pi@raspi3:~/abs_agent$ cat list1
#
# agent_strlist [2017/01/07 15:48:55]
#
試験ch=データ1
試験ch=データ2
pi@raspi3:~/abs_agent$ ./agent_strlist -r 192.168.100.14 -p list1
pi@raspi3:~/abs_agent$ ./agent_strlist -r 192.168.100.14 -c '試験ch'

ServerName: raspberrypi      Data: 2
------------------------------------------------------------------------
<channel>=<data>
------------------------------------------------------------------------
試験ch=データ1
試験ch=データ2

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent　　abs_agent サーバープログラム

agent_hosts　　リモートからのアクセス許可

BACKUP_RESTORE　グローバル共有文字列リストをファイルにバックアップしたりリストアするためのスクリプト

12.7　　agent_task (実行中スクリプトのタスク管理)

- **機能説明**

abs_agent サーバーで実行中のスクリプトやイベントハンドラの管理を行います。

- **コマンド実行形式**

agent_task *[-k <taskid_or_name>[-v <signal>]] [-K] [-a] [-r <hostname>]*

- **コマンドパラメータ**

-k <taskid_or_name>

実行中のスクリプトを強制終了または、-v<signal> パラメータで指定したシグナルを設定します。<taskid_or_name> には タスクID 文字列またはスクリプト名を指定します。タスクID 文字列は agent_task コマンドのタスク一覧や script_fork_exec()ライブラリ関数のリターン値で返されるタスクID 文字列を指定します。

-k パラメータでスクリプトを終了させる場合には -v <signal> パラメータを省略するか若しくは、<signal> 値に 9 を指定("-v 9")します。

<taskid_or_name> にスクリプト名を指定した場合は、スクリプト名に一致する実行中のタスクにシグナルが設定されます。同一スクリプトが複数実行中の場合には、それら全てにシグナルが設定されます。スクリプト名に '*' を指定した場合はabs_agent 内で実行中の全タスクが対象となります。

-v <signal>

-k<taskid_or_name> パラメータと同時に使用して、実行中のスクリプトにシグナルを設定します。<signal> には 1 から 255 までの整数値を指定します。シグナルを受信した実行中のスクリプトでは g_signal 変数が設定されます。

シグナル値に 1 から 8 までの値を指定した場合はログメッセージも同時に出力されます。10 以上 255 以下の値を指定した場合はロメッセージは発生しません。9 を指定した場合にはスクリプトが強制終了します。パラメータ省略時は 9 が指定されます。

-K

実行中の全スクリプトを強制終了させます。

-a

サーバー設定ファイルで指定された全てのスクリプトプールのエントリを表示します。
現在未使用(未実行)のエントリの <task_id>,<script_name> 項目は空白になります。 <start> 項目が "1899/12/30 00:00:00" のエントリは abs_agent 起動後、一度も使用されていない事を示します。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーで現在実行中の Lua スクリプト一覧を取得できます。agent_task コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールにスクリプト名とタスクID 一覧が表示されます。

実行中のイベントハンドラやユーザースクリプトを全て表示しますが、各々のイベントハンドラやユーザースクリプトは起動から終了までの期間が短いため、ユーザーが明示的に作成した無限ループに入るデーモンタイプのスクリプトのみが主に表示されます。

"-k <taskid_or_name>" で現在実行中のスクリプトを終了させることができます。強制的にスクリプト実行が中断されて、ログには下記のエラーメッセージが記録されます。

```
2016/07/03 09:50:07 raspi3      LUASessionPool      0 ScriptHookFunc:*WARNING* signal[9] received, g_taskid = LUA041C69C75D6D92
2016/07/03 09:50:07 raspi3      SCRIPT              0 TForkScriptExecThread:*EXCEPTION* lua_pcall() runtime error
```

"agent_task -k <taskid_or_name>" でスクリプトを終了させるときに、対象のスクリプトが Lua インストラクションを実行していない場合には強制終了を関知できません。例えば wait_time() で長時間待ちの状態になっている場合や、OS システムコールの処理待ちの場合などが該当します。この場合でもライブラリ関数を抜けて Lua インストラクションを実行するようになると強制終了されます。長期間、強制終了が出来ない状態を防ぐ為には、Lua スクリプトで作成したループ構造と wait_time()ライブラリコールを組み合わせて長いウェイトを実現したり、時間のかかる OS コマンドをバックグランド("&"を付けて実行)で処理するようにしてください。

長時間のスリープ中にも中断シグナルを受け付けるようにする例：

```
function my_wait(ms10)
  for i = 1,ms10,1 do
    wait_time(10)
  end
end
-- 10*1000(ms) => 10(sec)のスリープ
my_wait(1000)
```

"agent_task -a" を実行するとサーバー設定ファイル(abs_agent.xml)で指定されたスクリプトプールの全エントリを表示します。
<start> 項目が "1899/12/30 00:00:00" のエントリは一度も使用されていない事を示します。もしこのエントリ数が合計 4以下の場合には、ユーザースクリプトやイベントハンドラ実行時にスクリプトプールが逼迫する

可能性が高くなります。この場合はサーバー設定ファイル(abs_agent.xml) 中のスクリプトプール・セッション数を増やす必要があります。詳しくは "サーバープログラム設定ファイル" の章や、"script_free_count()" API の項を参照して下さい。

```
pi@mercury:~/abs_agent$ ./agent_task -a

ServerName: mercury      Total: 16
------------------------------------------------------------------------
<start>                  <task_id>           <script_name>
------------------------------------------------------------------------
2021/10/04 15:31:46
2021/10/04 15:31:44
2021/07/27 15:35:24      LUA055F6F8D85C485    TEST/GET_MAIL_NEWS_HEADERS
2021/10/04 15:31:24
2021/10/04 14:56:24
2021/10/01 18:45:24
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00
1899/12/30 00:00:00

pi@mercury:~/abs_agent$
```

(agent_task -a 実行例)

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") で実行中のスクリプト・タスク一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_task
```

```
ServerName: raspi3      Active task(s): 1   Total: 16
-------------------------------------------------------------------------
<start>                 <task_id>           <script_name>
-------------------------------------------------------------------------
2016/06/30 13:42:27     LUA041BCDA9B7BC7F   TEST/RASPI_HC595_COUNTER

pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）上で、無限ループに入るテストスクリプト TEST/LOOP_TASK を別スレッドで起動する。その後、タスク一覧からスクリプトのタスクID を確認して強制終了させる。

```
pi@raspi3:~/abs_agent$ ./agent_script -t -s TEST/LOOP_TASK -r 192.168.100.14
pi@raspi3:~/abs_agent$ ./agent_task -r 192.168.100.14

ServerName: raspberrypi      Active task(s): 1   Total: 16
-------------------------------------------------------------------------
<start>                 <task_id>           <script_name>
-------------------------------------------------------------------------
2016/07/03 10:16:59     LUA041C6A43B2E760   TEST/LOOP_TASK

pi@raspi3:~/abs_agent$ ./agent_task -k LUA041C6A43B2E760 -r 192.168.100.14
pi@raspi3:~/abs_agent$ ./agent_task -r 192.168.100.14

ServerName: raspberrypi      Active task(s): 0   Total: 16
-------------------------------------------------------------------------
<start>                 <task_id>           <script_name>
-------------------------------------------------------------------------

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
abs_script	スクリプト実行
agent_hosts	リモートからのアクセス許可

12.8 agent_get (スクリプト参照・ダウンロード)

- **機能説明**

abs_agent サーバーの指定したスクリプトをダウンロードします。

- **コマンド実行形式**

agent_get -s <script_name> *[-f <output_file>] [-r <hostname>]*

- **コマンドパラメータ**

-s <script_name>

ダウンロードするスクリプト名を指定します。スクリプト名は agent_script コマンドをパラメータ無しで実行したときに得られる、スクリプト名リストの中から指定できます。スクリプト名は abs_agent が参照しているスクリプトフォルダのトップディレクトリからの相対パスになります。また Lua スクリプトを示すファイル拡張子(.lua) 部分は除いて指定します。

-f <output_file>

ダウンロードしたスクリプトをコンソールに表示する代わりに、"-f <output_file>" オプションで指定したファイルに書き込みます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーに設定されているスクリプトを表示します。指定するスクリプト名は agent_script コマンドで得られるスクリプト一覧の中から指定します。"-r" オプションを指定してリモートコンピュータからスクリプトを取得することもできます。

agent_script コマンドでスクリプト実行する前にスクリプト内容を確認したいときに使用します。また、ローカルやリモートコンピュータ上のスクリプトを編集したいときに、作業用のファイルとしてダウンロードするときにも使用できます。作業用にダウンロードしたファイルは編集の後、agent_put コマンドを使用することでアップロード(上書き)できます。

OS 上に配置された Lua スクリプトファイル(*.lua) を直接参照するよりも、agent_get コマンドを使用する方が便利です

agent_get コマンドを使用しなくても、abs_agent をインストールした先に保存されているスクリプトファイルを直接参照することも可能です。この時、スクリプトを保存しているトップディレクトリが abs_agent.xml サーバー設定ファイル中に記述されていますので、必ずそのディレクトリ内のファイルにアクセスするようにします。スクリプト保存先はサーバー毎に設定が異なっている可能性がありますので、agent_get コマンドを使用するほうが簡単に操作できます。スクリプトのダウンロード(agent_get)、アップロード(agent_put)、実行(agent_script)で指定するスクリプト名パラメータは共通で "-s <script_name> になっています。このため、Linux シェルのヒストリ編集機能を使うと効率よく作業を行えます。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") の SAMPLE スクリプトの内容をコンソールに表示。

```
pi@raspi3:~/abs_agent$ ./agent_get -s SAMPLE


file_id = "SAMPLE"


--[[
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
********************************************************************************
]]


log_msg("start..",file_id)
log_msg(string.format("モジュール開始時刻 %s ホスト名 %s リクエスト
元 %s",g_startup,g_hostname,g_sender),file_id)
log_msg(string.format("スクリプトに渡されたパラメーター覧"),file_id)
```

```
for key,val in pairs(g_params) do
    log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end


pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）上に保存されている TEST/LOOP_TASK スクリプトの内容をコンソールに表示

```
pi@raspi3:~/abs_agent$ ./agent_get -s TEST/LOOP_TASK -r 192.168.100.14


--[[
このスクリプトは無限ループのため終了しません。
必ず別スレッドでスクリプトを起動してください。
スクリプトを終了させる場合には "agent_task -k <TaskID>" または、
script_kill() ライブラリ関数を使用します。
]]


log_msg("start..",g_script)
while true do
        wait_time(100)
        log_msg("task running..." .. g_taskid,g_script)
end


pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

● localhost(サーバー名 "raspi3") の SAMPLE スクリプトの内容の内容を work_file にダウンロードします work_file の内容を編集してリターンパラメータにメッセージを出力するように変更します。その後、work_file をリモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）にスクリプト名 TEST/MY_SAMPLE として保存した後、スクリプトを実行します。

```
pi@raspi3:~/abs_agent$ ./agent_get -s SAMPLE -f work_file
pi@raspi3:~/abs_agent$ cat >> work_file << EOF
> script_result(g_taskid,"Message","リモートからのメッセージ")
> EOF
pi@raspi3:~/abs_agent$ cat work_file
```

```
file_id = "SAMPLE"
--[[
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
********************************************************************************
]]

log_msg("start..",file_id)
log_msg(string.format("モジュール開始時刻 %s ホスト名 %s リクエスト
元 %s",g_startup,g_hostname,g_sender),file_id)
log_msg(string.format("スクリプトに渡されたパラメーター覧"),file_id)
for key,val in pairs(g_params) do
    log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end

script_result(g_taskid,"Message","リモートからのメッセージ")
pi@raspi3:~/abs_agent$ ./agent_put -f work_file -s TEST/MY_SAMPLE -r 192.168.100.14
pi@raspi3:~/abs_agent$ ./agent_script -s TEST/MY_SAMPLE -r 192.168.100.14

ServerName: raspberrypi      ResultParam(s): 1
--------------------------------------------------------------------------
<Key>=<Value>
--------------------------------------------------------------------------
Message=リモートからのメッセージ

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
agent_script	スクリプト実行
agent_put	スクリプトアップロード

12.9 agent_put (スクリプト作成・アップロード・削除)

- **機能説明**

abs_agent サーバーに、指定したスクリプトをアップロードします。サーバー上のスクリプトファイルを削除することもできます。

- **コマンド実行形式**

agent_put -s <script_name> *[-f <input_file>] [-d] [-r <hostname>]*

- **コマンドパラメータ**

-f <input_file>

アップロードする Lua スクリプトが記述されたローカルファイルを指定します。ファイルは UTF-8(BOM無し)エンコード形式で作成されたものを指定します。

-s <script_name>

アップロード先のスクリプト名を指定します。既存のスクリプト名を指定すると <input_file> で指定したファイルの内容で上書きされます。新規のスクリプト名を指定すると、abs_agent が動作しているコンピュータ上にスクリプトファイルが作成されます。スクリプト名は abs_agent が参照しているスクリプトフォルダのトップディレクトリからの相対パスになります。また スクリプト名には Lua スクリプトを示すファイル拡張子(.lua) 部分を除いて指定します。スクリプト名に "/" 文字を入れると直前の文字列はディレクトリ名になります。たとえば "agent_put -s ABC/DEF/GHI -f xxxx" は、abs_agent で設定されたスクリプト・トップディレクトリ内の ABC サブディレクトリ、その中の DEF サブディレクトリ内の GHI.lua ファイルが更新または新規作成されます。サブディレクトリが存在しない場合には自動的に作成されます。

-d

スクリプトファイルを削除します。削除対象のスクリプトファイルは、-s <script_name> オプションで指定します。agent_put プログラムではスクリプトフォルダは削除しませんので、必要な場合には abs_agent が動作しているコンピュータ上で、直接ファイルシステム中のディレクトリを削除して下さい。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0　　正常終了した

1　　コマンドパラメータが間違っています

2　　実行時にエラーが発生した

- **説明**

abs_agent サーバーのスクリプトをアップロードします。既存のスクリプトを更新する場合には、スクリプト名は agent_script コマンドで得られるスクリプト一覧の中から指定します。"-r" オプションを指定してリモートコンピュータのスクリプトを更新することもできます。

スクリプトを新規に作成する場合には、ワークファイルに Lua スクリプトを記述した後 agent_put コマンドでファイルをアップロードします。このとき、スクリプト名には好きなディレクトリ名とスクリプト名を指定することができます。ディレクトリが存在しない場合には自動的に作成されます。スクリプト名（ディレクトリ名を含む）には日本語を使用することもできます。

> **OS 上に Lua スクリプトファイル(*.lua) を直接作成・更新するよりも、agent_put コマンドを使用する方が便利です**
>
> agent_put コマンドを使用しなくても、abs_agent をインストールした先に保存されているスクリプトファイルを直接参照してファイルを作成したり更新することも可能です。この時、スクリプトを保存しているトップディレクトリが abs_agent.xml サーバー設定ファイル中に記述されていますので、必ずそのディレクトリ内のファイルにアクセスするようにします。スクリプト保存先はサーバー毎に設定が異なっている可能性がありますので、agent_put コマンドを使用するほうが簡単に操作できます。スクリプトのダウンロード(agent_get)、アップロード(agent_put)、実行(agent_script)で指定するスクリプト名パラメータは共通で "-s <script_name> になっています。このため、Linux シェルのヒストリ編集機能を使うと効率よく作業を行えます。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") に新規のスクリプト TEST/試験/SAMPLE を作成する。その後そのスクリプトを実行する。

```
pi@raspi3:~/abs_agent$ cat > my_script << EOF
> script_result(g_taskid,"Message","こんにちは")
> EOF
pi@raspi3:~/abs_agent$ cat my_script
script_result(g_taskid,"Message","こんにちは")
pi@raspi3:~/abs_agent$ ./agent_put -s TEST/試験/SAMPLE -f my_script
```

```
pi@raspi3:~/abs_agent$ ./agent_script | grep 試験
TEST/試験/SAMPLE
pi@raspi3:~/abs_agent$ ./agent_script -s TEST/試験/SAMPLE

ServerName: raspi3      ResultParam(s): 1
---------------------------------------------------------------------
<Key>=<Value>
---------------------------------------------------------------------
Message=こんにちは

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の SAMPLE スクリプトの内容の内容を work_file にダウンロードします work_file の内容を編集してリターンパラメータにメッセージを出力するように変更します。その後、work_file をリモートコンピュータ (サーバー名 "raspberrypi", IP アドレス 192.168.100.14) にスクリプト名 TEST/MY_SAMPLE として保存した後、スクリプトを実行します。

```
pi@raspi3:~/abs_agent$ ./agent_get -s SAMPLE -f work_file
pi@raspi3:~/abs_agent$ cat >> work_file << EOF
> script_result(g_taskid,"Message","リモートからのメッセージ")
> EOF
pi@raspi3:~/abs_agent$ cat work_file

file_id = "SAMPLE"
--[[
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
*******************************************************************************
]]

log_msg("start..",file_id)
log_msg(string.format("モジュール開始時刻 %s ホスト名 %s リクエスト
元 %s",g_startup,g_hostname,g_sender),file_id)
log_msg(string.format("スクリプトに渡されたパラメーター覧"),file_id)
for key,val in pairs(g_params) do
    log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end
```

```
script_result(g_taskid,"Message","リモートからのメッセージ")
pi@raspi3:~/abs_agent$ ./agent_put -f work_file -s TEST/MY_SAMPLE -r 192.168.100.14
pi@raspi3:~/abs_agent$ ./agent_script -s TEST/MY_SAMPLE -r 192.168.100.14

ServerName: raspberrypi      ResultParam(s): 1
--------------------------------------------------------------------------
<Key>=<Value>
--------------------------------------------------------------------------
Message=リモートからのメッセージ

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

abs_agent	abs_agent サーバープログラム
agent_script	スクリプト実行
agent_get	スクリプトダウンロード

12.10 agent_webuser (Web API ユーザー管理)

- **機能説明**

  Web API ログイン認証時に使用する、ユーザーアカウントの管理を行います。

- **コマンド実行形式**

  agent_webuser *[*-a <add_user_name> *[*-p <password>*]] [*-d <delete_user_name>*] [*-r <hostname>*]*

- **コマンドパラメータ**

  -a <add_user_name>

  新しいログインユーザーを作成します。<add_user_name> で指定したログインユーザー名でユーザーアカウントを作成します。

  "-p <password>" パラメータを同時に指定すると、初期パスワードを設定することもできます。ユーザー作成時にパスワードを指定しなかった場合には、ログイン認証時に指定するパスワード文字列はチェックされません。この場合にはログイン認証後に、Web API コマンド /command/json/session_password でパスワードを後から設定することができます。

-p <password>

ログインユーザー作成時に指定する、初期パスワード文字列。

-d <delete_user_name>

<delete_user_name> で指定したユーザーを削除します。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent の Web API コマンドを実行するときのログイン認証で使用できる、ログインユーザーを作成したり削除します。

現在登録中のユーザーアカウント一覧は、agent_webuser プログラムを "-a" や "-d" オプションを指定しないで実行することで得られます。

abs_agent に登録可能なユーザー数に制限はありません。

このコマンドで管理しているログインユーザーは、abs_agent プログラムが動作しているコンピュータのマスターファイル "masters.xml" に登録されています。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") に新規のログインユーザーを作成する。ユーザー名は "user1" 初期パスワードは "mypassword" に設定する。

```
pi@raspi3:~/abs_agent$ ./agent_webuser -a user1 -p mypassword
pi@raspi3:~/abs_agent$ ./agent_webuser

ServerName: raspi3    Users: 1    Master: /home/pi/my_config/masters.xml
------------------------------------------------------------------------
<login_name>
------------------------------------------------------------------------
user1

pi@raspi3:~/abs_agent$
```

● リモートコンピュータ（サーバー名 "raspberrypi", IP アドレス 192.168.100.14）から、上記の例で作成したコンピュータ(サーバー名 "raspi3", IP アドレス 192.168.100.15)に対して Web API コマンドでログイン認証を行う。その後、得られたセッション情報を使用してスクリプト PARAM_ECHO を実行する。

```
pi@raspberrypi:~$ curl '192.168.100.15:8080/command/json/session_login?user=user1&pass=mypassword'
{"Result":"Success","ErrorText":"","SessionToken":"ST04450F2CAC3BBE"}pi@raspberrypi:~$
pi@raspberrypi:~$
pi@raspberrypi:~$ curl '192.168.100.15:8080/command/json/script?session=ST04450F2CAC3BBE&name=PARAM_ECHO&key1=val1'
{"Result":"Success","ErrorText":"","TaskID":"LUA04450F3089C669","ResultParams":{"key1":"val1"}}pi@raspberrypi:~$
pi@raspberrypi:~$
pi@raspberrypi:~$ curl '192.168.100.15:8080/command/json/session_logout?session=ST04450F2CAC3BBE'
{"Result":"Success","ErrorText":""}pi@raspberrypi:~$
pi@raspberrypi:~$
```

Web API の session=xxxxx URL パラメータで、ログイン認証で得られたセッショントークンを指定しています。

- **参照**

abs_agent　　abs_agent サーバープログラム

Web API /command/json/session_login　ログイン認証

Web API /command/json/session_password パスワード変更

Web API /command/json/session_logout ログアウト

Web API /command/json/script スクリプト実行

12.11　agent_session (Web API セッション管理)

- **機能説明**

Web API アクセス時に使用するセッション情報(セッショントークン)の管理を行います。

- **コマンド実行形式**

agent_session *[-k <kill_session_token>] [-n <new_session_token> [-i]] [-r <hostname>]*

- **コマンドパラメータ**

-k <kill_session_token>

<kill_session_token> で指定したセッション情報を削除します。ログイン認証 Web API や agent_session プログラム、create_session() ライブラリ関数で作成したセッションを削除します。

-n <new_session_token> [-i]

<new_session_token> で指定した文字列で、新しいセッション情報を作成します。ここで作成したセッション情報を使用して Web API を使用することができます。ここで作成したセッションにはログインユーザー情報が含まれませんので、パスワード変更の Web API は使用できません。

"-i" オプションを指定すると作成したセッションは自動削除の対象になりません。"-i" オプションを指定しない場合には最後にセッション情報を使用してから 10分以上経過すると自動的に abs_agent から削除されます。

<new_session_token> に空文字列 '' を指定すると abs_agent が自動的にセッショントークン文字列を生成します。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent に新規のセッションを作成したり、既存のセッションを削除することができます。

現在のセッション一覧は、agent_session プログラムを "-k" や "-n" オプションを指定しないで実行するこ

とで得られます。

abs_agent に登録可能なセッション数に制限はありません。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") に登録済みのログインユーザー(ユーザー名 "user1",パスワード "mypassword" でログインする。ログイン後のセッション情報を表示した後、ログインセッションを強制削除する。

```
pi@raspi3:~/abs_agent$ curl 'localhost:8080/command/json/session_login?user=user1&pass=mypassword'
{"Result":"Success","ErrorText":"","SessionToken":"ST04451121E8595C"}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$ ./agent_session

ServerName: raspi3      Session count: 1
-------------------------------------------------------------------------------------------------
<session_token>      <login_name>     <terminal>  <indef>        <start>          <last_update>
-------------------------------------------------------------------------------------------------
ST04451121E8595C           user1        127.0.0.1     No    2017/01/08 17:08:30  2017/01/08 17:08:30

pi@raspi3:~/abs_agent$ ./agent_session -k ST04451121E8595C
pi@raspi3:~/abs_agent$ ./agent_session

ServerName: raspi3      Session count: 0
-------------------------------------------------------------------------------------------------
<session_token>      <login_name>     <terminal>  <indef>        <start>          <last_update>
-------------------------------------------------------------------------------------------------

pi@raspi3:~/abs_agent$
```

● リモートコンピュータ (サーバー名 "raspberrypi", IP アドレス 192.168.100.14) に "MY_SESSION" の名前で新規にセッションを作成する。その後、そのセッション情報を使用してWeb API 経由でスクリプト PARAM_ECHO をリモートコンピュータ上で実行する。

```
pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$ ./agent_session -r 192.168.100.14 -n MY_SESSION -i
```

```
NewSessionToken=MY_SESSION
pi@raspi3:~/abs_agent$ ./agent_session -r 192.168.100.14

ServerName: raspberrypi      Session count: 1
-------------------------------------------------------------------------------------------------
<session_token>      <login_name>     <terminal>  <indef>        <start>          <last_update>
-------------------------------------------------------------------------------------------------
      MY_SESSION       -- Null --        -- Null --    Yes    2017/01/08 17:14:26  2017/01/08 17:14:26

pi@raspi3:~/abs_agent$ curl '192.168.100.14:8080/command/json/script?session=MY_SESSION&name=PARAM_ECHO&Key2=Val2'
{"Result":"Success","ErrorText":"","TaskID":"LUA0445113FB13DE3","ResultParams":{"Key2":"Val2"}}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

- **参照**

  abs_agent　　abs_agent サーバープログラム

  Web API /command/json/script スクリプト実行

12.12　agent_fastdb (時系列集計用インメモリデータベース操作)

- **機能説明**

  集計用インメモリデータベース(FASTDB) のデータ参照、集計計算、データ追加・削除、ファイルへのデータ保存やファイルからのデータ読み込みを行います。

- **コマンド実行形式**

  (データ検索、ファイルにデータ保存)

  agent_fastdb *[*-k <key> *[*-s <dump_from_datetime>*]* *[*-t <dump_until_datetime>*]* *[*-m <maxlist>*]* *[*-e*]* *]*
  [-f <result_file>*]* *[*-r <hostname>*]*

  (集計計算)

  agent_fastdb -k <key> -s <summary_start_datetime> -i <summary_interval> -c <summary_count>
  [-f <result_file>*]* *[*-r <hostname>*]*

  (データ追加)

  agent_fastdb -k <key> -v <add_new_data> *[*-s <new_data_timestamp>*]*
  [-r <hostname>*]*

(古いデータ削除)

agent_fastdb -k <key> -d [<purge_until_datetime>]
[-r <hostname>]

(ファイルからデータ読み込み)

agent_fastdb -p <data_file> [-q] [-r <hostname>]

(ロック中のキーを表示)

agent_fastdb -l [-r <hostname>]

(全てのキーをアンロック)

agent_fastdb -u [-r <hostname>]

(別コンピュータのグローバル共有メモリ領域にデータ・バックアップ)

agent_fastdb -B <backup_hostname> [-k <key>]

(別コンピュータのグローバル共有メモリ領域に保存されたバックアップからデータ・リストア)

agent_fastdb -R <backup_hostname> [-a <adj_sec>] [-k <key>]

(別コンピュータのグローバル共有メモリ領域に保存されたバックアップデータ削除)

agent_fastdb -D <backup_hostname> [-k <key>]

(別コンピュータのグローバル共有メモリ領域に保存されたバックアップデータをリスト表示)

agent_fastdb -L <backup_hostname>

(日付時刻間の秒数を計算)

agent_fastdb -b -s <datetime1> -t <datetime2>

- **コマンドパラメータ**

-k <key>

データベースに格納されたデータのキー名を指定する。データ検索時にこのオプションを指定した場合には、指定したキーを持つデータを検索してコンソールに出力します。集計計算やデータ追加、削除時にもこのパラメータで対象となるデータのキー文字列を指定します。

データベースで使用中のキー名一覧は agent_fastdb をオプション無し ("-r <hostname>"は指定可)で実行すると得られます。

-s <dump_from_datetime>

指定した日時を含むそれ以降のタイムスタンプを持つデータを検索対象とする。
"YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。

このパラメータを省略した場合には、FASTDB に登録されている最も古いデータから検索します。

-t <dump_until_datetime>

指定した日時を含まないでそれよりも古いのタイムスタンプを持つデータを検索対象とする。"YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。

このパラメータを省略した場合には、FASTDB に登録されている最も新しいデータまでを検索します。

-m <maxlist>

検索結果で取得するデータ数の最大値を指定する。このパラメータを省略した場合には 500 が指定されたものとして検索結果が返されます。

このパラメータで指定した数よりも多くのデータが検索結果で得られた場合には、パラメータで指定された数までがコンソールに出力され、同時に警告メッセージが表示されます。検索結果を絞り込むために、"-s <dump_start_datetime>" や "-t <dump_end_datetime>"オプションを使用することができます。

-e

検索結果のソート順を昇順(タイムスタンプが古いものから先に表示)にする。このパラメータを省略した場合には降順(タイムスタンプが新しいものから先に表示)になります。

-s <summary_start_datetime>

集計計算を行う時に計算対象の開始日時を指定する。文字列形式で 'YYYY/MM/DD HH:MM:SS' のフォーマットで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。

-i <summary_interval>

集計計算の間隔を秒で指定する。例えば 10 分毎に集計する場合には 600 を指定します。

-c <summary_count>

集計計算の繰り返し回数を指定する。"-s <summary_start_datetime>" で指定した日時から "-i <summary_interval>" 秒経過した時刻までの集計計算を、このパラメータで指定した回数分繰り返し実行して集計結果を出力します。

-v <add_new_data>

データベースに新規データを追加するときのデータ値を指定する。

-s <new_data_timestamp>

"-v <add_new_data>" パラメータで追加するデータのタイムスタンプ値を指定する。

"YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。

このパラメータを省略したときには、abs_agentが動作しているコンピュータのシステム時刻がタイムスタンプに使用されます。

-d [<purge_until_datetime>]

データベースからデータを削除する。**"-d" オプションに続けて日付時刻パラメータを指定しなかった場合には "-k <key>" パラメータで指定したキー値を持つ全てのデータが削除されます。**

日付時刻を指定した場合には、指定した日時を含まないでそれよりも古いのタイムスタンプを持つデータを削除対象とする。

"YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。

-p <data_file> [-q]

複数のデータをデータベースに登録します。追加するデータのキー名、タイムスタンプ値と値のペアを <data_file> で指定したファイル中に記述します。

<data_file> はテキストファイル形式で、１行につき１つのデータを指定します。データのキー名を "=" 文字の左側に指定して、右側にはデータのタイムスタンプ値とデータ値を "," カンマでつなげた形で記述します。ファイルの１行毎のフォーマットは
<Key>=<YYYY/MM/DDHH:MM:SS>,<Val> または <Key>=<YYYY/MM/DDHH:MM:SS.mmm>,<Val> にします。

"-q" オプションを "-p"と同時に指定した場合には、データファイルで指定された全てのキー値を持つデータをデータベースから削除した後、データがロードされます。

"-q" オプションを指定しない場合には、既存のデータベースにデータ追加のみを行います。

キー名毎にまとめてファイル中で指定したデータを追加します。キー名は <data_file>ファイル中で行を連続して並べる必要はありません。またタイムスタンプ値によって予め <data_file> 中のデータをソートする必要もありません。

第一カラムが '#' 文字の場合には行全体をコメント行として読み飛ばします。キーや値に日本語を使用する場合は、必ず UTF-8(BOM無し)エンコード形式で保存して下さい。（下記参照）

```
# コメント行
Key#1=TimeStamp#1,Val#1
```

```
Key#1=TimeStamp#2,Val#2
..
Key#2=TimeStamp#1,Val#1
Key#2=TimeStamp#2,Val#2
```

-B <backup_hostname> *[-k <key>]*

-R <backup_hostname> *[-a <adj_sec>] [-k <key>]*

-L <backup_hostname>

-D <backup_hostname> *[-k <key>]*

"-B" 指定時は、FASTDBに登録済みのレコードデータを別コンピュータ上のグローバル共有メモリエリアにバックアップする。-k <key> を同時に指定すると特定のキーのレコードデータのみを対象にする。-k <key> パラメータ省略時は全てのキーとそのレコードデータが対象となる。バックアップ時に<backup_hostname> に対するアクセス権が必要になるので、バックアップコンピュータ上で agent_host コマンドで自身のホスト名を登録しておく事。複数回バックアップすると最後にバックアップしたデータで上書きされる。FASTDBに登録されたレコード数が 0 のキーはバックアップ対象から外される。

"-R" を使用すると、"-B" コマンドでバックアップされたデータを元に、特定のキーまたはFASTDBの全キーとそのレコードデータをリストアする。FASTDB内のリストア対象のキーに既存レコードが存在する場合には、リストア前に全て消去されるので注意すること。リストア時には自ホストから <backup_hostname> に対するアクセス権に加えて、自ホストへのアクセス権が <backup_hostname> 側の abs_agent に必要となる。このためバックアップコンピュータ上で agent_host コマンドで自身のホスト名の登録と自ホスト側の abs_agent に<backup_hostname> からのアクセス権を両方とも設定しておく事。"-a <adj_sec>" オプションを指定すると、リストア時にタイムスタンプ値を補正できる。単位は秒で小数点以下３桁(ミリ秒)まで指定可。プラス値はバックアップデータ中のタイムスタンプ値に補正分を足して未来にずらし、マイナス値は時刻を過去にずらす。 パラメータ省略時または 0 を指定するとタイムスタンプ値の補正は行わない。２つのタイムスタンプ値から <adj_sec> パラメータで指定する秒数を計算するために agent_fastdb コマンドの "-b"オプションを使用できます。

"-L" はバックアップ済みの FASTDBキー一覧を表示する。

"-D" はバックアップ済みデータを消去する。

バックアップ関連のオプション("-B","-R","-L","-D")は、リモートコンピュータ上でコマンド実行する "-r <hostname>" と同時に指定することはできません。

-b -s <datetime1> -t <datetime2>

<datetime1> と <datetime2> で指定された２つの日付時刻間の秒数を計算する。<datetime1> が <datetime2>と比較して過去(古いタイムスタンプ)の場合にはプラスの数値で、未来(新しいタイムスタンプ)の場合にはマイナスの数値になる。単位は秒で小数点以下３桁(ミリ秒)までの数値で返る。<datetime1>、<datetime2> パラメータは、"YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。このとき、パラメータの日付と時刻はシングルコート文字で囲ってください。計算結果で得られる秒数のミリ秒部分については、内部の時刻表現(浮動小数点)に起因する計算誤差によって数ミリ秒の誤差が発生する場合があります。

-f <result_file>

このオプションを指定しない場合には、データベースの検索を行うとコンソールにデータのタイムスタンプ値とデータ値のペアが表示されます。また、集計作業を行った場合には、集計範囲の日時やその間のデータの集計値等がコンソールに表示されます。

"-f" オプションを指定すると、これらのデーター覧を <result_file> で指定したファイルに書き込みます。

データ検索時の <result_file> の出力フォーマットは、データをファイルからロードするときに指定する "-p <data_file>" と同一のフォーマットです。このため、abs_agent で現在使用中のデータベースのバックアップやリストアを行ったり、他のサーバーにデータをコピーするときにも使用できます。

データ集計時の <result_file> の出力フォーマットは、集計回数で指定された行分のカンマ区切りデータフォーマット(CSV)で出力されます。キー名や集計期間、集計値がそれぞれカラムごと出力されますので、表計算ソフトで集計結果を簡単に利用することができます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

-l

FASTDB へのデータ追加・削除・集計計算中には、該当するキーのデータを短期間自動的にロックしてデータの整合性を保っています。-l オプションを指定すると、現在使用中でロックした状態のキー名一覧を表示します。これらの使用中のロックしたキーをアンロックする場合には -u オプションを使用します。

-u

FASTDB へのデータ追加・削除・集計計算中には、該当するキーのデータを短期間自動的にロックしてデータの整合性を保っています。何らかの原因でキーがロック中のままデータ操作ができなくなった場合に、-u オプションを指定すると、現在使用中でロックした状態の全てのキーを強制的にアンロックします。

ロックが原因でデータ操作ができなくなった場合には、ログに下記の様なメッセージが表示されます。また、このコマンドを使用しなくても abs_agent を再起動することでロック状態は解除されます。

2017/12/07 07:55:48 raspi3 LuaApiFASTDB 0 fastdb_add:*EXCEPTION* could not reserve the key

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーの FASTDB データベースの参照や更新を行います。abs_agent のスクリプトやイベントハンドラから FASTDB 用のライブラリ関数で操作するのと同等の機能です。FASTDB の詳細機能についてはライブラリ関数 API の項も参照してください。

現在 FASTDB で使用中の全てのキー名一覧は agent_fastdb コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

指定したキー名を使用したデータを検索する場合には "-k<Key>" オプションを付けて実行します。このとき、検索範囲や取得レコード最大数やソート順などをオプションパラメータで指定できます。

集計計算を行う場合には集計対象となるデータのキー名を "-k <Key>" オプションと、集計開始日時と集計間隔、集計回数を各々パラメータで指定します。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

バックアップやリストアは下記のスクリプトと連動して動作します。スクリプトは Lua スクリプトが記述されたテキストファイルなので、必要に応じてユーザーがある程度カストマイズすることも可能です。

agent_fastdb プログラム中から起動するスクリプト	
スクリプト	説明
FASTDB/BACKUP_RESTORE	指定したキーのFASTDB データを別コンピュータ上のグローバル共有メモリエリアに転送してバックアップを作成する。 また、別コンピュータ上にあるバックアップデータを転送して FASTDBデータのリストアを行う

- **使用例**

● localhost(サーバー名 "raspi3") のFASTDB に登録中のキー名一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb

ServerName: raspi3      Keys: 12
------------------------------------------------------------------------
<Key>=<RecordCount>
------------------------------------------------------------------------
温度(２階)=8807
気圧(２階)=8807
湿度(２階)=8807
明るさ(２階)=8807
温度(１階)=534
赤外(１階)=534
温度(２階リモート)=537
気圧(２階リモート)=537
湿度(２階リモート)=537
赤外(２階リモート)=537
TEST1=3
TEST4=4

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の FASTDB にデータ値 3.14 を追加する。追加データのキー名は '試験キー1' で、タイムスタンプはシステム時刻とパラメータで指定した日時の２つの方法で追加する。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '試験キー1' -v 3.14
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '試験キー1' -v 3.14 -s '2001/1/1 0:0:0'
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '試験キー1'
```

```
ServerName: raspi3   Key: 試験キー1    Data: 2
------------------------------------------------------------------------
<TimeStamp>             <data>
------------------------------------------------------------------------
2017/04/19 13:10:15     3.14
2001/01/01 00:00:00     3.14


pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の FASTDB キー名 '温度(２階)'で最近登録されたデータ10件を検索する。

```
  pi@raspi3:~/abs_agent$ ./agent_fastdb -k '温度(２階)' -m 10
  *WARNING* more data exists >10, refer to "-m <maxlist>" option.


  ServerName: raspi3   Key: 温度(２階)    Data: 10
  ------------------------------------------------------------------------
  <TimeStamp>             <data>
  ------------------------------------------------------------------------
  2017/04/19 13:13:32     22.1
  2017/04/19 13:12:32     22.2
  2017/04/19 13:11:32     22.3
  2017/04/19 13:10:32     22.2
  2017/04/19 13:09:32     22.1
  2017/04/19 13:08:32     22.2
  2017/04/19 13:07:32     22.2
  2017/04/19 13:06:32     22.1
  2017/04/19 13:05:32     22
  2017/04/19 13:04:32     21.9


pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の FASTDB データで、キー名 '温度(２階)' で 20017/4/19 AM10時台に登録された全てのデータを検索する。また出力時にタイムスタンプが古いデータ順に並べる。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '温度(２階)' -s '2017/4/19 10:0:0' -t '2017/4/19 11:0:0' -e


ServerName: raspi3   Key: 温度(２階)    Data: 60
------------------------------------------------------------------------
<TimeStamp>             <data>
```

```
-------------------------------------------------------------------

2017/04/19 10:00:24      21.9
2017/04/19 10:01:24      21.9
2017/04/19 10:02:24      21.9
2017/04/19 10:03:24      21.9
2017/04/19 10:04:24      21.9
.. 途中省略
2017/04/19 10:57:27      21.8
2017/04/19 10:58:27      21.8
2017/04/19 10:59:27      21.8


pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の FASTDB データで、キー名 '温度(２階)' の 2017/4/18 1日分の 10分毎の集計値を計算する。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '温度(２階)' -s '2017/4/18 0:0:0' -i 600 -c 144


ServerName: raspi3   Key: 温度(２階)    Count: 144
-------------------------------------------------------------------------------------------------------------------
 No      DateTimeFrom          DateTimeUntil     Sample     Total        Mean         Max          Min
-------------------------------------------------------------------------------------------------------------------
   1 2017/04/18 00:00:00 - 2017/04/18 00:10:00    10         247.8       24.78        24.8         24.7
   2 2017/04/18 00:10:00 - 2017/04/18 00:20:00    10          247        24.7         24.7         24.7
   3 2017/04/18 00:20:00 - 2017/04/18 00:30:00    10         246.4       24.64        24.7         24.6
   4 2017/04/18 00:30:00 - 2017/04/18 00:40:00    10         245.7       24.57        24.6         24.4
   5 2017/04/18 00:40:00 - 2017/04/18 00:50:00    10         245.1       24.51        24.6         24.5
   6 2017/04/18 00:50:00 - 2017/04/18 01:00:00    10         244.9       24.49        24.5         24.4
   7 2017/04/18 01:00:00 - 2017/04/18 01:10:00    10          244        24.4         24.5         24.3
   8 2017/04/18 01:10:00 - 2017/04/18 01:20:00    10         243.8       24.38        24.4         24.3
   9 2017/04/18 01:20:00 - 2017/04/18 01:30:00    10         242.9       24.29        24.3         24.2
  10 2017/04/18 01:30:00 - 2017/04/18 01:40:00    10         242.9       24.29        24.3         24.2
.. 途中省略
 142 2017/04/18 23:30:00 - 2017/04/18 23:40:00    10         221.7       22.17        22.2         22.1
 143 2017/04/18 23:40:00 - 2017/04/18 23:50:00    10         220.9       22.09        22.1          22
 144 2017/04/18 23:50:00 - 2017/04/19 00:00:00     9          198         22           22           22


pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") の FASTDB データで、キー名 '試験キー1'の全データをローカルファイル list1 に出力する。その後、list1 ファイルを使用してリモートコンピュータ(サーバー名 "raspberrypi", IP アドレス 192.168.100.14)にデータを登録する。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '試験キー1' -f list1
pi@raspi3:~/abs_agent$ cat list1
#
# agent_fastdb [2017/04/19 13:28:38]
#
試験キー1=2017/04/19 13:10:15,3.14
試験キー1=2001/01/01 00:00:00,3.14
pi@raspi3:~/abs_agent$ ./agent_fastdb -r 192.168.100.14 -p list1 -q
pi@raspi3:~/abs_agent$ ./agent_fastdb -r 192.168.100.14 -k '試験キー1'

ServerName: raspberrypi   Key: 試験キー1    Data: 2
-----------------------------------------------------------------------
<TimeStamp>             <data>
-----------------------------------------------------------------------
2017/04/19 13:10:15     3.14
2001/01/01 00:00:00     3.14

pi@raspi3:~/abs_agent$
```

予めリモートコンピュータ側の abs_agent では、agent_hosts コマンドで raspi3 ホストからのリクエストを許可しておく必要があります。

● localhost(サーバー名 "raspi3") の FASTDB データで、キー名 '試験キー1' の全データを削除する。また、キー名 '温度(２階)' の 2017年よりも古いデータも削除する。

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '試験キー1' -d
pi@raspi3:~/abs_agent$ ./agent_fastdb -k '温度(２階)' -d '2017/1/1 0:0:0'
```

● localhost(サーバー名 "raspi3") の FASTDB 全レコードをリモートコンピュータ mercury(192.168.100.18) にバックアップする。その後、localhost の abs_agent 再起動後にバックアップしたデータを元に FASTDB 全レコードをリストアする。

(1) バックアップ先コンピュータmercury(192.168.100.18) 側で、バックアップ元コンピュータ raspi3(192.168.100.15) からのアクセス権を追加しておく。

```
pi@mercury:~$ cd
pi@mercury:~$ cd abs_agent/
```

```
pi@mercury:~/abs_agent$ ./agent_hosts -a raspi3
pi@mercury:~/abs_agent$
```

(2) バックアップ元コンピュータ raspi3(192.168.100.15) 側で後のリストア作業時に必要となるバックアップ先コンピュータ mercury(192.168.100.18) からのアクセス権を追加しておく。

```
pi@raspi3:~$ cd
pi@raspi3:~$ cd abs_agent/
pi@raspi3:~/abs_agent$ ./agent_hosts -a mercury
pi@raspi3:~/abs_agent$
```

(3) バックアップ元コンピュータ raspi3(192.168.100.15)で動作中の abs_agent で、全 FASTDB レコードをリモートコンピュータ mercury(192.168.100.18)にバックアップする。

```
pi@raspi3:~$ cd
pi@raspi3:~$ cd abs_agent/
pi@raspi3:~/abs_agent$ ./agent_fastdb

ServerName: raspi3      Keys: 7
------------------------------------------------------------------------
<Key>=<RecordCount>
------------------------------------------------------------------------
温度(２階)=14871
気圧(２階)=14871
湿度(２階)=14871
明るさ(２階)=14870
温度(１階)MQTT=988
赤外(１階)MQTT=988
FreeMemMB=1250

pi@raspi3:~/abs_agent$ ./agent_fastdb -B 192.168.100.18
[create-backup] 温度(２階) (14872 records)
[create-backup] 気圧(２階) (14872 records)
[create-backup] 湿度(２階) (14872 records)
[create-backup] 明るさ(２階) (14871 records)
[create-backup] 温度(１階)MQTT (988 records)
[create-backup] 赤外(１階)MQTT (988 records)
[create-backup] FreeMemMB (1250 records)

pi@raspi3:~/abs_agent$ ./agent_fastdb -L 192.168.100.18
```

```
BackupHost: mercury
------------------------------------------------------------
<Key>           <records(bytes)>        <BackupTimeStamp>
------------------------------------------------------------
温度(２階)      14872(237952)           2023/09/09 14:02:44
気圧(２階)      14872(237952)           2023/09/09 14:02:44
湿度(２階)      14872(237952)           2023/09/09 14:02:45
明るさ(２階)    14871(237936)           2023/09/09 14:02:46
温度(１階)MQTT  988(15808)              2023/09/09 14:02:46
赤外(１階)MQTT  988(15808)              2023/09/09 14:02:46
FreeMemMB       1250(20000)             2023/09/09 14:02:47

pi@raspi3:~/abs_agent$
```

(4) バックアップ元コンピュータ abs_agentシャットダウン

```
pi@raspi3:~/abs_agent$ ./agent_task -K
pi@raspi3:~/abs_agent$ ./agent_shutdown
pi@raspi3:~/abs_agent$
```

(5) バックアップ元コンピュータ abs_agent 再起動 (OS 再起動でも構わない)

```
pi@raspi3:~$ sudo /home/pi/abs_agent/abs_agent -I 192.168.100.45 -c /home/pi/my_config_raspi3/abs_agent.xml
```

(6) バックアップ元コンピュータ raspi3(192.168.100.15)でバックアップ先コンピュータ mercury(192.168.100.18) に保存されているバックアップデータを元にリストア

```
pi@raspi3:~$ cd abs_agent/
pi@raspi3:~/abs_agent$ ./agent_fastdb

ServerName: raspi3      Keys: 0
------------------------------------------------------------------------
<Key>=<RecordCount>
------------------------------------------------------------------------

pi@raspi3:~/abs_agent$ ./agent_fastdb -L 192.168.100.18

BackupHost: mercury
------------------------------------------------------------
<Key>           <records(bytes)>        <BackupTimeStamp>
------------------------------------------------------------
```

```
温度(２階)        14872(237952)          2023/09/09 14:02:44
気圧(２階)        14872(237952)          2023/09/09 14:02:44
湿度(２階)        14872(237952)          2023/09/09 14:02:45
明るさ(２階)      14871(237936)          2023/09/09 14:02:46
温度(１階)MQTT    988(15808)             2023/09/09 14:02:46
赤外(１階)MQTT    988(15808)             2023/09/09 14:02:46
FreeMemMB         1250(20000)            2023/09/09 14:02:47

pi@raspi3:~/abs_agent$ ./agent_fastdb -R 192.168.100.18
[restore-backup] 温度(２階)
[restore-backup] 気圧(２階)
[restore-backup] 湿度(２階)
[restore-backup] 明るさ(２階)
[restore-backup] 温度(１階)MQTT
[restore-backup] 赤外(１階)MQTT
[restore-backup] FreeMemMB

pi@raspi3:~/abs_agent$
```

(7) バックアップ元コンピュータ raspi3(192.168.100.15)のコンソールから、不要になったバックアップ先コンピュータ mercury(192.168.100.18) 上のバックアップデータを削除

```
pi@raspi3:~/abs_agent$ ./agent_fastdb -D 192.168.100.18
[delete-backup] 温度(２階)
[delete-backup] 気圧(２階)
[delete-backup] 湿度(２階)
[delete-backup] 明るさ(２階)
[delete-backup] 温度(１階)MQTT
[delete-backup] 赤外(１階)MQTT
[delete-backup] FreeMemMB

pi@raspi3:~/abs_agent$
```

- **参照**

abs_agent	abs_agent サーバープログラム
agent_hosts	リモートからのアクセス許可

12.13 agent_shmem (グローバル共有メモリエリア操作)

- **機能説明**

グローバル共有メモリエリアの操作を行います。指定したメモリエリアの内容をコンソールに表示したり、サイズの変更、メモリエリアの新規作成や削除などを実行できます。

メモリエリアの内容をファイルに保存することができます。また反対に、ファイルの内容をロードしてメモリエリアに格納することもできます。

リモート側に設置した abs_agent に対してメモリエリアの操作をすることもできます。

- **コマンド実行形式**

(チャンネルリスト表示)

agent_shmem *[*-f <result_file>*]* *[*-r <hostname>*]*

(メモリエリアの内容を表示)

agent_shmem -c <channel> *[*-i <offset>*]* *[*-s <dump_size>*]* *[*-N*]* *[*-f <result_file>*]* *[*-r <hostname>*]*

(バイナリデータをメモリエリアに格納、メモリエリア作成、メモリエリアサイズ拡張)

agent_shmem -t <hexstr_or_num> -c <channel> *[*-i <offset>*]* *[*-N*]* *[*-r <hostname>*]*

(メモリエリア全体を指定したバイトデータで埋める)

agent_shmem -z <fill_byte_data> -c <channel> *[*-r <hostname>*]*

(メモリエリア内のデータを16進数文字列データで出力)

agent_shmem -h -c <channel> *[*-i <offset>*]* *[*-s <copy_size>*]* *[*-r <hostname>*]*

(メモリエリア作成、メモリエリアサイズ拡張・縮小)

agent_shmem -e -c <channel> *[*-s <new_size>*]* *[*-r <hostname>*]*

(メモリエリア内のデータを移動(複製)する)

agent_shmem -m -c <channel> -i <offset_from> -x <offset_to> -s <move_size> *[*-r <hostname>*]*

(メモリエリアチャンネル名変更)

agent_shmem -c <channel> -n <new_channel> *[*-r <hostname>*]*

(メモリエリア削除)

agent_shmem -d -c <channel> *[*-r <hostname>*]*

(ローカルファイルの内容をメモリエリアに格納、メモリエリア作成、メモリエリアサイズ拡張)

agent_shmem -l <local_file> -c <channel> *[*-R*]* *[*-i <offset>*]* *[*-r <hostname>*]*

（メモリエリア内のデータをローカルファイルに出力）

agent_shmem -b <local_file> -c <channel> [-i <offset>] [-s <copy_size>] [-r <hostname>]

（メモリエリアの内容を別コンピュータに転送）

agent_shmem -c <channel> -X <target_host> [-r <hostname>]

- **コマンドパラメータ**

-c <channel>

メモリエリアのチェンネル名を指定する。
"-t <store_data_hexstr>" や "-e" オプション指定時に、存在しないチャンネル名を指定すると新規にメモリエリアが作成されます。

チャンネル名一覧は agent_shmem をオプション無し ("-r <hostname>"と
"-f <result_file>" は指定可)で実行すると得られます。

-i <offset>

メモリエリア内のデータを指定するときに、メモリエリア先頭からのオフセット・バイト数を指定する。メモリエリアの内容を参照したり、メモリエリアにデータを格納するときに指定したオフセット位置以降のデータが操作対象になります。-i パラメータ省略時は 0 を指定したのと同じです。

-N オプション指定時は、IEEE-754 倍精度浮動少数値(8バイト単位)のメモリエリア先頭からの個数を指定する。

-N

メモリ内容を表示する時にこのオプションを指定すると、メモリエリアの内容を 8バイト単位の IEEE-754 倍精度浮動少数値フォーマットで表示する。このオプション指定時は -i <offset> パラメータはメモリエリア先頭からの浮動少数値の個数になり、-s <dump_size> パラメータは表示する浮動小数値の個数になる。

メモリ内容を更新する時(-t <hexstr_or_num>指定時)にこのオプションを指定すると、
hexstr_or_num パラメータに指定した数値を8バイト単位のIEEE-754 倍精度浮動少数値フォーマットでメモリに格納する。"-i <offset>" パラメータで指定した浮動少数値の個数以降にデータが格納される。

-s <dump_size>

メモリエリアの内容を表示するときのバイト数を指定する。このパラメータを省略した場合や、0 を 指定するとメモリエリアの終端までのバイト数を指定したのと同じです。

-s <copy_size>

メモリエリアの内容を16進数表示するときのメモリバイト数を指定する。このパラメータを省略した場合や、0 を 指定するとメモリエリアの終端までのバイト数を指定したのと同じです。

-s <new_size>

メモリエリアをリサイズするときに、新しいメモリサイズを指定する。既存のメモリサイズよりも大きな値を指定すると既存部分のデータ内容は不変ですが、追加部分のデータには不定値が格納されます。既存のメモリサイズよりも小さな値を指定すると、メモリエリア先頭から新しいサイズまでのデータが保存されます。

-t <hexstr_or_num>

16進数文字列で指定したバイナリデータをメモリエリアに格納します。"-c <channel>" で指定したメモリエリアが存在しない場合には新規にメモリエリアが作成されます。既存のメモリエリアを指定した場合には、"-i <offset>" パラメータで指定したオフセットバイト値以降にデータが格納されます。<offset> に -1 を指定した場合には現在のメモリエリアのサイズを指定したのと同じです。(既存のメモリエリアの終端にデータが追加されます)

-N オプションを同時に指定した場合には hexstr_or_num に数値を指定するとこで、メモリエリアにIEEE-754 倍精度浮動少数値フォーマットで格納します。-N オプションの説明も参照してください。

-h

メモリエリアの内容を16進数文字列で表示します。

-e

現在のメモリエリアのサイズを変更します。"-c <channel>" で指定するチャンネル名が存在しない場合には、新しくメモリエリアが作成されます。

-m

メモリエリア内のデータを移動(複製)します。-i <offset_from> で指定した位置から
-s <move_size> で指定したバイト数のデータを -x <offset_to> 位置にコピーします。コマンド実行後も複製元データは変化しません。複製元と複製先のデータ領域が重ならないようにしてください。

-n <new_channel>

メモリエリアのチェンネル名を変更するときの、新しいチャンネル名を指定する。
"-c <channel>" オプションで、変更対象となる既存のチャンネルを指定します。

-d

メモリエリアを削除します。このコマンドを使用して明示的にメモリエリアを削除しない場合

には、abs_agent 終了時に自動的に全てのメモリエリアが削除されます。

-z <fill_byte_data>

メモリエリア全体に <fill_byte_data> で指定した数値を書き込む。
1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。

-l <local_file>

ローカルファイルの内容をメモリエリアに格納します。"-c <channel>" で指定したメモリエリアが存在しない場合には新規にメモリエリアが作成されます。既存のメモリエリアを指定した場合には、"-i <offset>" パラメータで指定したオフセットバイト値以降にデータが格納されます。<offset> に -1 を指定した場合には現在のメモリエリアのサイズを指定したのと同じです。(既存のメモリエリアの終端にデータが追加されます)

-R

-l <local_file> と同時に指定するとメモリエリア格納前に既存のメモリエリアを削除します。

-b <local_file>

メモリエリアの内容をローカルファイルに保存します。ファイルに保存するときの開始位置を指定したい場合には "-i <offset>"パラメータを使用します。また、保存するバイト数を指定する場合には "-s <copy_size>" パラメータを指定します。これらのパラメータを省略した場合にはメモリエリア全体がファイルに保存されます。

-X <target_host>

メモリエリアの内容を <target_host> で指定したabs_agent に転送する。転送先 abs_agent のハードウエアタイプ(X86,RASPI)が転送元と異なっていても構わない。-c <channel> パラメータで転送対象となるメモリエリアのチェンネル名を指定する。この時、転送元の abs_agent のメモリエリアは変化しない。転送先に同一名のチャンネルが存在した場合には、転送したデータで置き換えられる。-r <hostname> を同時に指定すると -r <hostname> から -X <target_host> への転送が行われる。この場合には agent_shmem コマンドを実行したコンピュータは <hostname> からのリモートアクセス許可を必要とするのに加えて、<target_host> 上の abs_agent のリモートアクセス許可ホスト名リスト中に <hostname> が含まれている必要がある。

-f <result_file>

このオプションを指定しない場合には、メモリエリアの内容や、チェンネルリスト一覧がコンソールに表示されます。

"-f" オプションを指定すると、これらのデータを <result_file> で指定したファイルに書き込みます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーの グローバル共有メモリエリアの参照や更新を行います。abs_agent のスクリプトやイベントハンドラから グローバル共有メモリエリア用のライブラリ関数で操作するのと同等の機能です。グローバル共有メモリエリアの詳細機能についてはライブラリ関数 API の項も参照してください。

グローバル共有メモリエリアで使用中の全てのチェンネル名一覧は agent_shmem コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

グローバル共有メモリエリアの内容をバイナリファイルの形で保存したりロードしたい場合には、ライブラリ関数 shmem_save_file(), shmem_load_file() を使用できます。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") のグローバル共有メモリエリアに登録中のチャンネル名とメモリエリアサイズ一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem

ServerName: raspi3      Channels: 4
------------------------------------------------------------------
<channel>=<size>
```

```
------------------------------------------------------------------------
試験メモリエリア=100(0x000064)
試験メモリエリア2=256(0x000100)
SerialBuffer1=512(0x000200)
FontSmall=976(0x0003D0)

pi@raspi3:~/abs_agent$
```

● チャンネル名 "TestBuff" で 0x100(64バイト) サイズのメモリエリアを作成する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -e -c TestBuff -s 0x100
pi@raspi3:~/abs_agent$
```

● "TestBuff" メモリエリア先頭から 0xF0 バイトと 0xE0 バイトの位置に、0x41, 0x42, 0x43 の３バイトをそれぞれ格納する。その後、0xD0 バイト以降のメモリエリアの内容ををコンソールに表示する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -c TestBuff -t "414243" -i 0xf0
pi@raspi3:~/abs_agent$ ./agent_shmem -c TestBuff -t "414243" -i 0xe0
pi@raspi3:~/abs_agent$ ./agent_shmem -c TestBuff -i 0xd0

ServerName: raspi3      Data: 16
------------------------------------------------------------------------
SharedMemory: TestBuff

mem_size = 256(0x000100) bytes  offset = 0x0000D0

data[0000D0]-[0000D7] FF FF FF FF FF FF FF FF

data[0000D8]-[0000DF] FF FF FF FF FF FF FF FF

data[0000E0]-[0000E7] 41 42 43 FF FF FF FF FF
                       A  B  C
data[0000E8]-[0000EF] FF FF FF FF FF FF FF FF

data[0000F0]-[0000F7] 41 42 43 FF FF FF FF FF
                       A  B  C
data[0000F8]-[0000FF] FF FF FF FF FF FF FF FF
```

```
pi@raspi3:~/abs_agent$
```

● 0x41, 0x42, 0x43 の３バイトを格納したメモリエリアを新規に作成する。チャンネル名は "ASCII" で作成する。作成後、メモリ内容をコンソールに表示する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -c ASCII -t "414243"
pi@raspi3:~/abs_agent$ ./agent_shmem -c ASCII

ServerName: raspi3      Data: 6
------------------------------------------------------------------------
SharedMemory: ASCII

mem_size = 3(0x000003) bytes  offset = 0x000000

data[000000]-[000002] 41 42 43
                      A  B  C

pi@raspi3:~/abs_agent$
```

● "ASCII" メモリエリアのサイズを 10 バイトに拡張した後、メモリエリアの内容をコンソールに表示。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -e -c ASCII -s 10
pi@raspi3:~/abs_agent$ ./agent_shmem -c ASCII

ServerName: raspi3      Data: 8
------------------------------------------------------------------------
SharedMemory: ASCII

mem_size = 10(0x00000A) bytes  offset = 0x000000

data[000000]-[000007] 41 42 43 FF FF FF FF FF
                      A  B  C
data[000008]-[000009] FF FF

pi@raspi3:~/abs_agent$
```

● "ASCII" メモリエリア を削除する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -c ASCII -d
pi@raspi3:~/abs_agent$
```

● 1MBytes(1024*1024Bytes) のサイズでランダムデータが格納されたバイナリファイルを作成する。その後、そのバイナリファイルの内容をチャンネル名 '乱数データ' でメモリエリアに格納する。ファイル名は "datafile"

```
pi@raspi3:~/abs_agent$ dd if=/dev/urandom bs=1M count=1 of=datafile
1+0 レコード入力
1+0 レコード出力
1048576 バイト (1.0 MB) コピーされました、 0.0567914 秒、 18.5 MB/秒
pi@raspi3:~/abs_agent$ ls -l datafile
-rw-r--r-- 1 pi pi 1048576  8月 20 15:35 datafile
pi@raspi3:~/abs_agent$ ./agent_shmem -c '乱数データ' -l datafile
LocalFile: datafile -> Channel: 乱数データ
pi@raspi3:~/abs_agent$ ./agent_shmem

ServerName: raspi3      Channels: 6
------------------------------------------------------------------------
<channel>=<size>
------------------------------------------------------------------------
試験メモリエリア=100(0x000064)
試験メモリエリア2=256(0x000100)
SerialBuffer1=512(0x000200)
FontSmall=976(0x0003D0)
TestBuff=256(0x000100)
乱数データ=1048576(0x100000)

pi@raspi3:~/abs_agent$
```

● 先の例で操作した "TestBuff" メモリエリアの内容をファイルに格納する。ファイル名 "bckfile"

```
pi@raspi3:~/abs_agent$ ./agent_shmem -c TestBuff -b bckfile
Channel: TestBuff -> LocalFile: bckfile
pi@raspi3:~/abs_agent$ ls -l bckfile
```

```
-rw-r--r-- 1 pi pi 256  8月 20 15:40 bckfile
pi@raspi3:~/abs_agent$
```

● ローカルファイル "bckfile" の内容をリモート側の abs_agent（サーバー名 "raspberrypi", IPアドレス 192.168.100.14）に登録する。登録するときのチャンネル名は "RemoteTestBuff" を指定する。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -l bckfile -c RemoteTestBuff -r 192.168.100.14
LocalFile: bckfile -> Channel: RemoteTestBuff
pi@raspi3:~/abs_agent$
```

● リモート側の abs_agent（サーバー名 "raspberrypi", IPアドレス 192.168.100.14）のメモリエリア "RemoteTestBuff" の 0xE0 バイト目から 24 バイト分のデータをローカルファイルに保存する。ファイル名は "copyfile"。その後 "copyfile" ファイルの内容をコンソールに表示。

```
pi@raspi3:~/abs_agent$ ./agent_shmem -b copyfile -c RemoteTestBuff -i 0xe0 -s 24 -r 192.168.100.14
Channel: RemoteTestBuff -> LocalFile: copyfile
pi@raspi3:~/abs_agent$ hexdump -v -C copyfile
00000000  41 42 43 ff ff ff ff ff  ff ff ff ff ff ff ff ff  |ABC.............|
00000010  41 42 43 ff ff ff ff ff                           |ABC.....|
00000018
pi@raspi3:~/abs_agent$
```

● チャンネル名 "TestNum" に数値データを格納した後、数値フォーマットと16進数フォーマットで表示。

```
pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum -i 0 -t 0 -N
pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum -i 1 -t 100 -N
pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum -i 2 -t 3.14159 -N
pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum -i 3 -t -1.234567e-12 -N
pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum -N

ServerName: mercury
------------------------------------------------------------------------
SharedMemory(Double Precision Numeric): TestNum

total_size = 4

num[0] 0
```

```
num[1] 100
num[2] 3.14159
num[3] -1.234567E-12

pi@mercury:~/abs_agent$ ./agent_shmem -c TestNum

ServerName: mercury
------------------------------------------------------------------------
SharedMemory: TestNum

mem_size = 32(0x000020) bytes  offset = 0x000000

data[000000]-[000007] 00 00 00 00 00 00 00 00

data[000008]-[00000F] 00 00 00 00 00 00 59 40
                                        Y  @
data[000010]-[000017] 6E 86 1B F0 F9 21 09 40
                      n              !     @
data[000018]-[00001F] C8 82 DA D7 FE B7 75 BD
                                        u
```

- **参照**

abs_agent	abs_agent サーバープログラム
agent_hosts	リモートからのアクセス許可

12.14 agent_mgcp (MGCPデバイス管理)

- **機能説明**

  MQTT ブローカー経由で任意の MGCP デバイスにリクエストメッセージを送信した後、リプライメッセージを受信する。イベントメッセージ送信やMGCPデバイス一覧表示などの管理機能も提供する。

- **コマンド実行形式**

  (MGCP デバイス一覧を表示)

  agent_mgcp *[-c] [-f <result_file>] [-r <hostname>]*

  (MQTT ブローカ・エンドポイント一覧を表示)

  agent_mgcp -p *[-f <result_file>] [-r <hostname>]*

(MGCPリクエストメッセージを送信)

agent_mgcp -d <mgcp_node> -m <request_json_str> *[-n] [-i <timeout>]*
[-f <result_file>] [-r <hostname>]

(MGCPイベントメッセージを送信)

agent_mgcp -d <mgcp_node> -v <event_json_str> -t <type> *[-a <application>] [-r <hostname>]*

(MGCP "heartbeat" イベントメッセージを送信)

agent_mgcp -e *[-r <hostname>]*

(MGCPデバイスの現在のコンフィギュレーションをダウンロードしてファイルに保存)

agent_mgcp -d <mgcp_node> -l <config_file> *[-r <hostname>]*

(ローカルにあるコンフィギュレーションファイルをアップロードしてMGCPデバイスに設定)

agent_mgcp -d <mgcp_node> -u <config_file> *[-r <hostname>]*

(MGCPダイレクトコマンドを送信)

agent_mgcp -d <mgcp_node> -x <dxcmd_hexstr> *[-n] [-i <timeout>]*
[-f <result_file>] [-r <hostname>]

- **コマンドパラメータ**

-p

abs_agent の設定ファイルで指定した MQTT ブローカへの接続用エンドポイント一覧を表示します。 agent_mgcp コマンドでは -d<mgcp_node> で指定したノード名を元に、MQTTエンドポイントが自動的に選択されます。この処理は API ライブラリ関数 mgcp_find_endpoint() を使用しています。

ノード名変更やデバイスの入れ替えによってエンドポイントとノード名の関係が変更された場合には、-c オプションを使用して abs_agent 内部に保存・更新している変換テーブルをクリアすることができます。

-c

MGCP デバイス一覧中に表示される IP アドレスと MAC アドレス情報をクリアする。また、MGCPデバイスノード名とそのデバイスが収容されている MQTTブローカへの接続エンドポイントとの対応テーブルも同時にクリアする。

これらの情報は MGCP デバイスから定期的に送信される heartbeat イベントメッセージ中に含まれるデータを元に abs_agent 側で保持しています。これらは常に最新の情報に更新されてい

ますので通常はクリアする必要はありません。メンテナンス時に既存の情報を削除して最新の情報のみを参照したい場合に使用します。一旦クリアすると、MGCP デバイスから最新の heartbeat イベントを受信するまでの間は、そのデバイスへの送信はできません。

-d <mgcp_node>

リクエストメッセージやイベント送信先の MGCP デバイスのノード名を指定します。ここで指定可能なデバイスノード名とMACアドレス一覧は agent_mgcp コマンドをオプション指定無しで実行することで得られます。

"_ALL_" のノード名は全てのデバイスに対してリクエストメッセージやイベントを送信するときに常に使用できます。ただし "_ALL_" で送信されたリクエストメッセージに応答するかどうかは、各 MGCP デバイス側で設定されています。また "_ALL_" を使用した場合にはデバイスからのリプライメッセージは受信できません (-n オプションを指定したのと同等)。

16進数表記のMAC アドレスをノード名の代わりに指定することもできます。同一ノード名を持ったデバイスが複数存在する場合などに使用できます。

-m <request_json_str>

送信するリクエストメッセージを JSON 文字列で指定します。 JSON 文字列はリクエスト先の MGCP デバイスでサポートされているコマンドとオプションを指定します。 JSON 文字列中に '"' ダブルコートや "," カンマが含まれるので、JSON 文字列全体をシングルコートで囲んでパラメータを 指定してください。

-x <dxcmd_hexstr>

送信するダイレクトコマンドのコマンドとパラメータ部分のバイナリデータを 16進数文字列で指定します。MGCP デバイスから返された実行結果のステータスとデータ部分のバイナリデータを 16進数文字列ダンプします。
MGCP デバイス側でダイレクトコマンドを有効にしている場合のみ実行できます。

-n

このオプションを指定すると、MGCP デバイス側のリクエストメッセージ処理結果ステータスやリプライメッセージを受信しません。

-i <timeout>

リプライメッセージを受信するまでのタイムアウト時間(ms)を指定する。-i オプションを指定しない場合には 2000 (2秒)をデフォルト値として使用します。タイムアウト値以内にリプライメッセージを受信できなかった場合にはエラーとなります。

-v <event_json_str>

送信するイベントメッセージを JSON 文字列で指定します。JSON 文字列中に '"' ダブルコートや "," カンマが含まれるので、JSON 文字列全体をシングルコートで囲んでパラメータを指定してください。

-e

"-v" オプションと同様にイベントメッセージを送信しますが、送信先は常に全てのノードが対象になります。abs_agent の MQTT サービスに登録されている全てのエンドポイントに対して MGCP heartbeat イベントを送信します。このとき、送信対象となるエンドポイントに MGCP デバイスが接続中であるかに関わらず無条件で送信します。このため イベント送信元ノードを MGCP デバイスノードとして外部に通知する目的で使用します。イベント送信時のデータは常に JSON 文字列で '{}' になります。

-t <type>

イベントメッセージ送信時のイベントタイプ文字列を指定します。

-a <application>

イベントメッセージ送信時のアプリケーション文字列を指定します。このオプションを省略した場合には "abs_agent" がデフォルトのアプリケーション名として指定されます。

-l <config_file>

MGCP デバイスの現在のコンフィギュレーション情報をダウンロードしてローカルファイルに保存する。
対象となる MGCP デバイスが "Demo" ライセンスの場合にはこの機能は使用できません。

-u <config_file>

コンフィギュレーション情報を格納したファイルをアップロードして MGCP デバイスに設定する。アップロード後に、最新のコンフィギュレーションで MGCP デバイスを動作させるためには、MGCP デバイスをリセットする コマンド("reset")を実行してください。
対象となる MGCP デバイスが "Demo" ライセンスの場合にはこの機能は使用できません。

-f <result_file>

コマンド実行結果や MGCP デバイスから返されたリプライメッセージは、通常コンソールに表示されます。"-f" オプションを指定すると、これらの結果を <result_file> で指定したファイルに書き込みます。また、MQTT ブローカ・エンドポイント一覧や MGCP デバイス一覧も、同様に <result_file> に出力されます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモート

に設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

  0　　正常終了した

  1　　コマンドパラメータが間違っています

  2　　実行時にエラーが発生した

- **説明**

MGCP プロトコルを使用して、MQTT ブローカ経由でリクエストメッセージやイベントを送信します。

MGCP デバイス一覧は agent_mgcp コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

agent_mgcp コマンドは下記のスクリプトやイベントハンドラと連動して動作します。

スクリプト・イベントハンドラ	説明
MGCP/MGCP_EXEC	MGCP リクエストメッセージ送信とリプライメッセージ受信
MGCP/DXCMD_EXEC	MGCP ダイレクトコマンド送信とリプライ受信
MGCP/MGCP_EVENT	MGCP イベントメッセージ送信
MGCP/MGCO_NODE_CONFIG	ノードコンフィギュレーションのダウンロードとアップロード
MQTT_PUBLISH	(1) リプライメッセージ受信待ち合わせ用のイベント処理 (2) MGCP デバイスから配信される heartbeatイベントを解析して、ノード名リストと MAC アドレスリスト、IP アドレス情報を保管する。 (3) abs_agent を MGCP デバイスとして使用する場合に、abs_agent側で提供する MGCP コマンド処理

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

abs_agent が動作しているコンピュータを MGCP デバイスとして使用することもできます。この場合には MQTT_PUBLISH イベントハンドラ中の MGCP プロトコルを処理する部分に任意のリクエストコマンド処理をユーザーが組み込めます。デフォルトでは "version","hostname","echo" コマンドをサンプル実装しています。それぞれ、abs_agent のバージョン番号取得、abs_agent が動作しているコンピュータのホスト名取得、リクエストメッセージをそのまま返す機能になっています。詳しくは MQTT_PUBLISH.lua ファイルを参照してください。

また、abs_agent 側から heartbeat イベントを最低１回(できれば定期的に) "_ALL_" ノードに対して送信して、利用側のノード名リストに abs_agent のノード名を登録させてください。イベントを送信するときには mgcp_event() ライブラリ関数を使用します(イベントデータ内容は任意で空 "{}" でも構いません)。サンプルとして提供している MGCP/SEND_HEARTBEAT_EVENT スクリプトを実行すると、heartbeat イベント送信を行います。詳しくはファイルの内容を参照してください。

- **MGCP プロトコル**

  agent_mgcp コマンドで使用する MGCP プロトコルの詳細については、インストールキット中に含まれるスクリプトライブラリファイル (preload/012_MQTT_MGCP/MGCP_LIB.lua) 中のコメント欄を参照してください。

- **使用例**

  ● localhost(サーバー名 "raspi3") で動作中の abs_agent が接続している MQTT ブローカと同じ MQTT ブローカに接続している MGCP デバイス一覧を表示。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp

ServerName: raspi3     Node(s): 2
-------------------------------------------------------------------
<node_name>              <ip_address>            <MAC_address>
-------------------------------------------------------------------
M5_NO1                   192.168.100.50          B4E62D8B78C5
ESP32_NO1                192.168.100.51          240AC4077F74

pi@raspi3:~/abs_agent$
```

  ● localhost(サーバー名 "raspi3") の abs_agent が接続中の MQTT ブローカー・エンドポイント一覧を表示。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -p

ServerName: raspi3     EndPoint(s): 5     Default:
-------------------------------------------------------------------------
<client_id>              <host>             <port>       <title>
-------------------------------------------------------------------------
abs9k:1111-raspi3        192.168.100.15     1883      センサーデータ登録
abs9k:444-raspi3         192.168.100.15     1883      エラー発生エンドポイント
abs9k:93501-raspi3       192.168.100.15     1883      センサーデータ送受信
abs9k:888-raspi3         192.168.100.15     1883      自宅監視用ブローカー
```

```
alarm_raspi3                192.168.100.15          1883        アラーム装置MQTT接続


pi@raspi3:~/abs_agent$
```

● MGCP デバイス・ノード名 "ESP32_NO1" の GPIO#27 を入出力モードに設定した後、ポート値を High に設定する。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_mode","gpio":27,"mode":"in_out"}'
{"result":"success","error_text":""}
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_set","gpio":27,"level":1}'
{"result":"success","error_text":""}
pi@raspi3:~/abs_agent$ ./agent_mgcp -d ESP32_NO1 -m '{"command":"gpio_get","gpio":27}'
{"result":"success","error_text":"","level":1}
pi@raspi3:~/abs_agent$
```

● 全MGCP デバイスにイベントタイプ "test_message" でイベント内容が {"message":"hello,world"} の JSON 文字列を送信する。

```
pi@raspi3:~/abs_agent$ ./agent_mgcp -d _ALL_ -t test_message -v '{"message":"hello,world"}'
pi@raspi3:~/abs_agent$
```

● 全MGCP デバイスにリセットコマンドを送信して全てのデバイスを強制的に再起動させる。

```
pi@mars:~/abs_agent$ ./agent_mgcp -d _ALL_ -n -m '{"command":"reset"}'
pi@mars:~/abs_agent$
```

12.15 agent_websocket (WebSocketデータ送信、クライアントリスト)

- **機能説明**

  abs_agent の WebSocketサーバーからクライアントにデータを送信したり、接続中のクライアントリストを出力します。

- **コマンド実行形式**

  (接続中の WebSocketクライアント一覧を表示)

  agent_websocket *[-r <hostname>]*

  (テキストフレームをクライアントに送信)

  agent_websocket *[-c <channel_or_wsid>]* -t <text> *[-r <hostname>]*

（バイナリフレームをクライアントに送信）

agent_websocket *[*-c <channel_or_wsid>*]* -b <hexstr> *[*-r <hostname>*]*

- **コマンドパラメータ**

-c <channel_or_wsid>

送信先WebSocket クライアントのチャンネル名もしくは WebSocketID 文字列。チャンネル名は WebSocket クライアントが接続するときに指定した URL パス名 "/<channel>/<SessionToken>" 中の <channel> に指定した文字列。複数のクライアント接続で同じチャンネル名を使用している場合には、チャンネル名が一致する全てのクライアントにフレームを送信する。

-c <channel_or_wsid> パラメータを省略するか空文字列 '' を指定した場合には、現在接続中の全ての WebSocket クライアントに対してフレームを送信する。

-t <text>

WebScoekt クライアントに送信する文字列。

JSON 形式の文字列を指定する場合には、ダブルコートが文字列中に含まれるため <text> 文字列全体をシングルコートで囲んで下さい。

-b <hexstr>

WebSocket クライアントに送信するバイナリデータを16進数文字列形式にしたもの。

例えば 0x01,0x10,0xaa,0xff の 4バイトを送信する場合には -b 0110aaff を指定する。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent 上の WebSocketサーバーで現在実行中の WebSocket クライアント一覧を取得できます。

agent_websocket コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに websocket_id と channel 一覧が表示されます。

現在接続中の WebSocket クライアントに任意のテキストフレームやバイナリフレームを送信することができます。チャンネル名を指定した場合には、WebSocket クライアント接続時に同じチャンネルを指定した全ての

クライアントに送信されます。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "raspi3") の WebSocket サーバーに接続中のクライアント一覧を表示する。

```
pi@raspi3:~/abs_agent$ ./agent_websocket

ServerName: raspi3      Client count: 4
------------------------------------------------------------------------
<websocket-id>            <start>                 <channel>
------------------------------------------------------------------------
WS04B3A98D6E92DB     2018/12/29 09:35:22          app2
WS04B3A9929A2007     2018/12/29 09:36:41          my_app
WS04B3A9929A4FBE     2018/12/29 09:36:41          app2
WS04B3A995B48018     2018/12/29 09:37:27          app2

pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") にチャンネル名 my_app で接続中のクライアントに文字列を送信する。

```
pi@raspi3:~/abs_agent$ ./agent_websocket -c my_app -t 'これは /my_app/xxxx 向けのメッセージです'
pi@raspi3:~/abs_agent$
```

● localhost(サーバー名 "raspi3") に接続中の全クライアントにバイナリデータ 0x01,0x02,0x03 を送信する。

```
pi@raspi3:~/abs_agent$ ./agent_websocket -b 010203
pi@raspi3:~/abs_agent$
```

- **参照**

abs_agent　　abs_agent サーバープログラム

agent_hosts　　リモートからのアクセス許可

12.16　agent_xbee, agent_zb (XBee 802.15.4, XBee-ZB Zigbeeデバイス管理)

- **機能説明**

シリアルポートに接続した XBee デバイスと同一 PAN 内にあるリモート XBee デバイスに AT コマンドを実

行したり、任意のデータを送信する。XBee デバイス一覧表示などの管理機能も提供する。

XBee 802.15.4 デバイスには agent_xbee コマンドを使用して、XBee-ZB Zigbee デバイスの管理には agent_zb コマンドを使用します。２つのコマンドの機能とパラメータ指定方法はほぼ共通です。

- **コマンド実行形式**

(デバイス一覧を表示)

```
agent_xbee [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
agent_zb   [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
```

(デバイス探索とマスター登録・更新)

```
agent_xbee -m [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
agent_zb   -m [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
```

(全てのデバイスをマスターから削除)

```
agent_xbee -D [-f <result_file>] [-r <hostname>]
agent_zb   -D [-f <result_file>] [-r <hostname>]
```

(abs_agent のシリアルポート一覧を表示。"XBEE","ZB"デバイスタイプのみ)

```
agent_xbee -s [-f <result_file>] [-r <hostname>]
agent_zb   -s [-f <result_file>] [-r <hostname>]
```

(ATコマンド実行)

```
agent_xbee -a <at_cmd> [-d <dest>] [-b <hexstr>] [-t <string>]
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
agent_zb   -a <at_cmd> [-d <dest>] [-b <hexstr>] [-t <string>]
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
```

(データパケット送信)

```
agent_xbee -x -d <dest> [-b <hexstr>] [-t <string>]
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
agent_zb   -x -d <dest> [-b <hexstr>] [-t <string>]
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
```

(TDCPコマンド実行)

```
agent_xbee -p <tdcp_cmd> [-n] -d <dest>
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
agent_zb   -p <tdcp_cmd> [-n] -d <dest>
                  [-c <port_or_title>] [-f <result_file>] [-r <hostname>]
```

(abs_agent で保持している XBee-ZB用 アドレス変換用キャッシュテーブルをクリア)

agent_zb -Z *[-f <result_file>] [-r <hostname>]*

- **コマンドパラメータ**

-c <port_or_title>

XBee, XBee-ZB ローカルデバイスが接続されたシリアルポート名またはタイトルを指定する。このパラメータを省略した場合には、abs_agent に設定されたシリアルポートを検索して、見つかった "XBEE" または "ZB" デバイスタイプを持つシリアルポートが指定される。

-m

abs_agent のシリアルポートに接続したローカル XBee, XBee-ZB デバイスで "Node Discover" または "Network Discovery" コマンドを実行して同一 PAN 内の XBee, XBee-ZB デバイスを検索する。見つかったリモート側とローカルの XBee, XBee-ZB デバイスは abs_agent のマスターファイルに登録される。登録済みのデバイス一覧は "agent_xbee", "agent_zb" を実行すると取得できます。

-D

abs_agent のマスターファイルに登録済みの XBee, XBee-ZB デバイスを全て削除する。このオプションを実行した場合には、再度 -m オプションで同一 PAN 内の XBee, XBee-ZB デバイス検索と登録が必要になります。

-Z

abs_agent 内部で管理している XBee-ZB デバイス用の 64ビットアドレス <-> 16ビットアドレス変換テーブルを全て削除する。このコマンド実行直後は、リモートデバイスへのアクセス時には XBee-ZB デバイス自身の持つアドレス変換・探索機能を使用することになります。リモート側からのレスポンスフレームを受信したり、イベントフレームを受信すると再び abs_agent 内部でアドレスキャッシュが自動的に作成され、リモートデバイスアクセス時のパフォーマンスを向上させます。

-s

abs_agent のシリアルデバイスに登録済みの "XBEE" または "ZB" デバイスタイプを持つシリアルポート一覧を表示する。ここで表示されたシリアルポート名またはタイトル名を -c <port_or_title> オプションに指定できる。

-a <at_cmd>

XBee, XBee-ZB デバイスで実行する AT コマンドを指定する。AT コマンドは ASCII 大文字で２文字ですが、at_cmd には小文字を指定することもできます。

コマンド実行結果は、レスポンスフレーム中のコマンドデータ部分のみを16進数文字列形式で表示します。"NI"(Node Identification) AT コマンドの場合のみ、実行結果を ASCII 文字列で表示します。このコマンドはマルチフレーム部分の表示は行いません。この場合にはライブラリ関数 xbee_at_command() または zb_at_command() を使用してください。

XBee モジュールの設定値変更のためにコマンドパラメータを指定する場合には、-b <hexstr> または -t <string> パラメータを -a パラメータと同時に指定します。

-d <dest>

AT コマンドや TDCPコマンド、データパケット送信先となる XBee, XBee-ZB デバイスを指定します。このコマンドを省略した場合には、abs_agent のシリアルポートに接続されたローカル XBee, XBee-ZB モジュールが対象となります。<dest> には以下のいずれかの文字列を指定します。

XBee, XBee-ZBデバイスの Node Identification 文字列
XBee, XBee-ZBデバイスの Serial Number(64bitアドレス)16進数表記の文字列
XBee デバイスの Source Address(16bitアドレス)16進数表記の文字列
"*" ブロードキャストアドレス

注意：XBee-ZB デバイスの16bitアドレス(Network Address) は指定できません。

-b <hexstr>

AT コマンドのコマンドパラメータや、データパケット送信データを16進数文字列で指定する。
-t パラメータとは同時に使用できません。

-t <string>

AT コマンドのコマンドパラメータや、データパケット送信データを文字列形式で指定する。
シェル上で文字列が自動解釈されるのを防ぐために、シングルコートで文字列全体を囲う必要がある場合があります。
-b パラメータとは同時に使用できません。

-p <tdcp_cmd>

指定したXBee, XBee-ZB デバイスに接続している、TDCP モニタプログラムが動作しているコントローラデバイスで実行するTDCPコマンドを指定する。

TDCP コマンドはコマンドとパラメータをカンマ区切り文字列で指定するので、シェルから -p パラメータを指定するときにはシングルコートで <tdcp_cmd> 文字列全体を囲んで下さい。
（後述の実行例を参考にしてください）

-n

TDCP コマンド実行時にリプライ受信操作を行わない。TDCP コントローラ側でのリプライパケット送信と、abs_agent 側の受信処理を省略して処理時間を短縮することができます。ただし、TDCP コントローラ側で実行時に発生したエラーは検出できません。

-f <result_file>

コマンド実行結果や MGCP デバイスから返されたリプライメッセージは、通常コンソールに表示されます。"-f" オプションを指定すると、これらの結果を <result_file> で指定したファイルに書き込みます。また、MQTT ブローカ・エンドポイント一覧や MGCP デバイス一覧も、同様に <result_file> に出力されます。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

XBee, XBee-ZB デバイスを abs_agentで管理するためのクライアントプログラムです。abs_agent のシリアルポート経由で接続した ローカル XBee, XBee-ZB デバイスと、そのローカルデバイスと同一 PAN(Personal Area Network) 内の複数の リモートXBee,XBee-ZB デバイスの管理を行うことができます。

最初にデバイス管理機能を使用する場合には、abs_agent のマスターにローカルとリモート側 XBee, XBee-ZB デバイスを登録する必要があります。-m オプションを指定することで、同一 PAN 内に存在する XBee, XBee-ZB デバイスを探索してマスターに登録することができます。

PAN 内に新規 XBee, XBee-ZB デバイスを設置した場合や、ローカルやリモート側の XBee デバイスの 16ビットアドレスを変更した場合、ローカルやリモートの XBee, XBee-ZB デバイスの "Node Identification"文字列を変更した場合には、再度 -m オプションを実行してマスターを更新してください。

-m オプションではデバイス情報の追加と更新のみ実行しますので、一度マスターに登録された XBee, XBee-ZB デバイスは削除されません。もし、デバイスリストを全てクリアしたい場合には -D オプション付きで

agent_xbee または agent_zb プログラムを実行してください。

マスターに登録済みの XBee, XBee-ZB デバイス一覧は agent_xbee または agent_zb コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

agent_xbee または agent_zb プログラムを使用して、ローカルやリモートにある XBee, XBee-ZB デバイスの設定内容(ATコマンドを使用)を確認したり、設定値を変更することもできます。

AT コマンド実行時には -d <dest> パラメータで指定した XBee, Xbee-ZB デバイスに応じて、リモートコマンド用のデータフレームまたは、ローカルデバイス用のデータフレームを自動選択して送信します。

AT コマンドフレーム中の FrameID はランダムな数値がアサインされます。またリモートコマンド中のコマンドオプションは常に 0x02(Apply Change on remote) になります。これらのデフォルト動作を変更したい場合には、agent_xbee, agent_zb プログラムからコールされるスクリプトファイル(後述)をユーザーがカストマイズできます。

AT コマンドパラメータ -a <atcmd> のみを指定して、-b <hexstr> や -t <string>パラメータを指定しない場合には、現在の設定値を取得して標準出力に表示します。設定値を変更したい場合には -a <atcmd> パラメータと同時に -b <hexstr> または -t <string>パラメータを指定します。

ATコマンドで XBee, XBee-ZB デバイスの設定を変更した場合は、必要に応じて不揮発メモリへの書き込み AT コマンド “WR” も実行してください。

-x オプションを使用すると XBee デバイスに任意のバイナリデータまたは、ASCII 文字列データを送信することができます。XBee, XBee-ZB デバイスに接続されたコントローラやサーバーのデータパケット受信機能をテストする場合等に使用します。データ送信機能はローカルデバイスに対しては使用できません。データパケットは、XBee API の Transmit Request(0x00) または XBee-ZB API の Transmit Request(0x10)を使用して送信します。

agent_xbee, agent_zb コマンドは下記のスクリプトやイベントハンドラと連動して動作します。スクリプトは Lua スクリプトが記述されたテキストファイルなので、必要に応じてユーザーがある程度カストマイズすることも可能です。

agent_xbee プログラム中から起動するスクリプト	
スクリプト・イベントハンドラ	説明
XBEE/DEVICE_LIST	XBee デバイス一覧を出力
XBEE/PORT_LIST	abs_agent にセットアップされたシリアルデバイスの中で、デバイスタイプが "XBEE" であるデバイス一覧を出力
XBEE/AT_COMMAND	AT コマンド実行

XBEE/MASTER_UPDATE	同一 PAN 内のデバイスを探索して、XBee デバイスリストを保持しているマスターファイルに反映させる。
XBEE/TDCP_COMMAND	TDCP コマンド実行
XBEE/TX_DATA	XBee デバイスにデータパケット送信
SERIAL_XBEE	(1) リモート XBee から送信されるイベントパケット処理 (2) TDCP コマンドでリプライフレームを受信したときのデータ受け渡し処理

agent_zb プログラム中から起動するスクリプト	
スクリプト・イベントハンドラ	説明
ZB/DEVICE_LIST	XBee-ZB デバイス一覧を出力
ZB/PORT_LIST	abs_agent にセットアップされたシリアルデバイスの中で、デバイスタイプが "ZB" であるデバイス一覧を出力
ZB/AT_COMMAND	AT コマンド実行
ZB/MASTER_UPDATE	同一 PAN 内のデバイスを探索して、XBee-ZB デバイスリストを保持しているマスターファイルに反映させる。
ZB/TDCP_COMMAND	TDCP コマンド実行
ZB/TX_DATA	XBee-ZB デバイスにデータパケット送信
SERIAL_ZB	(1) リモート XBee-ZB から送信されるイベントパケット処理 (2) TDCP コマンドでリプライフレームを受信したときのデータ受け渡し処理

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● XBee デバイスが接続されているシリアルポートを表示

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee -s

ServerName: raspberrypi     Port(s): 1
------------------------------------------------------------------------
<port_name>
------------------------------------------------------------------------
/dev/ttyUSB0

pi@raspberrypi:~/abs_agent$
```

-s オプションで表示されるシリアルポートが一つも無い場合には、シリアルデバイスに XBee が接続されているデバイスファイルの登録が必要です。詳しくは、サーバー設定ファイルの項を参照してください。

● 同一 PAN 内にある XBee デバイスを探索してマスターを更新する。その後 XBee デバイス一覧を表示。

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee
*ERROR* xbee_all_list() failed            ** マスタ未作成の場合にはエラーを表示します **
*ERROR* XBEE/DEVICE_LIST failed.
pi@raspberrypi:~/abs_agent$ ./agent_xbee -m
pi@raspberrypi:~/abs_agent$ ./agent_xbee

ServerName: raspberrypi     Node(s): 5        XBeeComPort: /dev/ttyUSB0
------------------------------------------------------------------------
   <serial_no>          <my_addr>            <node_id>        <is_local>
------------------------------------------------------------------------
0013A200404AC39C           0A01                Device1          false
0013A200404AC397           0B02                Device2          false
0013A20040558026           0D04                Device4          false
0013A200404AC398           0C03                Device3          false
0013A200409FD39D           0F05                Device5          true

pi@raspberrypi:~/abs_agent$
```

XBee-ZB デバイス用に実行した例は下記になります。

```
pi@raspi3:~/abs_agent$ ./agent_zb
*ERROR* zb_all_list() failed            ** マスタ未作成の場合にはエラーを表示します **
*ERROR* ZB/DEVICE_LIST failed.
pi@raspi3:~/abs_agent$ ./agent_zb -m
pi@raspi3:~/abs_agent$ ./agent_zb

ServerName: raspi3     Node(s): 4        ZBComPort: /dev/ttyUSB0
------------------------------------------------------------------------------------------------------
   <serial_no>        <net_addr>      <node_id>          <type>          <digi_id>      <is_local>
------------------------------------------------------------------------------------------------------
0013A20040A0D5BE         0000          Node2           coordinator      00030000         true
0013A20040A0D533         A35B          Node1           end device       00030000         false
0013A2004195C856         A5B5          Node4           router           00120000         false
0013A2004093D388         AB57          Node3           router           00030000         false
```

```
pi@raspi3:~/abs_agent$
```

● ローカル XBee デバイスの Node Identification と 16ビットアドレスを表示

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee -a NI
Device5
pi@raspberrypi:~/abs_agent$ ./agent_xbee -a MY
0F05
pi@raspberrypi:~/abs_agent$
```

XBee-ZB デバイス用に実行した例は下記になります。

```
pi@raspi3:~/abs_agent$ ./agent_zb -a NI
Node2
pi@raspi3:~/abs_agent$ ./agent_zb -a MY
0000
pi@raspi3:~/abs_agent$
```

● リモート XBee デバイス(64bit_address: 0013A200404AC39C) の Node Identification と 16ビットアドレスを表示

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee -a NI -d 0013A200404AC39C
Device1
pi@raspberrypi:~/abs_agent$ ./agent_xbee -a MY -d 0013A200404AC39C
0A01
pi@raspberrypi:~/abs_agent$
```

● リモート XBee デバイス(64bit_address: 0013A200404AC39C, NodeID: Device1) の 16ビットアドレスを 0x0A01 から 0x0A02 に変更する。その後マスターを更新した後、デバイス一覧を表示。

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee -a MY -d Device1 -b 0A02

pi@raspberrypi:~/abs_agent$ ./agent_xbee -a MY -d Device1
0A02
pi@raspberrypi:~/abs_agent$ ./agent_xbee -m
pi@raspberrypi:~/abs_agent$ ./agent_xbee

ServerName: raspberrypi     Node(s): 5        XBeeComPort: /dev/ttyUSB0
------------------------------------------------------------------------
   <serial_no>          <my_addr>            <node_id>        <is_local>
```

```
--------------------------------------------------------------------------------
0013A200404AC39C        0A02            Device1         false
0013A20040558026        0D04            Device4         false
0013A200404AC397        0B02            Device2         false
0013A200404AC398        0C03            Device3         false
0013A200409FD39D        0F05            Device5         true


pi@raspberrypi:~/abs_agent$
```

XBee デバイスの Node Identification, 16ビットアドレスの何れかを変更した場合には、必ず -m オプションを使用して abs_agent 側のデバイスマスターを最新の状態に更新してください。また、XBee の設定値を変更してその内容を XBee 上の不揮発メモリに保存する場合には、AT コマンド "WR" を続けて実行してください。

● リモート XBee デバイス(NodeID: Device3) に文字列 "Hello World!!" を送信する。

```
pi@raspberrypi:~/abs_agent$ ./agent_xbee -x -d Device3 -t 'Hello World!!'
pi@raspberrypi:~/abs_agent$
```

12.17 agent_copycfg (サーバー動作環境コピー)

- **機能説明**

  スクリプトファイルや abs_agent 設定ファイル、マスターファイル、Webサーバー用公開ファイルを指定したディレクトリ以下にコピーする。コピー時にユーザー環境に合わせたスクリプトファイルの更新も行う。

- **コマンド実行形式**

  agent_copycfg *[-v] [-w] [-c] [-t [-x]]* -s <source_path> -d <destination_path>

- **コマンドパラメータ**

  -s <source_path>

  コピー元となるディレクトリ名を相対パスまたは絶対パス名で指定します。abs_agent をインストールしたディレクトリやユーザーが作成した abs_agent 設定ファイルが格納されている動作環境ディレクトリを指定します。指定したディレクトリ以下に必ず scripts ディレクトリが存在しなければいけません。

  -d <destination_path>

  コピー先となるディレクトリ名を相対パスまたは絶対パス名で指定します。新規に動作環境を作成する場合には空ディレクトリを指定します。動作環境をアップデートする場合には、現在使用中の abs_agent 動作環境ディレクトリやユーザーが作成した abs_agent 設定ファイルが格納されている動作環境ディレクトリを指定します。

-t

ファイルコピーは行わずに、-s オプションで指定した <source_path>/scripts 以下に存在するファイルと -d オプションで指定した <destination_path>/scripts 以下のファイルが一致しているかを調べる。ファイル内容が異なる場合にはコンソールにメッセージを出力する。このオプション指定時は -w , -c オプションは無視される。

-x

-t オプションと同時に使用して <source_path>/scripts 以下に存在するファイルの内容が <destination_path>/scripts 以下のファイルと同じ場合に、<destination_path> 側のファイルを削除する。

-v

詳細な動作ログメッセージをコンソールに表示します。

-w

コピー元ディレクトリにある webroot ディレクトリの内容もコピーします。

-c

コピー元ディレクトリにある abs_agent.xml, masters.xml ファイルもコピーします。
コピー先に既にファイルが存在する場合にはコピーしません。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent インストールキット scripts ディレクトリ以下に格納されているオリジナルのイベントハンドラやスクリプトファイルを、新規や既存の動作環境にコピーして、abs_agent 動作環境を構築・更新します。同時に設定ファイルやマスターファイル、webroot に格納されている Webサーバー用公開ファイルや Webアプリケーションファイルもコピーすることもできます。 agent_copycfg プログラムは abs_agent サーバーが動作していない状態でも実行できます。

-s <source_path> で指定したコピー元ディレクトリ内の scripts ディレクトリ内の全てのファイルを -d <destination_path> で指定したディレクトリにコピーします。

-s <source_path> にはコピー元の abs_agent 動作環境ディレクトリを指定します。指定したディレクトリが存在しない場合にはエラーになります。通常は最新のインストールキットのオリジナルファイルを元に既存や新規の動作環境を更新するので、現在の作業ディレクトリがインストールディレクトリの場合には "-s . " を指定します。

-d <destination_path> にコピー先のディレクトリを指定します。指定したディレクトリが存在しない場合にはエラーになります。

agent_copycfg でファイル・ディレクトリコピー実行時には下記の動作を行います。
項目 ●(xxxxx) 部分の文字列は -v オプション指定時にコンソールにログ表示するときのタグ"名です。

● (MKDIR) ディレクトリ作成
<source_path> 内の "scripts" ディレクトリ内にある全てのディレクトリとサブディレクトリを <destination_path> ディレクトリ以下に同じディレクトリ名で作成する。既にコピー先に同じディレクトリが存在する場合には既存のディレクトリのまま更新しない。

● (COPY) ファイルコピー
<source_path> 内の "scripts" ディレクトリ内にある全てのファイルを <destination_path> ディレクトリ以下に同じファイル名でコピーする。この時、コピー先に同じファイル名のファイルが存在する場合には上書きする。

● (APPEND) ユーザースクリプトのカストマイズ部分をコピー元スクリプト終端に追加してからコピーする。コピー元ファイル名の拡張子が ".lua" で、コピー先に "_app_<コピー元ファイル名>.lua" のファイルが既に存在する場合に、"_app_<コピー元ファイル名>.lua" の内容をコピー元ファイルの最後尾に追加したファイルをコピー先に作成する。

● (OVERRIDE) ユーザースクリプトをコピー元スクリプトに代わって上書きコピーする。
コピー元ファイル名の拡張子が ".lua" の時に、コピー先に "_cpy_<コピー元ファイル名>.lua" のファイルが既に存在する場合には、"_cpy_<コピー元ファイル名>.lua" ファイルを、そのままの内容でコピー先に作成する。この場合には、コピー元ファイルの内容はコピー先に一切反映されない。

"-t" オプション指定時は、実際のファイルコピーを行わずに <source_path>/scripts 内のファイルが <destination_path>/scripts 以下にあるファイルの内容と一致しているかをチェックする。もしファイル内容が一致しない場合にはメッセージ "[!] **** <-> ****" (**** はファイルパス名)をコンソールに出力する。この時、コピー時の(APPEND),(OVERRIDE) 動作用に使用するファイルが <destination_path>/scripts 以下に存在する場合にはそのファイル名も出力する。<source_path>/scripts 内のファイルに対応するファイルが <destination_path>/scripts 以下に存在しない場合にはメッセージ "[?] **** --- ****"を出力する。

"-x" オプションを "-t" と同時に指定した場合には、<source_path>/scripts 内のファイル内容が一致する <destination_path>/scripts 側のファイルを削除する。ファイル削除後に <destination_path>/scripts 内に残された空のディレクトリを削除したい場合には、下記コマンド例を参考にしてください。

```
$ find <destination_path>/scripts -type d -empty -delete
```

これらの操作を行う事で、<destination_path>/scripts 内には更新または新規に作成済みのスクリプトだけが残された状態になります。その後、それらのスクリプトファイルの内容を編集して "_app_xxxx.lua" や

"_cpy_xxxx.lua" ファイルの作成を行うことで、セットアップ済みのアーカイブ・キットの作成が簡単にできます。

"-c" オプションを指定すると、<source_path> 内の abs_agent.xml ファイルと masters.xml ファイルも <destination_path> ディレクトリにコピーする。ただしこれらのファイルがコピー先に既に存在する場合には上書きしないで警告メッセージを出力するだけになる。

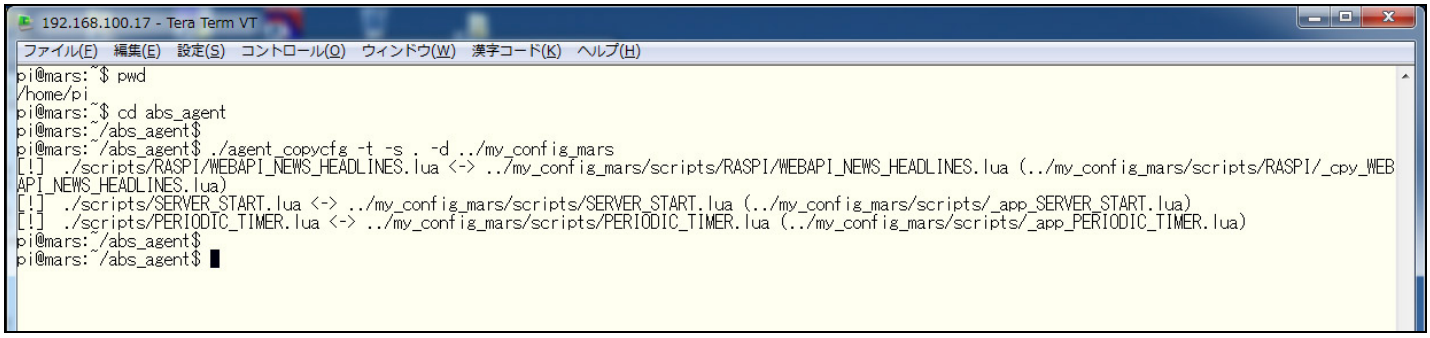
"-w" オプション指定時は <source_path> 内の "webroot" ディレクトリとその中に格納されている全てのファイルディレクトリをコピーする。この時、コピー先に同じファイル名のファイルが存在する場合には上書きする。

- **使用例**

● abs_agent 動作環境を "my_config" ディレクトリ以下にコピーする。

```
pi@mercury:~$ pwd
/home/pi
pi@mercury:~$ mkdir my_config
pi@mercury:~$ cd abs_agent
pi@mercury:~/abs_agent$ pwd
/home/pi/abs_agent
pi@mercury:~/abs_agent$ ./agent_copycfg -s . -d ../my_config -w -c
pi@mercury:~/abs_agent$ cd ../my_config
pi@mercury:~/my_config$ ls -Fa
./  ../  abs_agent.xml  masters.xml  scripts/  webroot/
```

● abs_agent がユーザ環境ディレクトリ "my_config_mars" ディレクトリで動作中で、このユーザー環境用に変更されたスクリプトファイル一覧を取得する。この時、インストールキットとプログラムは ~/abs_agent 以下にインストールしてあるものとする。



（インストールキットのデフォルトスクリプトファイルから内容が更新されたスクリプトファイル一覧を得る）

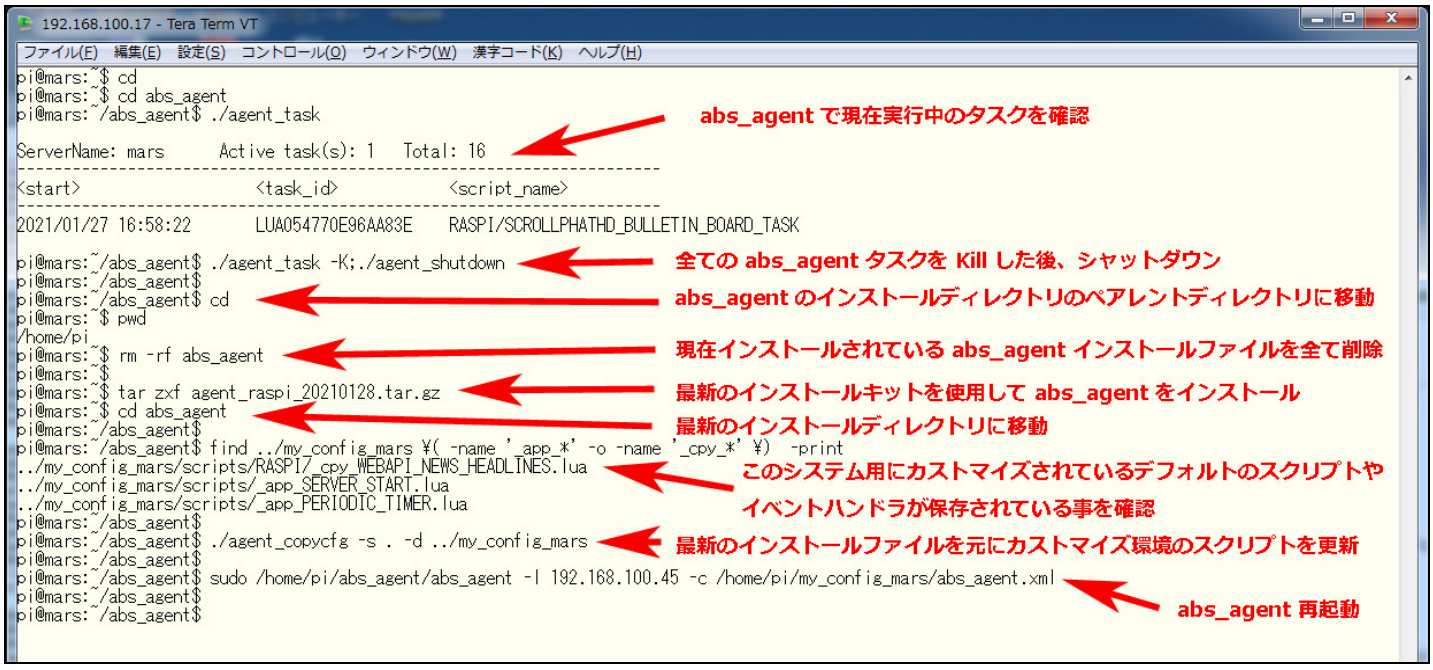
"-t" オプションで表示したスクリプトファイルは、"-t" オプション無しで agent_copycfg を実行すると "-s"

で指定したパスにあるインストールキット中のデフォルトスクリプトによって置き換えられます。

ファイル名後ろの () 内に "_app_" または "_cpy_" で始まるファイルが表示される場合があります。これらのファイルはユーザーによってスクリプト修正部分を書き出したものが保存されていることを示します。これらのファイルが存在する場合には、agent_copycfg でコピーを行う時に (APPEND) や (OVERRIDE) 機能によって自動的にインストールキット中のスクリプトに修正(追加または置き換え)されてからコピーされます。

> ⚠ **インストールキットを使用して既存の動作環境をアップデートする場合の注意**
> 既存の abs_agent 中のスクリプトファイルを agent_copycfg コマンドを使用してアップデートする場合には、インストールキットに含まれるデフォルトスクリプトからユーザー環境用に修正を加えたファイルについては、これらの修正部分を抽出した "_app_" または "_cpy_" で始まるスクリプトファイルを必ず作成してください。インストールキットには含まれないスクリプトファイルやサブディレクトリをユーザーが独自に作成している場合にはそのままで構いません。agent_copycfg コマンドは "-s" で指定したディレクトリ以下にあるファイルと同一名のパス上にあるファイルのみを更新して、それ以外のファイルやディレクトリはそのままの状態にします。

● ユーザ環境ディレクトリ "my_config_mars" ディレクトリで動作中の abs_agent をアップデートする。この時、既存のインストールキットとプログラムは ~/abs_agent 以下にインストールしてあるものとする。またサーバー設定ファイルやマスターファイルはユーザー環境ディレクトリに格納済みのものを引き続き使用する。スクリプトファイルがデフォルトのインストールキットからユーザー環境用にカストマイズされている部分については、_app_<スクリプトファイル名> や _cpy_<スクリプトファイル名> に書き出し済みであるとする。



(agent_copycfg コマンドを使用して既存動作環境を最新版のインストールキットで更新する)

> ℹ **abs_agent のアップデートで更新されるデフォルトスクリプトの確認**

最新の abs_agent インストールキットを展開した後、キットに含まれるデフォルトスクリプトをユーザー環境にコピーする前に前項の例の様に、agent_copycfg コマンドを "-t" オプション付で実行することで、更新対象のデフォルトスクリプトファイルとユーザーが自身がカストマイズしたスクリプトを事前に確認することができます。最新のインストールキットで更新されるデフォルトスクリプト(preload ライブラリを含む)は、ユーザーが既存動作環境でカストマイズしていない場合にはそのまま上書きしても問題ありません。

12.18 agent_serial (シリアルポート操作)

- **機能説明**

abs_agent に設定されているシリアルポートの操作を行います。agent_serial コマンドは、主にバッファーモードを有効にしている場合に利用可能な機能を提供します。

シリアルポートの文字列バッファからデータを取り出したり、シリアルポートからデータ送信(バッファーモードが無効の場合でも可)を行うことができます。文字列バッファを強制的にクリアすることもできます。

リモート側に設置した abs_agent に対してシリアルポートの操作をすることもできます。

- **コマンド実行形式**

(シリアルポート・リスト表示)

agent_serial *[-r <hostname>]*

(シリアルポート・文字列バッファの内容を１つ取り出す。-a を指定するとバッファ内全ての内容を取り出す)

agent_serial -p <serial_port> *[-a]* *[-r <hostname>]*

(シリアルポート・文字列バッファクリア)

agent_serial -p <serial_port> -d *[-r <hostname>]*

(シリアルポートから文字列を送信する。終端文字 CR(0x0D)を自動的にアペンドしてから送信する)

agent_serial -p <serial_port> -t <text> *[-r <hostname>]*

(シリアルポートからバイナリデータを送信する。バイナリデータは 16進数文字列で指定する)

agent_serial -p <serial_port> -x <hexstr> *[-r <hostname>]*

- **コマンドパラメータ**

-p <serial_port>

abs_agent 設定ファイル(abs_agent.xml)に指定したシリアルポートのデバイスファイル名、もしくはタイトル名を指定する。

シリアルポート名一覧は agent_serial をオプション無し ("-r <hostname>"は指定可)で実行すると得られます。

-a

シリアルポートの文字列バッファの内容を全て取り出す。

このオプションを指定しない場合には、文字列バッファ内の一番古いデータを1つ取り出す。

-d

シリアルポートの文字列バッファの内容をクリアする。

-t <text>

シリアルポートから <text> に指定した文字列を送信する。文字列の終端には CR(0x0d) が自動的に追加される。

-x <hexstr>

シリアルポートから <hexstr> に指定した16進数文字列をバイナリに変換して送信する。

-r <hostname>

ローカルコンピュータで動作している abs_agent の代わりに<hostname> で指定した、リモートに設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent サーバーに設定されているシリアルポートの一覧やデータの送受信を行います。abs_agent のスクリプトやイベントハンドラからシリアルポート用のライブラリ関数で操作するのと同等の機能です。シリアルポート用ライブラリ関数 API の項も参照してください。

シリアルポート一覧は agent_serial コマンドをパラメータ無し("-r <hostname>"は指定可) で実行するとコンソールに出力されます。

シリアルポートがバッファーモードに設定されている場合には、受信したデータが文字列バッファに格納されています。agent_serial コマンドではこの文字列バッファの内容をコンソールに出力できます。取り出した文字列は、シリアルポートの文字列バッファから自動的に取り除かれます。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "mercury") に設定されているシリアルポート一覧を表示する。

```
pi@mercury:~/abs_agent$ ./agent_serial

ServerName: mercury      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  TWELite0      /dev/ttyUSB0     STRING     True       0          115200     8          1          NONE       NONE
  LoopBack      /dev/ttyUSB1     STRING     True       0          115200     8          1          NONE       NONE
pi@mercury:~/abs_agent$
```

● リモートホスト(サーバー名 "raspi3") に設定されているシリアルポート一覧を表示する。

```
pi@mercury:~/abs_agent$ ./agent_serial -r 192.168.100.15

ServerName: raspi3      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  XBee_ZB       /dev/ttyUSB0     ZB         False      0          9600       8          1          NONE       NONE
  XBee_raspi3   /dev/ttyUSB1     XBEE       False      0          9600       8          1          NONE       NONE
pi@mercury:~/abs_agent$
```

● localhost(サーバー名 "mercury") のシリアルポートから1文字列を取り出す。

```
pi@mercury:~/abs_agent$ ./agent_serial

ServerName: mercury      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  TWELite0      /dev/ttyUSB0     STRING     True       50         115200     8          1          NONE       NONE
  LoopBack      /dev/ttyUSB1     STRING     True       0          115200     8          1          NONE       NONE
pi@mercury:~/abs_agent$ ./agent_serial -p TWELite0
```

```
ServerName: mercury      PortName: TWELite0      Remain: 49
-------------------------------------------------------------------------------------------------------------------------
<serial_buffer>
-------------------------------------------------------------------------------------------------------------------------
これは試験データです ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890 <>!"$%&()/  #1
pi@mercury:~/abs_agent$ ./agent_serial

ServerName: mercury      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  TWELite0      /dev/ttyUSB0     STRING     True       49         115200     8          1          NONE       NONE
  LoopBack      /dev/ttyUSB1     STRING     True       0          115200     8          1          NONE       NONE
pi@mercury:~/abs_agent$
```

● localhost(サーバー名 "mercury") のシリアルポートから全文字列を取り出す。

```
      pi@mercury:~/abs_agent$ ./agent_serial

ServerName: mercury      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  TWELite0      /dev/ttyUSB0     STRING     True       49         115200     8          1          NONE       NONE
  LoopBack      /dev/ttyUSB1     STRING     True       0          115200     8          1          NONE       NONE
pi@mercury:~/abs_agent$ ./agent_serial -p TWELite0 -a

ServerName: mercury      PortName: TWELite0      Length: 49
-------------------------------------------------------------------------------------------------------------------------
<serial_buffer>
-------------------------------------------------------------------------------------------------------------------------
これは試験データです ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890 <>!"$%&()/  #2
これは試験データです ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890 <>!"$%&()/  #3
..
..
これは試験データです ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890 <>!"$%&()/  #49
これは試験データです ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890 <>!"$%&()/  #50
pi@mercury:~/abs_agent$ ./agent_serial
```

```
ServerName: mercury      SerialPorts: 2
-------------------------------------------------------------------------------------------------------------------------
  <title>    <serial_port>     <type>   <buffered>  <records>  <baudrate> <data_bits> <stop_bits> <parity> <flow_cntl>
-------------------------------------------------------------------------------------------------------------------------
  TWELite0       /dev/ttyUSB0      STRING      True        0          115200     8          1          NONE       NONE
  LoopBack       /dev/ttyUSB1      STRING      True        0          115200     8          1          NONE       NONE
pi@mercury:~/abs_agent$
```

● localhost(サーバー名 "mercury") のシリアルポートから文字列を送信する。

```
pi@mercury:~/abs_agent$ ./agent_serial -p TWELite0 -t これは送信する文字列です
```

● localhost(サーバー名 "mercury") のシリアルポートからバイナリデータを送信する。

```
./agent_serial -p TWELite0 -x 0102030405060F0F1F2
```

12.19 agent_net (TCPServer,UDPServer管理、クライアントデータ送信)

- **機能説明**

  汎用 TCPサーバーと TCPクライアントの操作や管理機能を提供します。TCPサーバー管理機能では現在接続中のクライアント接続リスト表示や、TCPサーバーのインスタンスをリセットすることができます。

  TCPクライアント管理機能では、任意の TCPサーバーにバイナリデータを送信することが出来ます。オプションでサーバーからのリプライデータを受信することもできます。受信するデータサイズを明示的に指定することや、abs_agent コンフィギュレーション中の NETSERVER に指定されたフレーム構造(固定長+可変長)で計算されたデータ長で自動で計算することができます。

  任意の UDPサーバーにバイナリデータを送信することもできます。送信後にサーバーからリプライデータを受信することも可能で、この場合は受信する最大データサイズを明示的に指定します。

  TCPクライアントでは retain モードを指定することで、サーバー間のソケット接続を維持したままにすることもできます。agent_net コマンドを使用して、ratain モードで維持されているクライアント接続一覧を表示したり、タイムスタンプを指定して古い接続やエラーを検出しているクライアント接続を削除することができます。

汎用TCPサーバー,汎用UDPサーバーを利用する場合は、コンフィギュレーションで有効に設定してください

abs_agent インストールキットのデフォルト設定で、汎用 TCPサーバー機能と汎用UDPサーバーは無効に設定されています。有効にする場合は NETSERVER/TCPServerEnabled, NETSERVER/UDPServerEnabledを True に変更してから abs_agent を再起動してください。クライアント機能は常に使用することができます。詳しくは "サーバープログラム・設定ファイル"章中の NETSERVER項を参照してください。

- **コマンド実行形式**

(TCPサーバーに接続中のクライアント一覧と retain モードで維持しているクライアント接続一覧を表示)

agent_net *[-r <hostname>]*

(TCPサーバーにバイナリデータを送信する。データ送信のみでリプライデータを受信しない場合や受信データサイズを明示的に指定する場合)

agent_net -h <tcp_host> -x <hexstr> *[-p <port>] [-c] [-l <read_len> [-t <timeout>]]*
[-r <hostname>]

(TCPサーバーにバイナリデータを送信する。データ送信後に受信するリプライデータサイズをコンフィギュレーションで指定したフレーム構造に従って自動計算させる場合)

agent_net -h <tcp_host> -x <hexstr> *[-p <port>] [-c]* -L *[-t <timeout>]]*
[-r <hostname>]

(UDPサーバーにバイナリデータを送信する。リプライで受信する最大サイズを指定する)

agent_net -d -h <udp_host> -x <hexstr> -p <port> *[-l <read_len> [-t <timeout>]] [-r <hostname>]*

(retain モードで維持している古いクライアント接続を削除する)

agent_net -R *[-r <hostname>]*

agent_net -u <datetime_until> *[-r <hostname>]*

(TCPサーバーをリスタートさせる)

agent_net -S *[-r <hostname>]*

- **コマンドパラメータ**

-h <tcp_host>, -h <udp_host>

データ送信先のサーバーホスト名または IP アドレスを指定する。

-x <hexstr>

サーバーに送信するバイナリデータを16進数文字列で指定する。

-p <port>

TCPサーバーまたは UDPサーバーに接続する時のポート番号を指定する。
このオプションを省略してかつ "-d" オプションを指定していない場合には、abs_agent コンフィギュレーションファイルの NETSERVER/TCPPort に設定されているポート番号を使用する(デフォルト設定値は 502)

-c

送信時に接続したクライアントからのソケット接続を維持したままにする。(retain モード)
〈tcp_host〉と〈port〉で指定したクライアント接続が既に作成済みの場合には、そのソケットを使用してデータ送信を行う。

-L, -l 〈read_len〉

データ送信後にサーバーから受信するデータバイト数を指定する。

[TCP送信時 ("-d" オプション指定無し)]
-l, -L パラメータを両方とも省略した場合や、-l パラメータでバイト数を 0 に指定した場合にはデータ受信は行わない。
-L を指定した場合には abs_agent コンフィギュレーションファイルの NETSERVER に設定されている〈固定長〉+〈可変長〉フォーマットに従ったバイト数を自動で計算してサーバーから受信する。

[UDP送信時 ("-d" オプション指定時)]
-l パラメータを省略した場合は 4096 が指定されます。
-lパラメータでバイト数を 0 に指定した場合にはデータ受信は行わない。

-t 〈timeout〉

-l パラメータと同時に使用して、データ受信時のタイムアウト時間(ms)を指定する。
パラメータ省略時は 500(ms) が指定される。

-R, -u 〈datetime_until〉

retain モードで維持されているクライアント接続で、最後に使用された時刻が〈datetime_until〉よりも古いものを削除する。クライアント接続のソケットでエラーを検出しているものも同時に全て削除する。-u の代わりに -R を指定すると全てのクライアント接続を削除する。

-S

abs_agent が作成した TCPサーバーインスタンスをリセットする。TCPサーバーに接続中のコネクションを全て削除した後、TCPサーバーを再起動させる。

-d

UDPデータグラムでデータを送受信する。
UDP送信時にリプライ受信無し("-l 0")を指定した場合には、-h 〈udp_host〉で指定する IP アドレスにブロードキャストアドレスを指定することもできる。

-r 〈hostname〉

ローカルコンピュータで動作している abs_agent の代わりに〈hostname〉で指定した、リモート

に設置した abs_agent プログラムにアクセスします。<hostname> にはホスト名または IP アドレスを指定します。このとき、リモート側の abs_agent プログラムがリモート・クライアントからのリクエストを受け付けるように、abs_agent が動作しているコンピュータで "agent_hosts -a <hostname>" コマンドを実行して、リモートアクセスするクライアントのホスト名を登録しておきます。

- **リターンコード**

0	正常終了した
1	コマンドパラメータが間違っています
2	実行時にエラーが発生した

- **説明**

abs_agent に設定されている汎用 TCPサーバーの接続リストや、TCP クライアントからのデータ送受信を行います。abs_agent のスクリプトやイベントハンドラから TCPサーバーや TCPクライアント用のライブラリ関数で操作するのと同等の機能です。

TCPクライアント機能につては tcp_send_recv_binary(), tcp_client_purge(), ライブラリ関数の項も参照してください。TCPサーバーがデータを受信した時にユーザーが処理を行う為のイベントハンドラについては、"イベント" 章中の TCPSERVER_DATA の項を参照してください。

リモートに設置した abs_agent プログラムを対象にする場合には "-r <hostname>" オプションを指定します。このとき、リモート側の abs_agent ではクライアントからのアクセスを許可しておく必要があります。アクセス許可については agent_hosts コマンドの項を参照してください。

- **使用例**

● localhost(サーバー名 "mercury") で現在 TCPサーバーに接続されているクライアント一覧と、クライアント送信時に保持(retain)されたクライアント接続一覧を表示する。

```
pi@mercury:~/abs_agent$ ./agent_net

ServerName: mercury      Enabled server(port): TCPServer(502) UDPServer(8090)

Server connection list:
--------------------------------------------------------------------------------
       <uid>              <start>          <client>
--------------------------------------------------------------------------------
TCP05DB6E88475248  2023/06/27 14:34:04    127.0.0.1
TCP05DB6E891B3C0E  2023/06/27 14:34:17    192.168.100.18
TCP05DB6E8995C4BF  2023/06/27 14:34:25    192.168.100.17
```

```
Client retain list:
-------------------------------------------------------------------------------
     <uid>          <ready>  <busy>       <reserved>         <server>
-------------------------------------------------------------------------------
CSK05DB6E88474E64      Yes      No       2023/06/27 14:34:04    localhost
CSK05DB6E891B3996      Yes      No       2023/06/27 14:34:17    192.168.100.18
```

● リモートホスト(サーバー名 "raspi3"、IPアドレス "192.168.100.15") に TCPパケット(ModbusTCP)を送信した後リプライパケットを受信する

```
pi@mercury:~/abs_agent$ ./agent_net -h 192.168.100.15 -c -x 000000000006010300180001 -L
Reply=00000000000050103020005
```

● リモートホスト(サーバー名 "raspi3"、IPアドレス "192.168.100.15") に UDPパケットを送信した後リプライパケットを受信する

```
pi@mercury:~/abs_agent$ ./agent_net -d -h 192.168.100.15 -p 8090 -x 1122334455AABBCCDDEE
Reply=1122334455AABBCCDDEE
```

● LAN内のブロードキャストアドレスに UDPパケットを送信する

```
pi@mercury:~/abs_agent$ ifconfig eth0 | grep broadcast
        inet 192.168.100.18  netmask 255.255.255.0  broadcast 192.168.100.255
pi@mercury:~/abs_agent$ ./agent_net -d -h 192.168.100.255 -I 0 -p 8090 -x 1122334455AABBCCDDEEFF
```

同一 LAN 内で動作中の 3 つの abs_agent で UDPパケットを受信したときのログ

```
2023/06/28 06:25:59 raspi3     UDPSERVER_DATA       0 received from 192.168.100.18 data = 1122334455AABBCCDDEEFF
2023/06/28 06:25:59 mercury    UDPSERVER_DATA       0 received from 192.168.100.18 data = 1122334455AABBCCDDEEFF
2023/06/28 06:25:59 mars       UDPSERVER_DATA       0 received from 192.168.100.18 data = 1122334455AABBCCDDEEFF
```

13 イベント

abs_agent ではサービスモジュールが予め決められた条件になった場合や、デバイス状態が変化した場合などにイベントが発生します。また、定期タイマー(PERIODIC_TIMER, TICK_TIMER)など定期的に繰り返し発生するイベントもあります。これらのイベントが発生すると、イベントの種類ごとに予め決められたイベントハンドラ(Lua スクリプト)が自動的に実行されます。

デフォルトのイベントハンドラのスクリプトファイルは、イベント名に拡張子 ".lua" をつけた名前で abs_agent をインストールしたディレクトリ内の "scripts" ディレクトリに保管されています。これらのデフォルト・イベントハンドラスクリプトでは、イベント発生をログに記録するだけ等の記述がされています。これらのスクリプトファ

イルは通常のテキストファイル(日本語を使用する場合には UTF-8[BOM無し])なので、テキストエディタで簡単に編集できます。ユーザーはこれらのイベントハンドラスクリプトをカスタマイズしてシステムを構築していきます。

デフォルト・イベントハンドラをユーザー環境に合わせて変更した場合は、その修正箇所を抽出したファイル("_app_" または "_cpy_" で始まる Lua スクリプトファイル)を作成することをお勧めします。これらのファイルを設置することで、abs_agent を最新のインストールキットでアップデートした時に、インストールキット内のデフォルト・イベントハンドラに自動で編集を行ってからユーザー環境にコピーすることができます。これらの詳しい内容は、クライアントプログラム章内の "agent_copycfg" の項を参照してください。

イベントハンドラとして実行する Lua スクリプトでは abs_agent が提供する API 関数や Lua 標準ライブラリで提供される関数を利用できます。また、イベントハンドラで実行中のスクリプトから、別のユーザー・スクリプトをコールすることもできます。このとき別スクリプトをマルチスレッドで平行して動作させることや、別スクリプトからリターン値を取得することも可能です。

イベントで実行されたスクリプトは、スクリプトファイルの終端まで実行することで処理が完了します。スクリプト中で定義された関数以外の部分で "return" 文を実行させることでスクリプトを終了させることもできます。
Lua の error() 関数や stat_check() ライブラリ関数でスクリプト実行を強制終了させることも可能で、この時にはエラー発生のログメッセージが出力されます。

イベントハンドラスクリプトを変更すると、その直後から有効になりますのでサーバーコンピュータ や abs_agent を再起動する必要はありません。ただし、SERVER_START, SERVER_STOPイベントハンドラを変更した場合には、そのスクリプト内容を実行するために、明示的に abs_agent の再起動が必要になります。

abs_agent ではイベントハンドラもユーザーが独自に作成するユーザースクリプトと同様に扱いますので、ユーザーが任意のタイミングで script_exec() ライブラリ関数などを使用してイベントハンドラスクリプトを実行することもできます。この場合にはイベントハンドラに渡すスクリプトパラメータを、コールする側で適切に与える必要があります。

13.1 PERIODIC_TIMER (1 分間隔)

- **イベント発生条件**

約 1 分毎に発生します。

- **イベントハンドラに渡されるパラメータ**

なし

- **備考**

PERIODIC_TIMER イベントは SERVER_START イベント後に有効になります。このため、PERIODIC_TIMER イベントハンドラのスクリプト中で初期設定が必要なスクリプト記述は SERVER_START イベントハンドラに記述してください。

イベントハンドラ例

```
file_id = "PERIODIC_TIMER"
--[[
*********************************************************************************
* イベントハンドラスクリプト実行時間について                                    *
*********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*********************************************************************************
]]
-- log_msg("start..",file_id)
```

13.2 TICK_TIMER (1 秒間隔)

- **イベント発生条件**

約 1 秒毎に発生します。

- **イベントハンドラに渡されるパラメータ**

なし

- **備考**

TICK_TIMER イベントは SERVER_START イベント後に有効になります。このため、TICK_TIMER イベントハンドラのスクリプト中で初期設定が必要なスクリプト記述は SERVER_START イベントハンドラに記述してください。

- **イベントハンドラ例**

```
file_id = "TICK_TIMER"
--[[
*********************************************************************************
* イベントハンドラスクリプト実行時間について                                    *
*********************************************************************************
TICK_TIMER スクリプトの実行は１秒以内で必ず終了するようにしてください。
TICK_TIMER イベントハンドラは別スレッドで実行されます。このため、処理時間がかかると、
平行して実行されるスレッド数が増え続けて OS がハングアップする原因になります。

*********************************************************************************
```

```
]]


local stat,cur_val = raspi_gpio_read(18)
if not stat then error() end
if not raspi_gpio_write(18,not cur_val) then error() end -- GPIO18 を１秒毎にブリンク
```

13.3 SERVER_START (abs_agent起動時)

- **イベント発生条件**

abs_agent 起動時にイベントが一回だけ発生します。

このイベント発生時には、他のサービスモジュールで管理しているリソースやデバイスも有効になっていますので、システムの初期設定等に使用できます。

- **イベントハンドラに渡されるパラメータ**

無し

- **備考**

- **イベントハンドラ例**

```
file_id = "SERVER_START"
log_msg("start..",file_id)
--[[
********************************************************************************
MQTT モジュールが有効な場合にエンドポイントの接続を開始させる
********************************************************************************
]]
local mstat,module_stat = service_module_status()
if not mstat then error() end
if module_stat["MQTT"] then
        script_fork_exec("MQTT_CONNECT_ALL","","")
end


--------------------
-- GPIO 入出力設定
--------------------
if not raspi_gpio_config(18,"output","off") then error() end
if not raspi_gpio_config(22,"input","pullup") then error() end
if not raspi_gpio_config(23,"input","pullup") then error() end
----------------------------
-- GPIO ChangeDetect 設定
```

```
----------------------------
if not raspi_change_detect(22,true) then error() end
if not raspi_change_detect(23,true) then error() end
```

13.4 SERVER_STOP (abs_agent終了時)

- **イベント発生条件**

agent_shutdown コマンドを使用して abs_agent を終了させた時にイベントが発生します。

このイベント発生時には、abs_agent のサービスモジュールで管理しているリソースやデバイスは使用可能ですので、システムの終了処理等に使用できます。

- **イベントハンドラに渡されるパラメータ**

無し

- **備考**

agent_shutdown クライアントプログラムを実行したときにサーバー側でこのイベントが発生します。

このイベントはシステムのシャットダウン時などに、abs_agent 終了処理の直前に発生します。このイベントハンドラスクリプト中では abs_agent の全てのリソースが使用可能になっていますので、デバイス等の終了処理が必要な場合に、このイベントハンドラを使用できます。

このイベントハンドラ実行終了後に全てのリソースが使用できなくなります。このため、このスクリプトから他のスクリプトを実行する場合には script_exec() を使用してください。script_fork_exec() 等によって別スレッドでは実行しないでください。ただし、別コンピュータで動作している abs_agent のスクリプトを script_fork_rexec() でコールすることはできます。

OS を "shutdown" コマンドで終了した場合等、agent_shutdown プログラムを実行しない場合には、このイベントは発生しません。

- **イベントハンドラ例**

```
file_id = "SERVER_STOP"
--[[
*******************************************************************************
このスクリプトは abs_agent 終了時にコールされます
このスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。

このスクリプトが終了した直後に、abs_agent がシャットダウンして
全てのリソースが使用できなくなります。このため、このスクリプトから他のスクリプト
```

```
を実行する場合には、絶対に別スレッドで実行しないでください。
   ********************************************************************************
]]
log_msg("start..",file_id)
```

13.5 SERIAL_STRING (シリアルポートから文字列を受信)

- **イベント発生条件**

SERIALサービスモジュールで、シリアルポートから文字列を受信した時に発生します。

サーバー設定ファイルで定義するシリアルデバイスのデバイスタイプ

"/Document/Class/Serial/DeviceList/Item#n/Type" を "STRING" にして下さい。またバッファー入力モード

"/Document/Class/Serial/DeviceList/Item#n/BufferedMode" は False にして、データ受信時にイベントが発生するモードにしておきます。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
COMPort	シリアルデバイスの COM ポート名	"/dev/ttyUSB0"
Title	シリアルデバイスに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、このパラメータは渡されません。	"コントローラ#1"
STRING	COM ポートから入力された文字列データ。 文字列は、下記の何れかのデータで終端されたものです。 (ヌル文字 [0x00], CR [0x0D],LF [0x0A], CR-LF[0x0D,0x0A] STRING パラメータには､終端文字を含まない文字列部分が格納されています	"Hello World!!"

- **備考**

シリアルポートから文字列を受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。スクリプト処理中に同一のイベントが発生した場合は、平行して複数のスクリプトを実行します。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

イベントハンドラ例

```
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["STRING"],file_id);
```

13.6 SERIAL_FIRMATA (シリアルポートからFIRMATAパケット受信)

- **イベント発生条件**

SERIALサービスモジュールで、シリアルポートからFIRMATA パケットデータを受信した時に発生します。

サーバー設定ファイルで定義するシリアルデバイスのデバイスタイプ

"/Document/Class/Serial/DeviceList/Item#n/Type" を "FIRMATA" にして下さい。またバッファー入力モード "/Document/Class/Serial/DeviceList/Item#n/BufferedMode" は False にして、データ受信時にイベントが発生するモードにしておきます。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
COMPort	シリアルデバイスの COM ポート名	"/dev/ttyUSB0"
Title	シリアルデバイスに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、このパラメータは渡されません。	"コントローラ#1"
FIRMATA_DATA	COM ポートから入力されたFIRMATA パケットデータ。 バイナリデータを16進数文字列に変換したものが格納されます	"F07902035400650073 0074005300740061006 E006400610072006400 4600690072006D00610 074006100F7"

- **備考**

シリアルポートからFIRMATA パケットデータを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。スクリプト処理中に同一のイベントが発生した場合は、平行して複数のスクリプトを実行します。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

イベントハンドラ例

```
-------------------------------------
-- FIRMATA イベントデータをログに出力
-------------------------------------
local str = ""
for key,val in pairs(g_params) do
  str = str .. key .. " = " .. val .. " "
end
   log_msg(str,file_id);
-------------------------------------------------------------------------------------------------------------
-- parse_firmata() を使用して、FIRMATA フレームデータを解析して文字列データと配列データに変換しています。
-- parse_firmata() ライブラリ関数の詳しい仕様とソースファイルは、
-- "C:¥Program Files¥AllBlueSystem¥Scripts¥preload" を参照してください
-------------------------------------------------------------------------------------------------------------
local stat,msg,data = parse_firmata(g_params["FIRMATA_DATA"])
if stat then
  ------------------------------------------------------------
```

```
  -- FIRMATA フレームの解析が成功したら変換後の文字列をログに出力
  ------------------------------------------------------------------
  for key,val in pairs(msg) do
    log_msg("msg[" .. key .. "]=" .. val,file_id);
  end
  ------------------------------------------------------------------
  -- FIRMATA フレームにデータ配列が入っていた場合には、ログに出力
  ------------------------------------------------------------------
  if rdata ~= nil then
    for key,val in ipairs(data) do
      log_msg("data[" .. tostring(key) .. "]=" .. bit_tohex(val,2),file_id);
    end
  end
end
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

13.7 SERIAL_XBEE (XBee 802.15.4 APIフレームデータ受信)

- **イベント発生条件**

SERIALサービスモジュールで、シリアルポートから XBee API フレームを受信した時に発生します。

サーバー設定ファイルで定義するシリアルデバイスのデバイスタイプ

"/Document/Class/Serial/DeviceList/Item#n/Type" を “XBEE” にして下さい。Type を "XBEE" に設定するとバッファー入力モード"/Document/Class/Serial/DeviceList/Item#n/BufferedMode" の設定は無視され、常に False に解釈され XBee API フレーム受信時にイベントが発生します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
COMPort	シリアルデバイスの COM ポート名	“/dev/ttyUSB0”
Title	シリアルデバイスに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、このパラメータは渡されません。	“コントローラ#1”
FRAME_DATA	COM ポートから入力された XBee API フレームデータ。 バイナリデータを16進数文字列に変換したものが格納されます	“7E0008810C033C0001 02032D”

- **FRAME_DATAで渡された API フレームを解析して作成される変数**

下記の変数は、SERIAL_XBEE イベントハンドラ中に記述されたデフォルトスクリプトで作成されるデータ構造です。これらの変数を利用すると XBee フレームタイプが 0x80, 0x81 の場合に、ペイロードデータの内容を簡単に扱うこ

とができます。

変数名	説明	値の例
SenderNodeID SenderAddr64 SenderAddr16	FRAME_DATA に格納されたフレームデータ中のペイロード部分を解析して送信元ノードのノード名とアドレス（16進数文字列）に変換したもの。	SenderNodeID:"Device1" SenderAddr64:"0013A200404AC39C" SenderAddr16:"0A01"
PayloadString	FRAME_DATA に格納されたフレームデータ中のペイロード部分を文字列に変換したもの。文字列がペイロードに含まれていない場合には 空文字列が設定される。	"Hello World!!"
frame_arr	FRAME_DATA で渡されたフレームデータ全体を数値配列（バイト列）に変換したもの	

- **備考**

シリアルポートから XBee API フレームを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。スクリプト処理中に同一のイベントが発生した場合は、平行して複数のスクリプトを実行します。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

serial_write() ライブラリ関数を使用して XBee API コマンドを XBee モジュールに送信した場合に、XBee モジュールから返されたリプライフレームをシリアルポートで受信しますが、このとき SERIAL_XBEE イベントハンドラは起動しません。このときの リプライフレームは serial_write() ライブラリ関数の戻り値で取得できます。コマンド・レスポンスフレーム(API Identification = 0x88 or 0x97 or 0x89) 以外の API フレームを受信した場合にのみこのイベントハンドラが起動します。

イベントハンドラ例

```
file_id = "SERIAL_XBEE"
--------------------------------------------------------------------------------
-- XBee Packet のフレームデータからイベントハンドラ内で使用可能な変数を設定
--------------------------------------------------------------------------------
local frame_arr = hex_to_tbl(g_params["FRAME_DATA"])
local PayloadString = ""
local SenderNodeID,SenderAddr64,SenderAddr16,stat,is_local
if (frame_arr[4] == 0x80) or (frame_arr[4] == 0x81) then
  local AddrHex
  -- SenderNodeID, SenderAddr64, SenderAddr16
  if frame_arr[4] == 0x80 then
    AddrHex = tbl_to_hex(frame_arr,5,12)
  else
    AddrHex = tbl_to_hex(frame_arr,5,6)
```

```
  end
  stat, SenderAddr64, SenderAddr16, SenderNodeID, is_local = xbee_find_device(g_params["COMPort"],AddrHex)
  -- PayloadString
  local TmpArr = {}
  local payload_ptr
  if frame_arr[4] == 0x80 then
    payload_ptr = 15 -- 64bit RX Packet
  else
    payload_ptr = 9             -- 16bit RX Packet
  end
  for i = payload_ptr,#frame_arr - 1,1 do
    if (frame_arr[i] >= 0x20) and (frame_arr[i] <= 0x7E) then
      table.insert(TmpArr,frame_arr[i])
    end
  end
  PayloadString = tbl_to_str(TmpArr)
end
--------------------------------------------------------------------------
-- 受信したパケットのフレームデータまたはペイロード部分の文字列をログ出力
--------------------------------------------------------------------------
if PayloadString ~= "" then
  log_msg(SenderNodeID .. "[" .. g_params["COMPort"] .. "] " .. PayloadString,file_id)
else
  log_msg(SenderNodeID .. "[" .. g_params["COMPort"] .. "] " .. g_params["FRAME_DATA"],file_id)
end
local mframe_cnt = tonumber(g_params["MULTI_FRAME_COUNT"])
for i = 1,mframe_cnt,1 do
  log_msg(SenderNodeID .. "[" .. g_params["COMPort"] .. "] mframe[" .. tostring(i) .. "/" ..
          tostring(mframe_cnt) .. "] " .. g_params["MULTI_FRAME_DATA_" .. tostring(i)],file_id)
end
```

13.8 SERIAL_ZB (XBee-ZB Zigbee APIフレームデータ受信)

- **イベント発生条件**

SERIALサービスモジュールで、シリアルポートから XBee-ZB API フレームを受信した時に発生します。

サーバー設定ファイルで定義するシリアルデバイスのデバイスタイプ

"/Document/Class/Serial/DeviceList/Item#n/Type" を “ZB” にして下さい。Type を "ZB" に設定するとバッファー入力モード"/Document/Class/Serial/DeviceList/Item#n/BufferedMode" の設定は無視され、常に False に解釈

され XBee-ZB API フレーム受信時にイベントが発生します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
COMPort	シリアルデバイスの COM ポート名	“/dev/ttyUSB0”
Title	シリアルデバイスに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、このパラメータは渡されません。	“コントローラ#1”
FRAME_DATA	COM ポートから入力された XBee-ZB API フレームデータ。 バイナリデータを16進数文字列に変換したものが格納されます	“7E0008810C033C0001 02032D”

- **FRAME_DATAで渡された API フレームを解析して作成される変数**

下記の変数は、SERIAL_ZB イベントハンドラ中に記述されたデフォルトスクリプトで作成されるデータ構造です。これらの変数を利用すると XBee-ZB フレームタイプが 0x90 の場合に、ペイロードデータの内容を簡単に扱うことができます。

変数名	説明	値の例
SenderNodeID SenderAddr64 SenderAddr16	FRAME_DATA に格納されたフレームデータ中のペイロード部分を解析して送信元ノードのノード名とアドレス（16進数文字列）に変換したもの。	SenderNodeID:"Device1" SenderAddr64:"0013A200404AC39C" SenderAddr16:"0A01"
PayloadString	FRAME_DATA に格納されたフレームデータ中のペイロード部分を文字列に変換したもの。文字列がペイロードに含まれていない場合には 空文字列が設定される。	"Hello World!!"
frame_arr	FRAME_DATA で渡されたフレームデータ全体を数値配列（バイト列）に変換したもの	

- **備考**

シリアルポートから XBee-ZB API フレームを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。スクリプト処理中に同一のイベントが発生した場合は、平行して複数のスクリプトを実行します。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

serial_write() ライブラリ関数を使用して XBee-ZB API コマンドを XBee-ZB モジュールに送信した場合に、XBee モジュールから返されたリプライフレームをシリアルポートで受信しますが、このとき SERIAL_ZB イベントハンドラは起動しません。このときの リプライフレームは serial_write() ライブラリ関数の戻り値で取得できます。

コマンド・レスポンスフレーム(API Identification = 0x88) 以外の API フレームを受信した場合にのみこのイベントハンドラが起動します。ただし例外として、0x88 のコマンドレスポンスデータでマルチフレームを構成してい

るデータを受信した場合には、このイベントハンドラが起動されます。マルチフレームデータはデフォルトで記述されているイベントハンドラスクリプトによってグローバル共有文字列リストに格納された後、zb_at_command() のリプライデータを作成するときに使用されます。

イベントハンドラ例

```
file_id = "SERIAL_ZB"
--------------------------------------------------------------------------------
-- XBee Packet のフレームデータからイベントハンドラ内で使用可能な変数を設定
--------------------------------------------------------------------------------
local frame_arr = hex_to_tbl(g_params["FRAME_DATA"])
local PayloadString = ""
local stat,SenderAddr64, SenderAddr16
local SenderNodeID = ""
if (frame_arr[4] == 0x90) then
    SenderAddr64 = tbl_to_hex(frame_arr,5,12)
    SenderAddr16 = tbl_to_hex(frame_arr,13,14)
    local found,dev = master_item_find("ZBDevice","SerialNumber",SenderAddr64)
    if found then
        SenderNodeID = dev["NodeIdentifier"]
    end
    -- PayloadString
    local TmpArr = {}
    for i = 16,#frame_arr - 1,1 do
        if (frame_arr[i] >= 0x20) and (frame_arr[i] <= 0x7E) then
            table.insert(TmpArr,frame_arr[i])
        end
    end
    PayloadString = tbl_to_str(TmpArr)
end
--------------------------------------------------------------------------------
-- マルチフレーム受信時に zb_at_local() 関数に値を返すための文字列リストを構築する
--------------------------------------------------------------------------------
if (frame_arr[4] == 0x88) and (frame_arr[6] == 0x4E) and (frame_arr[7] == 0x44) then -- "ND" response?
    local req_flag = "ZB_STORE_REPLY_" .. g_params["COMPort"] .. "_" .. bit_tohex(frame_arr[5],2)
    local stat,flag = get_shared_data(req_flag)
    if flag ~= "" then
        add_shared_strlist("ZB_REPLY_MFRAME_" .. g_params["COMPort"] .. "_" ..
bit_tohex(frame_arr[5],2),g_params["FRAME_DATA"])
    end
```

```
end
--------------------------------------------------------------------------
-- 受信したパケットのフレームデータまたはペイロード部分の文字列をログ出力
--------------------------------------------------------------------------
if PayloadString ~= "" then
    log_msg(SenderNodeID .. "[" .. g_params["COMPort"] .. "] " .. PayloadString,file_id)
else
    log_msg(SenderNodeID .. "[" .. g_params["COMPort"] .. "] " .. g_params["FRAME_DATA"],file_id)
end
```

13.9 SERIAL_RAW (シリアルポートからデータ受信)

- **イベント発生条件**

SERIALサービスモジュールで、シリアルポートからデータを受信した時に発生します。

サーバー設定ファイルで定義するシリアルデバイスのデバイスタイプ

"/Document/Class/Serial/DeviceList/Item#n/Type" を “RAW” にして下さい。またバッファー入力モード

"/Document/Class/Serial/DeviceList/Item#n/BufferedMode" は False にして、データ受信時にイベントが発生するモードにしておきます。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
COMPort	シリアルデバイスの COM ポート名	“/dev/ttyUSB0”
Title	シリアルデバイスに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、このパラメータは渡されません。	“コントローラ#1”
RAW_DATA	COM ポートから入力されたデータ。 バイナリデータを16進数文字列に変換したものが格納されます	“41424300”

- **備考**

データを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。スクリプト実行中に同一のイベントが発生した場合は、平行して複数のスクリプトを実行します。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

RAW_DATA パラメータに設定されるデータは、シリアルポートから入力されたデータが加工されずにそのままバイナリデータとして格納します。連続してシリアルポートからデータを入力した場合には、OS や ドライバで複数のバイナリデータの固まりとして処理します。このイベントハンドラではそれらのデータの固まりをそのまま RAW_DATA パラメータに格納して、SERIAL_RAW イベントを複数回実行します。

イベントハンドラ例

```
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["RAW_DATA"],file_id);
```

13.10 GLOBAL_WATCH (監視対象のグローバル共有変数更新)

- **イベント発生条件**

監視対象に設定したグローバル共有変数が変化(追加、修正、削除)した時に発生します。

この機能を使用する場合には、サーバー設定ファイル中の "UseGlobalWatch" 項目を True に設定してください。(インストール直後のデフォルト値は False になっています)

グローバル共有変数を監視対象にするには、abs_agent のグローバル共有文字列リストのチャンネル名 "$GLOBAL_WATCH" に、監視対象としたいグローバル共有変数のキー名(文字列)を追加します。

監視対象から外す場合には、abs_agentのグローバル共有文字列リスト中のチャンネル名 "$GLOBAL_WATCH" から、対象となるグローバル共有変数キー名を削除します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
SharedDataKey	変化したグローバル共有変数名	DeviceStatus
SharedDataValue	変化したグローバル共有変数の値	100

- **備考**

スクリプトやイベントハンドラ、Web API、クライアントプログラムから監視対象となっているグローバル共有変数の内容を更新すると、イベントが発生してこのイベントハンドラが起動されます。

監視対象となっているグローバル共有変数を更新するライブラリ関数や WebAPI等は、このイベントハンドラの実行が終了するまで待たされますので、イベントハンドラの処理はできるだけ早く完了させるようにしてください。イベントハンドラ中で時間がかかる処理を行う場合には、処理を行うスクリプトを別に作成して、別スレッドで処理を行う様にします。

監視対象のキー名を設定した共有データ文字列リスト"$GLOBAL_WATCH" はサーバー再起動時には常にクリアされます。監視対象の変数をサーバー起動時に自動設定したい場合には、SERVER_START イベントハンドラに add_shared_strlist() ライブラリ関数を使用して記述します。

> **注意 GLOBAL_WATCHイベントハンドラ中で、監視対象となったグローバル共有変数を変更してはいけません**
> イベントハンドラ内または、イベントハンドラから呼び出すスクリプト中から g_params["SharedDataKey"] で表されたグローバル共有変数の値を絶対に更新しないでください。イベントハンドラが繰り返しコールされてシステムがハングアップする可能性があります

- **イベントハンドラ例**

```
file_id = "GLOBAL_WATCH"
log_msg("start..",file_id)
-----------------------------------------
-- リクエストパラメータをログに出力する --
-----------------------------------------
for key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end
```

13.11 MQTT_KEEP_ALIVE_TIMER (MQTT-KeepAliveTimer間隔)

- **イベント発生条件**

サーバー設定ファイル中で MQTT サービスモジュールを有効にしたときに、KeepAliveTimer 項目に設定した間隔で発生します。

- **イベントハンドラに渡されるパラメータ**

なし

- **備考**

このイベントハンドラにはインストール時に下記の動作を行うようにスクリプトが設定されています。これらの処理内容はユーザーのアプリケーションに合わせて自由にカストマイズしてください。デフォルトのイベントハンドラでは、サーバー設定ファイルで定義された全ての MQTTエンドポイント接続に対して以下の処理を行います。

● MQTT ブローカーに現在接続中で、かつソケットエラーが発生していないものにに対して、mqtt_ping() ライブラリ関数を使用して MQTT PINGREQ メッセージを送信します。

● MQTT ブローカー接続中に何らかの原因で通信エラーが発生していた場合には、一旦そのエンドポイントのソケットをクローズした後、再び MQTTブローカーに再接続を行います。

ユーザーはこのイベントハンドラスクリプトを変更することで、エンドポイントでエラーが発生した場合にアラーム装置に通知を行ったり、警報メールを送信する処理を組み込むことができます。

イベントハンドラ例

```
file_id = "MQTT_KEEP_ALIVE_TIMER"
--[[
●機能概要
現在ソケット接続中でかつエラーが発生していない MQTT エンドポイントに対して MQTT PINGREQ メッセージを
送信します。ソケットエラーが発生している MQTTエンドポイントに関しては、ブローカに再接続を試みます。
ソケット接続中ではない MQTT エンドポイントに関してはこのイベントハンドラではなにも行いません。
このスクリプトでは再接続を試みたときに、再びソケットエラーが発生したエンドポイントについては
```

接続不能と判断して、以降は再接続を試みないようになります。もし、永久にソケットの再接続を試みたい場合には下記のスクリプト部分を

```
-----------------------------------------------
-- エラーが発生しているエンドポイントを再接続する
-----------------------------------------------
for key,val in ipairs(id) do
  -- 現在ソケット接続中でエラーが発生しているエンドポイントを対象にする
  if enable[key] and (not ready[key]) then
```

以下の様に変更することで実現できます。

```
-----------------------------------------------
-- エラーが発生しているエンドポイントを再接続する
-----------------------------------------------
for key,val in ipairs(id) do
  -- 現在ソケット未接続またはエラー発生中のエンドポイントを対象にする
  if (not enable[key]) or (enable[key] and (not ready[key])) then
```

上記の変更を加えると mqtt_close() 関数がソケット未接続の場合にエラー(既にソケットが閉じているため) を検出しますが問題ありません。

●備考

このスクリプトに変更を加える場合には、スクリプトの実行は長くても数秒以内で必ず終了するようにしてください。同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が待たされる原因にもなります。

```
]]
-- log_msg("start..",file_id)
-----------------------------------------------
-- 最新のエンドポイントリストを取得
-----------------------------------------------
local stat, enable, ready, title, id, host, port, subscribe_topic_list, subscribe_qos_list,
user, password, will_topic, will_message, will_qos, will_retain, recv_buff_size,log = mqtt_all_list()
if not stat then error() end
-----------------------------------------------
-- PING メッセージをブローカに送信
-----------------------------------------------
for key,val in ipairs(id) do
  ---------------------------------------------------------------------------
  -- 現在ソケット接続中でエラーが発生していないエンドポイントのみを対象にする
  ---------------------------------------------------------------------------
  if enable[key] and ready[key] then
    ---------------------------------------
    -- PINGREQ メッセージをブローカに送信
    ---------------------------------------
```

```
    mqtt_ping(id[key])
  end
end
-------------------------------------------------------------
-- 先の PINGREQ 送信時にエラーを検出している場合があるので、
-- 再び最新のエンドポイントリストを取得
-------------------------------------------------------------
stat, enable, ready, title, id, host, port, subscribe_topic_list, subscribe_qos_list,
user, password, will_topic, will_message, will_qos, will_retain, recv_buff_size,log = mqtt_all_list()
if not stat then error() end
-----------------------------------------------
-- エラーが発生しているエンドポイントを再接続する
-----------------------------------------------
for key,val in ipairs(id) do
  ---------------------------------------------------------------------
  -- 現在ソケット接続中でエラーが発生しているエンドポイントを対象にする
  ---------------------------------------------------------------------
  if enable[key] and (not ready[key]) then
    log_msg("*WARNING* re-starting the endpoint ClientID = " .. id[key],file_id)
    --------------------------------
    -- エンドポイントのソケット削除
    --------------------------------
    mqtt_close(id[key])
    ------------------------------
    -- Broker にソケット接続開始
    ------------------------------
    if mqtt_open(id[key]) then
      ------------------------------------
      -- Broker に CONNECT リクエスト送信
      ------------------------------------
      local cstat,connack = mqtt_connect(id[key])
      if cstat then
        if (connack == 0) then
          if subscribe_topic_list[key] ~= "" then
            ------------------------
            -- デフォルトTopic購読
            ------------------------
            mqtt_subscribe(id[key])
          end
```

```
        else
          ------------------------------------------------------
          -- Brokerから CONNECT を拒否された時はソケット削除
          ------------------------------------------------------
          log_msg("connack is not 0, closing socket " .. id[key],file_id)
          mqtt_close(id[key])
        end
      end
    end
  end
end
```

13.12 MQTT_PUBLISH (MQTT-PUBLISHパケット受信)

- **イベント発生条件**

MQTTサービスで設定したエンドポイントから、MQTT PUBLISH メッセージを受信した時に発生します。

サーバー設定ファイルで MQTT サービスを有効にして、接続先 MQTT ブローカをエンドポイントに登録しておきます。エンドポイント登録時に "自動でSubscribe するトピック" "/Document/Class/MQTT/EndPointList/Item#n/AutoSubscribeTopicList" と "/Document/Class/MQTT/EndPointList/Item#n/AutoSubscribeQoSList" 項目を設定すると、サーバー起動時に自動的にPUBLISHメッセージを受信できる状態になります。

mqtt_subscribe() ライブラリ関数を使用すると、購読対象にする任意のトピックを後から追加したり、反対に mqtt_unsubscribe() ライブラリ関数を使用して、現在の購読対象から指定したトピックを外すことができます。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
ClientID	エンドポイントの ClientID	“abs9k:2222-eagle”
Title	エンドポイントに設定されたタイトル文字列。 タイトル文字列が設定されていない場合には、"" 空文字列が入ります。	“センサーデバイス#1”
MessageType	MQTT プロトコルで定義されたメッセージタイプが入ります。 PUBSLIH メッセージの場合には常に "3"が設定されます	"3"
MessageID	Brokerから送信するときに使用された MQTT メッセージID が入ります。(QoS = 1 または QoS = 2 の場合) 値は "1" から "65535" の整数値をとります。 QoS = 0 の場合には常に "0" が設定されます。	"1234"
Dup	MQTT 固定ヘッダ中の Dup フラグの値が設定されます。	"0"

	"0" または "1" の値をとります。	
Qos	MQTT 固定ヘッダ中の QoS フラグの値が設定されます。 "0", "1", "2" の何れかの値をとります。	"0"
Retain	MQTT 固定ヘッダ中の Retain フラグの値が設定されます。 "0" または "1" の値をとります。	"0"
PublishTopic	MQTT ブローカから受信した PUBLISH メッセージ中の Topic 文字列。	"センサー/ノード1"
PublishData	MQTT ブローカから受信した PUBLISH メッセージ中のペイロードデータ。 バイナリデータを16進数文字列に変換したものが格納されます ペイロードデータに格納されたデータが UTF-8 文字列の場合には文字列コードのバイト列が格納されています。 イベントハンドラ中でこれらの文字列データをデコードする処理がデフォルトで記述されていますので、UTF-8 文字列を扱う場合にはデコード後の変数を利用することができます。	"010203414243"

- **PUBLISH_DATAで渡されたペイロードデータを解析して作成される文字列変数**

下記の文字列変数は、MQTT_PUBLISHイベントハンドラ中に記述されたデフォルトスクリプトで作成されるデータ構造です。これらの変数を利用するとペイロードデータの内容を、文字列形式や JSON フォーマットで簡単に扱うことができます。

文字列変数名	説明	値の例
PublishString	PublishData に格納されたペイロードデータ部分のサイズが 4096 Bytes以内の場合に、データバイト列を UTF-8 形式で文字列にデコードした結果を PublishString に格納します。変換対象のバイト列のサイズを変更したいときには該当するスクリプト部分を変更して下さい。 PublishData 例 "414243"	PublishString = "ABC"

- **備考**

MQTT エンドポイントから PUBLISH メッセージを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。処理時間が長い場合でかつ複数並行して処理できる場合には、script_fork_exec() ライブラリ関数をイベントハンドラ中からコールすることができます。この場合には処理を別スレッドで行うことで、イベントハンドラ自身は直ぐに終了させることが可能になります。

受信した PUBLISH メッセージは abs_agent に登録したエンドポイント毎に作成される内部のキューに一旦格納されます。このキューを利用することで、複数の PUBLISH メッセージを連続して受信したときにも、ブローカから受信した PUBLISH メッセージ順にイベントハンドラを順に実行します。このとき、**同一エンドポイントで発生した**

PUBLISH メッセージについては、必ず先行する MQTT_PUBLISH イベントハンドラスクリプトの処理が完了してから、次の MQTT_PUBLISH イベントハンドラスクリプトが実行されます。

後で受信した PUBLISH メッセージ中の QoS が 1 または 2 の時には、それらの PUBLISH メッセージに対する受信応答 MQTT メッセージ(PUBACK, PUBREC, PUBCOMP) は、現在処理中の先行するイベントハンドラ処理が完了した後に送信されます。

複数のエンドポイントを abs_agent に登録して、それらが同時に PUBLISH メッセージを受信した場合には MQTT_PUBLISH イベントハンドラは同時に実行されます。ただし、abs_agent で同時実行可能なスクリプト数を超えた場合は、他のスクリプト実行が終了するまで実行が待たされます。

⚠ MQTT_PUBLISH イベントハンドラ中から mgcp_command() ライブラリ関数コール時の制限

MQTT_PUBLISH イベントハンドラ中から同一エンドポイントを経由した mgcp_command() ライブラリ関数を実行するとタイムアウトが発生します。この場合には、mgcp_command() 実行が必要なイベント処理スクリプトファイルを別に作成して、MQTT_PUBLISH イベントハンドラ中では script_fork_exec() 関数を実行して、イベント処理スクリプトファイルを別スレッドで実行するようにしてください。

イベントハンドラ例

```
file_id = "MQTT_PUBLISH"
--[[
MQTT_PUBLISH スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値          値                                  値の例
--------------------------------------------------------------------------------
ClientID        エンドポイントの ClientID 文字列          "abs9k:2222-eagle"
Title           エンドポイントに設定されたタイトル文字列。
                タイトル文字列が設定されていない場合には、"" 空文字列
                が入ります                                "センサーデバイス#1"
MessageType     MQTT プロトコルで定義されたメッセージタイプが入ります。        "3"
                PUBSLIH メッセージの場合には常に "3"が設定されます
MessageID       Brokerから送信するときに使用された MQTT メッセージID が入ります。
                (QoS = 1 または QoS = 2 の場合) 値は "1" から"65535" の整数値をとります。
                QoS = 0 の場合には常に "0" が設定されます。          "1234"
Dup             MQTT 固定ヘッダ中の Dup フラグの値が設定されます。
                "0" または "1" の値をとります。                   "0"
QoS             MQTT 固定ヘッダ中の QoS フラグの値が設定されます。
                "0", "1", "2" の何れかの値をとります。            "0"
Retain          MQTT 固定ヘッダ中の Retain フラグの値が設定されます。
```

"0" または "1" の値をとります。 "0"

PublishTopic MQTT ブローカから受信した PUBLISH メッセージ中の Topic文字列。 "センサー/ノード1"

PublishData MQTT ブローカから受信した PUBLISH メッセージ中のペイロードデータ。
バイナリデータを16進数文字列に変換したものが格納されますペイロードデータに格納されたデータが UTF-8 文字列の場合には文字列コードのバイト列が格納されています。イベントハンドラ中でこれらの文字列データをデコードする処理がデフォルトで記述されていますので、UTF-8 文字列を扱う場合にはデコード後の変数を利用することができます。 "010203414243"

PublishDataで渡されたペイロードデータを解析して作成される文字列変数

PublishString PublishData に格納されたペイロードデータ部分のサイズが 4096 Bytes以内の場合に、データバイト列を UTF-8形式で 文字列にデコードした結果を PublishString に格納します。
変換対象のバイト列のサイズを変更したいときには該当するスクリプト部分を変更して下さい。
]]

```
--------------------------------------------------------------------------------
-- 受信したペイロードデータのサイズが 4096 bytes 以内の場合には
-- バイナリデータ列を UTF-8 文字列としてデコードしたものを PublishString 変数に格納する
--------------------------------------------------------------------------------
local PublishString = ""
local pub_len = string.len(g_params["PublishData"]) / 2
if pub_len < 4096 then
  PublishString = readUTF_hex(bit_tohex(pub_len,4) .. g_params["PublishData"])
end

log_msg(g_params["Title"] .. "[" .. g_params["ClientID"] .. "] msg:" .. g_params["MessageID"] .. " dup:" ..
g_params["Dup"] .. " retain:" .. g_params["Retain"] .. " qos:" .. g_params["QoS"] .. " topic:" ..
g_params["PublishTopic"] .. " " .. PublishString,file_id)
```

13.13 RASPI_CHANGE_DETECT (Raspberry Pi GPIO値が変化)

- **イベント発生条件**

このイベントは Raspberry Pi ハードウエア用にビルドされた abs_agent のみ有効です。
ライブラリ関数 raspi_change_detect() をコールして GPIO 値を監視している時に、該当する GPIO 値が変化した場合に発生します。

検出対象の GPIO ポート値は 約 10ms 間隔で値を常に取得して、データ値が連続して同一の値を示した時に新しいポート値として内部に保存しています。この保存したポート値が前回保存していたポート値と比べて、変化していた

場合に RASPI_CHANGE_DETECT イベントが発生します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
BitNumList	GPIO の監視対象ビット中で変化したGPIO ピン番号が入る 複数同時に変化した場合にはカンマ区切りのリストになる 0 から 53 までの整数が入る	18,22,23
BitValList	BitNumList に格納されている GPIO ピン変化後の現在値 カンマ区切りのリストで表される場合にはBitNumList のピン番号並びと値の並びが対応しています。 0 または 1 の値が入る	0,1,1

- **BitNumList,BitValList パラメータからイベントハンドラ中で作成される数値配列**

change_bit 数値配列は、RASPI_CHANGE_DETECT イベントハンドラ中に記述されたデフォルトスクリプトで作成されます。この数値配列を利用すると簡単に変化した GPIO 値を取得することができます。

配列名	説明
change_bit[]	数値配列。キー(配列のインデックス値)が GPIO ピン番号の整数で、配列の値はGPIO の現在値を示します。 0(GPIO Low) または 1(GPIO High) の整数値。たとえば上記の BitNumList, BitValList の値がそれぞれ、"18,22,23", "0,1,1" の場合にはchange_bit[] 配列には下記の値が格納されます。 change_bit[18] = 0 change_bit[22] = 1 change_bit[23] = 1 Lua では存在しないキーの配列エントリにアクセスすると、値は nil が返ります。上記の例では change_bit[0] = nil です。

- **備考**

イベントハンドラ例

```
file_id = "RASPI_CHANGE_DETECT"
log_msg("start..",file_id)


--------------------------
-- change_bit[] 配列作成
--------------------------
local arr_pos = csv_to_tbl(g_params['BitNumList'])
```

```
local arr_val = csv_to_tbl(g_params['BitValList'])
local change_bit = {}
for i,v in ipairs(arr_pos) do
        change_bit[tonumber(v)] = tonumber(arr_val[i])
end
-----------------------------------------------
-- change_bit[] 配列内容確認のためログ出力
-----------------------------------------------
for key,val in pairs(change_bit) do
        log_msg(string.format("change_bit[%d] = %d",key,val),file_id)
end


-------------------------------------------------
--  GPIO22 に接続されたスイッチを押すとサウンド再生
-------------------------------------------------
if change_bit[22] then
        if change_bit[22] == 0 then
                os.execute('/usr/bin/aplay /home/pi/beep1.wav &')
        end
end
```

13.14 WEBSOCKET_DATA (WebSocketデータフレーム受信)

- **イベント発生条件**

WebSocketサーバーに接続したクライアントから、テキストフレームまたはバイナリフレームを受信した時に発生します。

複数のフラグメントに分かれたフレームを受信した場合には、全てのフレームのペイロードをまとめてからイベントが発生します。このためユーザー側ではフラグメントを意識しないでクライアントが送信したデータを扱うことができます。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
Channel	クライアントが WebSocket 接続時に指定した URL パス名部分の Channel 文字列が格納される。 パス名 := "/<channel>/<SessionToken>"	"my_app1"
SessionToken	クライアントが WebSocket 接続時に指定した URL パス名部分の SessionToken文字列が格納される。 セッショントークン文字列は WebSocket 接続時に認証に使用した後は、WebSocket サーバー側では一切使用しない。このた	"ST04B20D83368F1C"

	め、イベント発生時にこのセッショントークンが abs_agent 側に存在するかは関知していない点に注意。 パス名 := "/<channel>/<SessionToken>"	
WebSocketID	WebSocket クライアント接続毎に付けられたユニークな文字列。 このパラメータ値を websocket_emit_text(), websocket_emit_binary() ライブラリ関数の第一パラメータに指定すると、イベント発生時にデータを送信してきたWebSocketクライアント接続(１つ)にのみデータを送信することができる。	"WS04B20D98A65F4F"
PayloadType	WebSocketフレームタイプを示す。 "text" または "binary" が格納される。	"text"
PayloadSize	受信したフレームデータのサイズ(バイト数)が入る	"128"
PayloadData	受信したフレームのバイナリデータを16進数文字列に変換したものが格納されます PayloadType が "text" の場合は、UTF-8 文字列コードのバイト列が格納されています。イベントハンドラ中でこれらの文字列データをデコードする処理がデフォルトで記述されていますので、UTF-8 文字列を扱う場合にはデコード後の変数を利用することができます。	"010203414243"

- **WEBSOCKET_DATAで渡された PayloadDataを解析して作成される文字列変数**

下記の文字列変数は、WEBSOCKET_DATAイベントハンドラ中に記述されたデフォルトスクリプトで作成される文字列変数です。この変数を利用するとペイロードデータの内容を、文字列形式や JSON フォーマットで簡単に扱うことができます。

文字列変数名	説明	値の例
PayloadString	PayloadType が "text" の場合に、データバイト列をUTF-8形式で文字列にデコードした結果をPayloadString に格納します。	PayloadString = "ABC"

- **備考**

WebSocket クライアントからデータフレームを受信する毎に、このイベントが発生してスクリプトを実行しますので、スクリプト実行はできるだけ短い実行時間で完了するようにしてください。処理時間が長い場合でかつ複数並行して処理できる場合には、script_fork_exec() ライブラリ関数をイベントハンドラ中からコールすることができます。この場合には処理を別スレッドで行うことで、イベントハンドラ自身は直ぐに終了させることが可能になります。

複数の WebSocket クライアントから同時にデータフレームを受信した場合には、このイベントハンドラは複数並行して実行されます。

データフレームを送信した WebSocket クライアントに対してデータを送り返すような処理を行う場合には、スクリプトパラメータ g_params["WebSocketID"] の値を Lua ライブラリ関数 websocket_emit_text() や websocket_emit_binary() の第一パラメータに指定します。

イベントハンドラ例

```
--------------------------------------------------------------------------------------------
-- 受信したPayloadType が "text" の場合には
-- バイナリデータ列を UTF-8 文字列としてデコードしたものを PayloadString 変数に格納する
--------------------------------------------------------------------------------------------
local PayloadString = ""
if g_params["PayloadType"] == 'text' then
  local pub_len = tonumber(g_params["PayloadSize"])
  PayloadString = readUTF_hex(bit_tohex(pub_len,4) .. g_params["PayloadData"])
end
if PayloadString ~= "" then
  log_msg("/" .. g_params["Channel"] .. "/" .. g_params["SessionToken"] .. " [" .. g_params["WebSocketID"] ..
   "] " .. PayloadString,g_script)
else
  log_msg("/" .. g_params["Channel"] .. "/" .. g_params["SessionToken"] .. " [" .. g_params["WebSocketID"] ..
   "] " .. g_params["PayloadData"],g_script)
end


-------------------------------------------------------------------------------
-- 送信してきたメッセージ内容をそのままクライアント側に送り返す(Echo)例
-------------------------------------------------------------------------------
if PayloadString ~= "" then
  if not websocket_emit_text(g_params["WebSocketID"],PayloadString) then error() end
end
```

13.15 TCPSERVER_DATA (TCPデータ・ModbusTCPリクエスト受信)

- **イベント発生条件**

汎用TCPサーバー機能を有効にしている時に、TCPクライアントからデータを受信した時に発生します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
TCPSocketID	接続毎に付けられたユニークな文字列	"CSK05D2091F65172D"
IPAddress	クライアントの IP アドレス	"192.168.100.18"
EventData	受信したフレームのバイナリデータを16進数文字列に変換した	"000000000006010300

	ものが格納されます	180001″

- **イベントハンドラからスレッドに返すリターンパラメータ(オプション)**

パラメータキー名	説明	パラメータ値の例
EventResult	クライアント側に送信するバイナリデータを16進数文字列に変換したものをリターンパラメータに設定します このパラメータを設定しなかった場合はクライアントへのデータ送信は行いません。	″000000000005010302 0005″

- **備考**

汎用TCPサーバーにクライアントが接続すると接続毎に新規のワーカースレッドが作成されます。ワーカースレッドが接続中のソケットからデータを受信する毎にこのイベントハンドラが実行されます。このイベントハンドラの終了を待ってから、ワーカースレッドはクライアントへのデータ送信を行います。その後、ソケットから再びデータ受信を試みます。クライアント側からリクエストパケットを送信した後リプライパケットを受信するまでのディレイを短くするために、このスクリプト実行はできるだけ短い実行時間で完了するようにスクリプトを作成してください。

クライアントから受信するデータフォーマット(バイト数)は abs_agent コンフィギュレーション中の NETSERVER 部分に記述した <固定長>+<可変長>フォーマットで計算されます。

TCPSERVER_DATA イベントハンドラ中でリターンパラメータ EventResult を設定すると、ワーカースレッドから接続中のクライアントにリプライパケットを送信します。EventResultを設定しなかった場合には送信しません。

イベントハンドラ終了後に、ワーカースレッドが接続中のクライアントから引き続きパケットを受信すると、このイベントハンドラが再び起動されます。クライアント側からソケットを切断した場合には、サーバー側のワーカースレッドも終了します。

イベントハンドラ中で error() 関数をコールしたり、ランタイムエラー等でスクリプトが強制終了すると、ワーカスレッドとクライアント間のソケットが切断された後ワーカースレッドも終了します。

複数の TCPクライアントからTCPサーバーに接続している場合には、複数のワーカースレッドが平行して動作します。これらのワーカースレッドが同時にデータを受信した場合は、このイベントハンドラも複数並行して実行されます。

イベントハンドラ例

```
--[[
ModbusTCP プロトコルサンプル実装
●リクエストパケットフォーマット
req_arr[1] : トランザクション識別子
req_arr[2] : トランザクション識別子
req_arr[3] : プロトコル識別子
```

```
req_arr[4] ：プロトコル識別子
req_arr[5] ：フィールド長（上位バイト）
req_arr[6] ：フィールド長（下位バイト）
req_arr[7] ：UnitID・スレーブアドレス
req_arr[8] ：ファンクションコード
req_arr[9] ：Data#0
..
..
req_arr[x] : Data#n
●リプライパケットフォーマット
rpl_arr[1] ：トランザクション識別子
rpl_arr[2] ：トランザクション識別子
rpl_arr[3] ：プロトコル識別子
rpl_arr[4] ：プロトコル識別子
rpl_arr[5] ：フィールド長（上位バイト）
rpl_arr[6] ：フィールド長（下位バイト）
rpl_arr[7] ：UnitID・スレーブアドレス(リクエストパケットのコピー)
rpl_arr[8] ：ファンクションコー(リクエストパケットのコピーとMSB にエラーステータスを含む)
rpl_arr[9] ：Data#0
..
..
rpl_arr[x] : Data#n
]]
local req_arr = hex_to_tbl(g_params["EventData"])
local rpl_arr = {}
if #req_arr < 6 then error() end
-- transaction ID
table.insert(rpl_arr,req_arr[1])
table.insert(rpl_arr,req_arr[2])
-- protocol ID
table.insert(rpl_arr,0x00)
table.insert(rpl_arr,0x00)
-- reply length
table.insert(rpl_arr,0x00)
table.insert(rpl_arr,0x05)
-- reply data
table.insert(rpl_arr,req_arr[7]) -- set UnitID
table.insert(rpl_arr,req_arr[8]) -- set function-code without errors
table.insert(rpl_arr,0x02)
```

```
table.insert(rpl_arr,0x00)
table.insert(rpl_arr,0x05)
script_result("EventResult",tbl_to_hex(rpl_arr))
```

13.16 UDPSERVER_DATA (UDPデータ受信)

- **イベント発生条件**

汎用UDPサーバー機能を有効にしている時に、UDPクライアントからデータを受信した時に発生します。

- **イベントハンドラに渡されるパラメータ**

パラメータキー名	説明	パラメータ値の例
IPAddress	クライアントの IP アドレス	"192.168.100.18"
EventData	受信したフレームのバイナリデータを16進数文字列に変換したものが格納されます	"000000000006010300180001"

- **イベントハンドラからスレッドに返すリターンパラメータ(オプション)**

パラメータキー名	説明	パラメータ値の例
EventResult	クライアント側に送信するバイナリデータを16進数文字列に変換したものをリターンパラメータに設定します このパラメータを設定しなかった場合はクライアントへのデータ送信は行いません。	"0000000000501030200005"

- **備考**

汎用UDPサーバーを有効にしている時に、クライアント側から送信したデータを受信するとこのイベントハンドラが実行されます。このイベントハンドラの終了を待ってから、クライアント(データ送信元の peer ノード)へのリプライデータ送信と新規データの受信を行います。このためイベントハンドラはできるだけ短い実行時間で完了するようにスクリプトを作成してください。

汎用UDPサーバーは 汎用TCPサーバーとは違って、クライアントからのコネクション処理やクライアント接続毎のワーカースレッド作成等は行いません。汎用UDPサーバーでは自身のプロセス内で、UDP データ受信とイベントハンドラ実行、その後必要に応じたリプライ送信処理を繰り返しています。

UDPSERVER_DATA イベントハンドラ中でリターンパラメータ EventResult を設定すると、クライアント(データ送信元の peer ノード)にリプライパケットを送信します。EventResultを設定しなかった場合には送信しません。

イベントハンドラ例

```
-- echo サーバーサンプル実装
script_result("EventResult",g_params["EventData"])
```

14　システム関連 API・変数

ライブラリ関数 API の各章では、ユーザースクリプトやイベントハンドラから使用可能な abs_agent で定義されたライブラリ関数とグローバル変数について説明しています。

グローバル変数は、タイトルに変数名と Lua型名を “:” で区切って表してあります。ライブラリ関数は、タイトルに関数名、”関数定義”と”パラメータとリターン値” の項目で関数の呼び出し形式とパラメータ、リターン値とその Lua型名を定義しています。

ライブラリ関数定義のパラメータ記述で *[,<modulename>]* の様にイタリック体の鍵括弧 *[ ]* で囲まれている場合には、そのパラメータは省略可能です。複数のパラメータを指定するときに、途中のパラメータのみを省略指定する場合には、パラメータ値に nil を指定してください。

パラメータ記述が <value1|value2> の様に、”<” と ”>” で囲まれて且つ、複数のパラメータ名が ”|” 文字で連結されている場合には、何れかのパラメータを１つ選択できます。

abs_agent で使用している Lua のバージョンは “5.2.3” です。
このマニュアルに記載した abs_agent 専用の API 以外にも、Lua5.2 で定義された標準ライブラリ関数を使用できます。ただし、標準ライブラリ使用時には下記の使用制限があります。

制限事項

abs_agent のスクリプトやイベントハンドラでは、Lua ライブラリの print, io.＊ などで、標準入出力や Linux OS のログイン状態を想定したライブラリ関数は使用できません。これは abs_agent がデーモンプロセスで動作しているためです。メッセージ出力を行いたい場合には、ログサーバーを設置して abs_agent で用意されたライブラリ関数 log_msg() を使用してください。

注意：ライブラリ関数の戻り値について

ライブラリ関数の実行結果の戻り値に実行結果ステータス(”stat” の名前で定義されます) が返る場合には、実行時にエラーを検出した場合に false が返り、実行が成功した場合には true が返ります。
ただし、**ユーザースクリプトやイベントハンドラ中に記述したライブラリ関数の呼び出しパラメータ指定に問題がある場合（必須パラメータが指定されていない等）には、ユーザースクリプトやイベントハンドラ自身の実行が中断されます。**この場合には実行結果ステータス(stat) に値は返されないで、スクリプト実行自体が中断されてログにエラーメッセージが記録されます。

ヒント：ライブラリ関数のパラメータ型について

ライブラリ関数で指定するパラメータの型が Number や String の場合に、それぞれの型に変換可能な別の型の値を渡すこともできます。この場合には実行時に内部で自動変換されます。ただし、変換エラーが発生すると意図しない値になりますので、なるべく型を合わせてライブラリ関数をコールするようにしてください。
また、ライブラリ関数のパラメータとして整数値のみを受け付けるときに、浮動少数点値を渡した場合には自動的に

丸め処理が行われます。

14.1 g_startup:String

- **機能概要**

  abs_agent が起動した日付時刻(YYYY/MM/DD HH:MM:SSの形式)

- **備考**

- **使用例**

```
log_msg(string.format("モジュール開始時刻 %s ホスト名 %s **",g_startup,g_hostname))
```

14.2 g_hostname:String

- **機能概要**

  abs_agent が動作しているコンピュータのホスト名

- **備考**

- **使用例**

```
log_msg(string.format("モジュール開始時刻 %s ホスト名 %s **",g_startup,g_hostname))
```

14.3 g_params:Table of String

- **機能概要**

  イベントハンドラ起動時やユーザースクリプト実行時に渡されたリクエストパラメータ。パラメータキー名と値がテーブル形式で格納される。

- **備考**

  スクリプト中から、スクリプトパラメータを指定してスクリプト実行を行う関数については script_exec(), script_exec2() を参照してください。また、スクリプト中からリターン値を呼び出し元に返す関数については script_result() を参照してください。

  スクリプトパラメータキーに "key1" その対応するパラメータ値に "val1" を指定した場合には、スクリプト中から g_params["key1"] にアクセスすると、その内容は "val1" になっています。パラメータには複数のキーと値のペアを指定することができ、それぞれ g_params["<キー名>"] でアクセスできます。

  g_params[] テーブルで指定されるパラメータキーと値は文字列値のみになります。数値なども文字列としてリクエストパラメータで指定します。数値から文字列に変換したり、反対に文字列から数値に変換するときは Lua の標準関数 tostring(),tonumber() を使用します。

  Lua のテーブルをスクリプトパラメータに使用したい場合には JSON フォーマットで一旦文字列にシリアライズしてください。詳しい使用方法は "HTTP クライアント・JSON API" の項の使用例をご覧ください。

- **使用例**

```
--------------------------------------
-- スクリプトに渡されたパラメーター覧
--------------------------------------
for key,val in pairs(g_params) do
   log_msg(string.format("g_params[%s] = %s",key,val))
end
```

14.4 g_taskid:String

- **機能概要**

  スクリプト実行時に設定されるユニークなタスクID 文字列。

- **備考**

  abs_agent 内部ではこの値をタスク管理用の UID として使用しています。ユーザーが作成するスクリプトやイベントハンドラ中で、スクリプト実行時にユニークとなる文字列が必要な場合にも利用できます。

  script_result() ライブラリ関数で taskid パラメータを指定する場合にはこの変数を指定します。

- **使用例**

```
if not script_result(g_taskid,"key1","val1") then error() end
```

14.5 g_sender:String

- **機能概要**

  スクリプト実行をリクエストしたコンピュータのホスト名

- **備考**

  イベントハンドラなどで、スクリプトが起動された場合は abs_agent が動作しているコンピュータのホスト名(g_hostnameと同一)が入ります。

  クライアントプログラムからスクリプトが起動された場合は、クライアントプログラムを実行したコンピュータのホスト名が入ります。

  Web API 経由でスクリプトが起動された場合は、常に “anonymous” が設定されます。

  script_rexec(), script_rexec2() でスクリプトが起動された場合は、script_rexec(), script_rexec2() 関数を実行した abs_agent のホスト名が入ります。

- **使用例**

```
log_msg(string.format("ホスト名 %s リクエスト元 %s **",g_hostname,g_sender))
```

14.6 g_json:Table

- **機能概要**

  json.decode(), g_json.encode() API を提供している JSON モジュールのインスタンス。

  通常は g_json.decode(), g_json.encode() ライブラリ関数をコールする形で使用しますので、g_json 変数を単独で使用することはありません。

- **備考**

この変数は preload ライブラリ (preload/011_JSON/z_globalload.lua) で下記の様に設定されています。

```
g_json = require('json')
```

JSON モジュールの詳しい API 仕様は、HTTP クライアント,JSON API章の g_json.decode(), g_json.encode() の項を参照してください。

g_json グローバル変数を使わずに、スクリプトやイベントハンドラ中で独自に require('json') 文を記述してライブラリにアクセスすることもできます。

- **使用例**

```
local msg = g_json.decode('{"z":100}')
log_msg("z = " .. tostring(msg.z),g_script)
```

14.7 g_script:String

- **機能概要**

スクリプト実行時に指定したスクリプト名が設定されます。

- **備考**

スクリプト名は abs_agent が動作しているスクリプトディレクトリ内に配置されたスクリプトファイルの相対パス名から、ファイルの拡張子を取り除いたものです。

- **使用例**

```
log_msg(string.format("起動スクリプト名 %s ",g_script))
```

デーモンタイプ(無限ループに入って終了しない)のスクリプト先頭部分で２重起動を防止する例

```
-- ２重起動防止用チェック
if not exclusive_check(g_script) then
    log_msg("*ERROR* exclusive_check() failed. script = " .. g_script,g_script)
    return
end
log_msg("start..   TaskID = " .. g_taskid,g_script)
```

14.8 g_dir:String

- **機能概要**

スクリプト実行時に指定したスクリプトファイルが格納されているディレクトリ名。

- **備考**

ディレクトリ名は、実行中のスクリプトファイルが格納されている OS 上のディレクトリ名が絶対パス名で設定されます。

- **使用例**

```
log_msg(string.format("起動ディレクトリ %s ",g_dir))
```

14.9 g_signal:Number

- **機能概要**

実行中のスクリプトに設定されたシグナルが保存されている変数。

スクリプト起動直後やシグナルを受信していない場合には、この変数の値は nil になります。

- **備考**

シグナルは 1 から 255 までの整数値です(1Byte値)。0 はシグナル無しの意味で、9 はタスク強制終了を意味するためこの変数にアサインされることはありません。

スクリプト中から実行中のスクリプトインスタンスに対してシグナルを設定する場合には script_signal() ライブラリ関数を使用します。またクライアントプログラム agent_task を使用して実行中のスクリプトにシグナルを設定することもできます。詳しくはスクリプト実行 API章中の "script_signal()" の項を参照してください。

g_signal は Lua スクリプトインスタンス中にダイナミックに設定されるグローバル変数です。実行中のスクリプトから g_signal 変数にアクセスすると最後に設定されたシグナル値を取得できます。シグナルをクリアしたい場合には単に g_signal に nil を代入して下さい。(下記使用例を参照)

- **使用例**

```
while true do
  wait_time(100)
  if g_signal then
    log_msg("signal received, value " .. tostring(g_signal),g_script)
    g_signal = nil
  end
  log_msg("task running..." .. g_taskid,g_script)
end
```

14.10 log_msg(), log_hexdump(), log_msg_strlist()

- **機能概要**

文字列メッセージやバイナリ配列の16進数ダンプをログサーバーに出力する。

- **関数定義**

log_msg(message [,modulename])

log_hexdump(data_arr [,modulename])

log_hexdump(hexstr [,modulename])

log_msg_strlist(message [,maxlist])

- **パラメータとリターン値**

message:String　　メッセージ本文(任意の文字列)

data_arr:Table [1..#max_item] of Number

　　バイナリデータが格納された数値データ配列

　　(配列の各項目は1バイトに収まる 0から255 (0x0..0xff)の整数)

hexstr:String　　16進数表記の文字列

modulename:String　　モジュール名(任意の文字列)

　　パラメータ省略時は自身のスクリプト名になる。(g_script を指定したのと同じ)

maxlist:Number　　文字列リストに保存する最大メッセージ数。省略時は 100 を使用する。

- **備考**

abs_agent で使用するイベントハンドラやユーザースクリプトでは、メッセージを出力するために標準出力ではなくこのログ出力関数を使用してください。

ログサーバーには UDPパケットを使用してメッセージ送信を行っています。数ミリ秒間隔で複数のログメッセージを連続して送信すると、ログサーバーで全てのメッセージパケットを取り込めない場合があります。この場合はメッセージ出力後に wait_time() 関数を使用して、数ms程度の wait を入れてください。

log_hexdump() では配列に格納されたバイナリデータを 16進数フォーマットでダンプ出力します。ダンプ可能なバイナリデータの最大バイト数は 2048 バイトです。
ログに表示されるインデックス値は、data_arr配列のインデックス値から 1 デクリメントした値です。複数行に渡るログ出力時には 数ms の wait が自動で挿入されます。

log_msg_strlist() はログサーバーにメッセージを送信すると同時に、abs_agent のグローバル共有文字列リストにも同様のメッセージを格納します。保存するチャンネル名は 'LOG' + <g_script> です。頻繁にメッセージを出力する場合には log_msg() に比べてパフォーマンスが低下しますので、稀な事象をログサーバーの運用なしで確認したい場合に適しています。

- **使用例**

```
log_msg("これはログメッセージです","TestModule")
local buff = {}
for x = 0,255,1 do
  table.insert(buff,x)
end
log_hexdump(buff,g_script)
```

(上記スクリプト実行時のログ。buff テーブルのインデックスは 1 から始まるが、ログダンプ時は 0 から始

めるように調整される）

```
2020/07/22 17:12:55 raspberryp TestModule          0 これはログメッセージです
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  length = 256 bytes  checksum = 0x80
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[000]-[00F] 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[010]-[01F] 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[020]-[02F] 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                    ! " # $ % & ' ( ) * + , - . /
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[030]-[03F] 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                  0 1 2 3 4 5 6 7 8 9 : ; < = > ?
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[040]-[04F] 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                  @ A B C D E F G H I J K L M N O
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[050]-[05F] 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                  P Q R S T U V W X Y Z [ ¥ ] ^ _
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[060]-[06F] 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                  ` a b c d e f g h i j k l m n o
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[070]-[07F] 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0                  p q r s t u v w x y z { | } ~
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[080]-[08F] 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[090]-[09F] 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0A0]-[0AF] A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0B0]-[0BF] B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0C0]-[0CF] C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0D0]-[0DF] D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0E0]-[0EF] E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0  data[0F0]-[0FF] F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
2020/07/22 17:12:55 raspberryp TEST/LOG_HEX_TEST   0
```

14.11 wait_time()

- **機能概要**

  指定された時間ウェイトする。

- **関数定義**

  wait_time(time)

- **パラメータとリターン値**

  time:Number　　ウェイトを行う時間 (ms 単位)

- **備考**

- **使用例**

```
for count = 5,0,-1 do
    wait_time(1000)
    log_msg(string.format("count = %d",count),"COUNTER")
end
```

14.12 millis_delta()

- **機能概要**

  前回コール時にタイマー更新した時刻からの経過時間をミリ秒で返す。

- **関数定義**

  delta = millis_delta(timer_no *[,update]*)

- **パラメータとリターン値**

  delta:Number　　前回この関数をコールしてタイマーを更新した時点からの経過時間を
  ミリ秒単位の整数値で返す
  timer_no パラメータで時間計測を行うタイマーを指定する

  timer_no:Number　　経過時間測定用のタイマー番号を指定する。
  0, 1, 2, 3 の４つのタイマーを指定可能

  update:Boolean　　タイマーを現在時刻に更新する。省略時は true が設定される。

- **備考**

  リターン値 delta は、"32bit integer" サイズよりも大きい場合に値が切り捨てられます。このため長い期間(約 20日間以上)の測定はできませんので注意してください。

- **使用例**

```
while true do
  millis_delta(0) -- タイマーリセット
  -- 何らかの処理 --
  local delta = millis_delta(0) -- 上記処理を 100ms 間隔にするためのディレイ計算
    if delta < 100 then wait_time(100 - delta) end
```

```
  end
end
```

14.13 event_set()

- **機能概要**

  指定したイベントオブジェクトをセット状態にする

- **関数定義**

  stat = event_set(event_no)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
event_no:Number	対象となるイベントオブジェクト番号を指定する 0 または 1 を指定可能

- **備考**

  このライブラリ関数はユーザースクリプトやイベントハンドラから利用可能な、abs_agent 内部で予めリザーブされたイベントオブジェクトを操作します。操作対象のイベントオブジェクトは番号で指定して、複数のイベントオブジェクトを同時に使い分けることもできます。

  event_set() ライブラリ関数は event_wait() ライブラリ関数で待ち状態になっているイベントをセット状態にします。このとき、event_no パラメータに指定する値は同じものを指定します。

- **使用例**

```
  If not event_set(0) then error() end
```

14.14 event_wait()

- **機能概要**

  指定したイベントオブジェクトがセット状態になるのを待つ

- **関数定義**

  stat = event_wait(event_no, timeout)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
event_no:Number	対象となるイベントオブジェクト番号を指定する 0 または 1 を指定可能
timeout:Number	タイムアウト値をミリセコンドで指定する(ms) -1 を指定するとタイムアウトしないで永久に待ち状態になる(INFINITE)

- **備考**

  このライブラリ関数はユーザースクリプトやイベントハンドラから利用可能な、abs_agent 内部で予めリザーブされたイベントオブジェクトを操作します。操作対象のイベントオブジェクトは番号で指定して、複数のイベントオブジェクトを同時に使い分けることもできます。

event_wait() ライブラリ関数で待ち状態になっているイベントをセット状態にする場合には event_set() を使用します。このとき、event_no パラメータに指定する値は同じものを指定します。

この関数ではイベントの待ち状態に入る前に必ずイベントをリセット状態にしますので、この関数を実行する以前に実行された event_set() は無視されます。

別々のスレッドで実行中のスクリプトで同じイベントを待つこともできます。この時には event_set() が実行されると、同じイベントで待ち状態になっていた全てのスレッドで event_wait() 関数を抜け出ます。

指定した timeout 時間内にイベントオブジェトがセット状態にならなかった場合には、リターン値には false が返ります。

- **使用例**

```
If not event_wait(0,100000) then
    -- wait timeout --
end
```

14.15 event_set2()

- **機能概要**

  指定したイベントフラグをセット状態にする。

- **関数定義**

  stat = event_set2(name)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  name:String　　対象となるイベントフラグの名前を指定する

- **備考**

  このライブラリ関数はユーザースクリプトやイベントハンドラから利用可能な、グローバル共有文字列リストで作成したイベントフラグを操作します。操作対象のイベントフラグには任意の文字列を指定できます。

  event_wait2(),event_wait_either2() ライブラリ関数で待ち状態になっているイベントフラグをセット状態にします。

  このライブラリ関数は内部でグローバル共有文字列リストを使用するため、パフォーマンスは event_set(), event_wait() ライブラリ関数に比べると遅くなります。高速なイベント操作の繰り返しが必要で且つ、イベント待ちフラグが１つで充分な場合には、このライブラリ関数の代わりに event_set(), event_wait() ライブラリ関数の使用を検討してください。

- **使用例**

```
file_id = "BATCH_JOB"
--[[
バッチジョブ実行試験
                              +---> BATCH_JOB2 -----+
                              |                     |
                              |                     V
  BATCH_JOB ---> BATCH_JOB1 ----+                    +--> BATCH_JOB4 ---> BATCH_JOB (END)
                              |                     ^
                              |                     |
                              +---> BATCH_JOB3 -----+
]]
log_msg("start..",file_id)
-- 全ジョブを別スレッドで起動する
-- 各々のジョブスクリプトの先頭で、先行するジョブの完了を待つためにイベント待ちを行う
-- また、各々のジョブスクリプトの終了部分で、そのジョブが完了した事をイベントをセットすることで
-- イベント待ちのジョブスクリプトに伝える
if not script_fork_exec("BATCH_JOB1","MasterTaskID",g_taskid) then error() end
if not script_fork_exec("BATCH_JOB2","MasterTaskID",g_taskid) then error() end
if not script_fork_exec("BATCH_JOB3","MasterTaskID",g_taskid) then error() end
if not script_fork_exec("BATCH_JOB4","MasterTaskID",g_taskid) then error() end

-- 全てのジョブが終了するのを待つ
if not event_wait2(g_taskid .. "JOB1",g_taskid .. "JOB2",g_taskid .. "JOB3",g_taskid ..
"JOB4",30000) then
        log_msg("event_wait2() failed",file_id)
        return
end
log_msg("all job completed",file_id)
```

```
file_id = "BATCH_JOB1"
log_msg("start..",file_id)
-- 時間がかかるダミータスク
wait_time(2000)
-- ジョブが終了した
if not event_set2(g_params["MasterTaskID"] .. "JOB1") then error() end
log_msg("completed",file_id)
```

```
file_id = "BATCH_JOB2"
-- 先行するジョブ(JOB1) の完了を待つ
if not event_wait2(g_params["MasterTaskID"] .. "JOB1",10000) then
        log_msg("event_wait2() failed",file_id)
        return
end
log_msg("start..",file_id)
-- 時間がかかるダミータスク
wait_time(2000)
-- ジョブが終了した
if not event_set2(g_params["MasterTaskID"] .. "JOB2") then error() end
log_msg("completed",file_id)
```

```
file_id = "BATCH_JOB3"
-- 先行するジョブ(JOB1) の完了を待つ
if not event_wait2(g_params["MasterTaskID"] .. "JOB1",10000) then
        log_msg("event_wait2() failed",file_id)
        return
end
log_msg("start..",file_id)
-- 時間がかかるダミータスク
wait_time(2500)

-- ジョブが終了した
if not event_set2(g_params["MasterTaskID"] .. "JOB3") then error() end
log_msg("completed",file_id)
```

```
file_id = "BATCH_JOB4"
-- 先行するジョブ(JOB1と JOB2) の完了を待つ
if not event_wait2(g_params["MasterTaskID"] .. "JOB2",g_params["MasterTaskID"] .. "JOB3",10000)
then
        log_msg("event_wait2() failed",file_id)
        return
end
log_msg("start..",file_id)
-- 時間がかかるダミータスク
wait_time(1000)
-- ジョブが終了した
```

```
if not event_set2(g_params["MasterTaskID"] .. "JOB4") then error() end
log_msg("completed",file_id)
```

14.16 event_wait2(), event_wait_either2()

- **機能概要**

  指定した複数のイベントフラグがセット状態になるのを待つ

- **関数定義**

  stat = event_wait2(name1*[*,name2 .. name<n>*][*,timeout*]*)

  stat = event_wait_either2(name1*[*,name2.. name<n>*][*,timeout*]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  name<n>:String　　対象となるイベントフラグの名前を指定する

  複数のイベントフラグをパラメータに指定した場合に、event_wait2() は全てのフラグがセット状態になるまで待ち状態になる。event_wait_either2() は、何れかのフラグがセット状態になるまで待ち状態になる。

  timeout:Number　　タイムアウト値をミリセコンドで指定する(ms)

  -1 を指定するとタイムアウトしないで永久に待ち状態になる(INFINITE)

  このパラメータを省略した場合には -1 を指定したのと同様になる

- **備考**

  このライブラリ関数はユーザースクリプトやイベントハンドラから利用可能な、グローバル共有文字列リストで作成したイベントフラグを操作します。操作対象のイベントフラグには任意の文字列を指定できます。

  このライブラリ関数はイベントフラグを指定してセット状態になるのを待ちます。複数のイベントフラグを指定することもできます。複数のイベントフラグを指定する場合に、全てのフラグがセット状態になるまで待つ場合には event_wait2() を使用します。これに対して、何れかのフラグがセット状態になるまで待つ場合には、event_wait_either2() を使用してください。

  event_wait2(), event_wait_either2() ライブラリ関数で待ち状態になっているイベントフラグをセット状態にする場合には、event_set2()を使用してください。

  指定した timeout 時間内にイベントフラグがセット状態にならなかった場合には、リターン値には falseが返ります。

  この関数ではイベントの待ち状態に入る前に必ずイベントをリセット状態にしますので、この関数を実行する以前に実行された event_set2() は無視されます。

  このライブラリ関数は abs_agent 上のイベントハンドラやユーザースクリプト間でイベントの待ち合わせ機能を使用できます。

別々のスレッドで実行中のスクリプトで同じイベントを待つこともできます。この時には event_set2() が実行されると、同じイベントで待ち状態になっていた全てのスレッドで event_wait2() 関数を抜け出ます。

このライブラリ関数は内部でグローバル共有文字列リストを使用するため、パフォーマンスは event_set(), event_wait() ライブラリ関数に比べると遅くなります。高速なイベント操作の繰り返しが必要で且つ、イベント待ちフラグが１つで充分な場合には、このライブラリ関数の代わりに event_set(), event_wait() ライブラリ関数の使用を検討してください。

- **使用例**

```
If not event_wait2("JOB1","JOB2",100000) then
    -- wait timeout --
end
```

event_set2() 関数の使用例も参照してください。

14.17 critical_section_enter2()

- **機能概要**

  グローバル共有変数を使用した排他制御区間に入る

- **関数定義**

  stat = critical_section_enter2(key_name [,try_count])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
key_name:String	排他制御に利用する任意のグローバル共有変数名を指定する
try_count:Number	関数内部で排他制御権を取得するためのリトライ回数を指定する。リトライ間隔は約 1[ms] になります。例えば、1000 を指定すると 1秒程度、排他制御の取得を待ちます。 パラメータを省略すると制御権を獲得するまで永久に待ち状態になる(INFINITE)

- **備考**

  指定した try_count 数以内で排他制御権が確保できなかった場合には、リターン値に falseが返ります。この場合にはユーザースクリプトはエラー処理を適切に行って、排他制御区間に入ってはいけません。

  critical_section_enter2()の stat 値が true の場合は、排他制御権を確保した状態になっています。 排他制御が必要な処理を行った後に、critical_section_leave2() 関数をコールして排他制御権を開放してください。このとき key_name には critical_section_enter2()コール時に指定した同じ key_name を指定します。

  critical_section_enter2() コールのリターン値(stat)が false の場合には(リトライ回数エラー等)、critical_section_leave2() をコールする必要はありません。

- **使用例**

```
--------------------------------
-- クリティカルセクション開始
--------------------------------
if not critical_section_enter2("MyKey",5000) then error() end


--------------------------------
-- 排他が必要なタスク
--------------------------------
for i = 1,10,1 do
    wait_time(100)
    log_msg(g_params["Msg"],file_id)
end


--------------------------------
-- クリティカルセクション終了
--------------------------------
if not critical_section_leave2("MyKey") then error() end
```

14.18 critical_section_leave2()

- **機能概要**

  グローバル共有変数を使用した排他制御区間から抜ける

- **関数定義**

  stat = critical_section_leave2(name)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
name:String	排他制御に利用する任意のグローバル共有変数名を指定する critical_section_enter2()コール時に使用したのと同じ key_name を指定する。

- **備考**

  critical_section_enter2() を使用して、ユーザースクリプトが排他制御区間の処理を行った後、必ず critical_section_leave2() をコールして、他の待ち状態のスレッドが排他制御区間に入れる様にしてください。

14.19 exclusive_check()

- **機能概要**

  コールしたスクリプトが別タスク・スレッドで既に起動されていないことを確認する

- **関数定義**

  stat = exclusive_check(script_name)

- **パラメータとリターン値**

  stat:Boolean　　コールしたスクリプトのスレッドがサーバー上で唯一の場合には true, 別スレッドで同じスクリプトが実行中の場合には　false が返る。

  script_name:String　　自身のスクリプト名。"g_script" を指定する

- **備考**

  スクリプトを２重起動したくない場合に、先頭にこの関数を使用すると簡単に判定することができます。

  この関数の script_name パラメータにはグローバル変数 "g_script" を指定します。この関数の実行後、関数のリターン値が false の場合にはスクリプトを抜ける(return 実行) 様にすると、同一スクリプトが２重起動しないようになります。

  この関数は無限ループになるようなスクリプトを別スレッドで実行するときにも使用できます(下記の使用例を参照してください)。ライブラリ関数内部では、パラメータで渡されたスクリプト名を元に、内部でフラグを管理して排他的な実行ができるかどうかを確認しています。

  無限ループになるようなスクリプトを "script_kill()" ライブラリ関数や "agent_task -k <task_id>" クライアントプログラムでタスク・スレッドを強制終了させる場合でも、自動的に内部フラグがクリーンアップされて正しく動作するようになっています。

- **使用例**

```
file_id = "EXCLUSIVE_TASK"
log_msg("start..",file_id)

-- ２重起動防止用チェック
if not exclusive_check(g_script) then
        log_msg("*ERROR* exclusive_check() failed. script = " .. g_script,file_id)
        return
end

-- 以降無限ループに入る
while true do
        wait_time(100)
        log_msg("task running...",file_id)
end
```

14.20 create_session()

- **機能概要**

  abs_agent に新規セッションを作成する。ログイン認証を伴わないので作成したセッションにはユーザー情報が含まれない。

- **関数定義**

  stat, new_session_token = create_session(session_token *[,indefinite]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  new_session_token:String　　新規に作成されたセッショントークン文字列

  session_token:String　　新規に作成するセッショントークン文字列を指定する。
  空文字を指定した場合は abs_agent 側でユニークなセッショントークン文字列が生成される

  indefinite:Boolean　　セッションの自動削除対象から外す場合に true を指定する。
  false を指定した場合やパラメータ省略時には、最後にセッションが使用されてから 10 分経過すると自動削除される。

- **備考**

  ログイン認証を省略して、常に Web API で指定可能なセッションを作成したい場合に使用します。

- **使用例**

```
stat,token = create_session("PUBLIC_SESSION",true)
if not stat then error() end
```

14.21 delete_session()

- **機能概要**

  abs_agent のセッションを削除する。

- **関数定義**

  stat = delete_session(session_token)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  session_token:String　　削除対象のセッショントークン文字列を指定する。

- **備考**

  create_session() または Web API(/command/json/session_login) で作成されたセッションを削除する。

  Web API でログイン認証したセッションを削除する場合には、Web API の /command/json/session_logout も使用できます。

- **使用例**

```
stat = delete_session(“PUBLIC_SESSION”)
if not stat then error() end
```

14.22 renew_session()

- **機能概要**

abs_agent の既存のセッショントークン文字列を変更する。

- **関数定義**

stat, new_session_token = renew_session(current_session_token [,busy_timer])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

new_session_token:String　　新規に作成されたセッショントークン文字列

current_session_token:String　　既存のセッショントークン文字列を指定する。

busy_timer:Number　　セッションが最後に更新されてから busy_timer で指定された秒以上経過していない場合にはエラーにする。省略時は更新時刻のチェックをしない。

- **備考**

セキュリティを向上させるために、セッショントークン自身を定期的に更新したい場合に使用します。

セッションのユーザーアカウント情報は全て新規に作成したセッションに引き継がれます。

新規に作成されるセッショントークン文字列を明示的に指定することはできません。

- **使用例**

```
stat,current_token = renew_session(current_token,60)
if not stat then error() end
```

14.23 service_module_status()

- **機能概要**

abs_agent のサービスモジュールが ONLINE 状態かどうかを調べる。同時に、abs_agent のバージョン番号やコンフィギュレーション情報も取得できる。

- **関数定義**

stat, module_status, program_info = service_module_status()

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

module_status: Table of Boolean　　abs_agentの各サービスクラスのステータスが ONLINE の場合に、true, OFFLINE の場合に false がセットされる。

program_info: Table of String

abs_agentプログラムのバージョン番号や現在のコンフィギュレーション情報がセットされる。

- **備考**

  module_status のキー名は、abs_agent のサービスモジュール名（下記）になります。各々のキー名に対して現在 ONLINE の場合には true がセットされて OFFLINE の場合には false がセットされます。

  program_info テーブルには下記のキー名でコンフィギュレーション情報が文字列で格納されています。

- **module_status テーブルで取得できる abs_agent サービスモジュール一覧**

サービスモジュール名 キー名文字列	説明
CONVERT	内部で使用する変換テーブル機能、RGBグラフィック描画機能
CONFIG	内部設定値の管理機能
MASTERS	マスターテーブルの管理機能
SESSION	グローバル共有データ機能
WEBPROXY	Webサーバー、Web API 機能
SCRIPT	Lua スクリプト実行環境と、Lua APIライブラリ機能
SERIAL	シリアルデバイス管理機能
MQTT	MQTT ブローカとの接続管理機能
FASTDB	時系列集計用インメモリデータベース
LAST	ダミーモジュールでサービス起動確認用

- **program_infoテーブルで取得できる abs_agent コンフィギュレーション一覧**

キー名	説明
Version	abs_agent プログラムのバージョン番号
ConfigFile	abs_agent プログラムが使用しているコンフィギュレーションファイル名
ProgramFolder	abs_agent プログラムをインストールしたディレクトリ名
ScriptFolder	スクリプトファイルを格納しているトップディレクトリ名
MasterFile	マスターファイル名
WebRootFolder	Webサーバー機能で公開しているトップディレクトリ名
HardwareType	abs_agent プログラムが動作しているコンピュータのタイプ名
LicensedUser	ライセンス情報

- **制限事項**

- **使用例**

```
------------------------------------------------------------
-- コンフィギュレーションとモジュールステータスをログに出力
------------------------------------------------------------
local stat,module_stat,program_info = service_module_status();
if not stat then error() end;
for key,val in pairs(program_info) do
        log_msg(string.format("program_info[%s] = %s",key,val),g_script)
        wait_time(5)
end
for key,val in pairs(module_stat) do
        log_msg(string.format("module_stat[%s] = %s",key,tostring(val)),g_script)
        wait_time(5)
end
```

14.24 stat_check()

- **機能概要**

  第一パラメータ(p1) の値が偽(false または nil)の場合にエラーを発生させ、それ以外の場合は渡されたパラメータから第一パラメータを取り除いて返す。

- **関数定義**

  p2, p3,.., p#n = stat_check(p1, p2, .., p#n)

- **パラメータとリターン値**

  p1:boolean　　エラーチェック用のステータス

  p2 .. p#n:<任意の型>　　任意のパラメータ

- **備考**

  abs_agent で提供するライブラリ関数等で(1番目の)リターン値に実行時ステータス (stat) を返す場合に、この stat_check()ライブラリ関数でラップすることでエラーチェックとリターン値の取得を簡潔に記述できます。

  エラー発生は Lua の組み込みライブラリ関数 error() を実行しています。その結果、実行中のスクリプトは強制終了されログにメッセージが記録されます。

- **使用例**

```
local now = os.date "*t";
local wday = stat_check(day_of_week(now["year"],now["month"],now["day"]))
```

15　文字列・配列操作API

スクリプト内で、文字列の操作や配列変換を行うライブラリ関数を提供しています。abs_agent で提供している API のパラメータやイベントデータを処理するときに必要となる機能を提供しています。

Lua の標準文字列ライブラリ(string) で提供されている機能も使用することができます。

15.1　list_to_csv()

- **機能概要**

  文字列リストを CSV 形式の文字列にする。

- **関数定義**

  csv_str = list_to_csv(str#1, ...,str#n)

- **パラメータとリターン値**

  csv_str:String　　変換されたCSV 形式の文字列

  str#n:String　　任意の文字列

- **備考**

  文字列にスペース，カンマ，または引用符がある場合には，文字列は二重引用符で囲まれ，二重引用符がある場合には二重引用符が連続して付けられます。

- **使用例**

```
local csv = list_to_csv("aa¥"¥"","bb''","cc##","¥"d¥"d¥"","  ee ","ff")
```

```
local key = list_to_csv("aa¥"¥"","bb''","cc##")
local val = list_to_csv("これは¥"¥"","てすと''","です##")
stat = script_exec("SAMPLE",key,val)
if not stat then error() end
```

15.2　csv_to_tbl()

- **機能概要**

  CSV 形式文字列をカラムごとに分解して、文字列配列にする。

- **関数定義**

  str_arr = csv_to_tbl(csv_str)

- **パラメータとリターン値**

  str_arr:Table [1..#max_item] of String

  　　CSVで構成されていた各カラムの文字列を配列に格納したもの

  csv_str:String　　CSV 形式の文字列

- **備考**

  カンマ区切りで表現された複数カラムから構成される文字列データを、各カラム毎に分けた文字列配列に変換

します。

文字列はカンマで区切られ，任意で二重引用符で囲まれます。文字列に二重引用符がある場合，元の文字列と文字列を囲む引用符を区別するために続けて引用符を付けます。二重引用符で囲まれていないカンマは区切り記号です。2 つ並んだカンマは空の文字列を意味しますが，区切り記号の前後にあるスペースは無視されます。

- **使用例**

```
local csv = list_to_csv("aa¥"¥"","bb''","cc##","¥"d¥"d¥"","  ee ","ff")
local tbl = csv_to_tbl(csv)
log_msg(string.format("csv_text = %s",csv))
for akey,aval in ipairs(tbl) do
    log_msg(string.format("csv_column[%d] = %s",akey,aval))
end
```

15.3 hex_to_tbl()

- **機能概要**

16進数表記の文字列を、左から2文字毎に取得して数値配列で返す。

- **関数定義**

num_arr = hex_to_tbl(hexstr)

- **パラメータとリターン値**

num_arr:Table [1..#max_item] of Number
　　変換後の数値データ配列
　　(配列の各項目は1バイトに収まる 0から255 (0x0..0xff)の整数)

hexstr:String　　16進数表記の文字列

- **備考**

16進数文字列は、左から２文字ごとに区切って配列にロードします。

- **使用例**

```
data = "010203FDFEFF"
tbl = hex_to_tbl(data)
for key,val in ipairs(tbl) do
   log_msg(string.format("tbl[%d] = %d %2.2X",key,val,val))
   wait_time(10)
end
```

15.4 new_tbl()

- **機能概要**

  指定したサイズの数値配列を作成する。

- **関数定義**

  num_arr = new_tbl(size *[,init]*)

- **パラメータとリターン値**

  size:Number　　新規に作成する配列のサイズ。

  1以上の整数を指定する。作成した配列のインデックスは 1 から size までになる。

  num_arr:Table [1..#max_item] of Number

  0 または init で指定した値に初期化された数値が格納されている。

  init:Number　　配列の項目に初期設定される値。整数または浮動少数値を指定する。

  省略時は 0 になる。

- **備考**

  初期値をセットした数値配列を作成する。

- **使用例**

```
arr = new_tbl(100,0xff)
for key,val in ipairs(arr) do
   log_msg(string.format("arr[%d] = %d %2.2X",key,val,val))
   wait_time(10)
end
```

15.5 list_to_hex(), tbl_to_hex()

- **機能概要**

  数値リストまたは配列を16進数表記の文字列に変換する。

- **関数定義**

  hexstr = list_to_hex(num#1, ... ,num#n)

  hexstr = tbl_to_hex(num_arr *[,start [,end]]*)

- **パラメータとリターン値**

  num#n:Number　　#n番目の数値データ

  1バイトに収まる 0から255 (0x0..0xff)の整数のみが有効

  num_arr:Table [1..#max_item] of Number　　変換対象の数値データ配列

  start:Number　　num_arr の変換対象の開始インデックス番号

  省略時は 1 が指定される

  end:Number　　num_arr の変換対象の終了インデックス番号

  省略時は #num_arr が指定される

  hexstr:String　　16進数表記の文字列

- **備考**

  リスト中の数値が、マイナス値や 255 よりも大きい場合はエラーになります。

  start, end パラメータで指定する数値は 1 から #num_arr 間の整数のみ有効です。

- **使用例**

```
data = "010203FDFEFF"
tbl = hex_to_tbl(data)
hexdata = list_to_hex(unpack(tbl))
if data == hexdata then
   log_msg("success !!")
end
```

15.6 str_to_tbl()

- **機能概要**

  文字列を左から1バイト毎に数値に変換して配列で返す。

- **関数定義**

  num_arr = str_to_tbl(str)

- **パラメータとリターン値**

  num_arr:Table [1..#max_item] of Number
  　　文字列をバイト毎に分解して得られた数値データ配列
  　　配列の項目は1バイトに収まる 0から255 (0x0..0xff)の整数

  str:String　　任意の文字列

- **備考**

  文字列に日本語などのマルチバイト文字を指定した場合に、この関数で得られる数値データ列は単なるバイト列になる点に注意してください。マルチバイト文字コードを1文字毎のコードとして取得する場合には str_to_tbl2() 関数を使用してください。

- **使用例**

```
data = "test message!!"
tbl = str_to_tbl(data)
for key,val in ipairs(tbl) do
   log_msg(string.format("tbl[%d] = %d %2.2X",key,val,val))
   wait_time(10)
end
```

15.7 str_to_tbl2()

- **機能概要**

文字列を左から1文字毎に数値に変換して配列で返す。マルチバイト文字に対応している。

変換エラー検出のため、実行結果を示す stat が第一リターンパラメータに返る点に注意。

- **関数定義**

stat, num_arr = str_to_tbl2(str *[,code [,sjis_char]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
num_arr:Table [1..#max_item] of Number	文字コードを数値に変換したデータ。テーブルには１文字毎に文字コードデータが格納される。 文字コードは code パラメータで指定されたエンコード形式に変換される。 文字がマルチバイトコードの場合にはコード並び順のビッグエンディアン形式として１つの数値に変換される。例えば UTF-8 文字コードで 0xE5, 0x83, 0xAD の場合には、0xE583AD の数値がテーブル項目に格納される。
str:String	任意の文字列
code:Number	文字コードに変換するときのエンコード形式を指定する。 0: UTF-8（パラメータ省略時デフォルト） 1: Unicode 2: Shift-JIS
sjis_char:Number	code に 2 (Shift-JIS) 指定時に、変換対象となるコードが見つからなかった場合は、このパラメータで指定した文字コードに強制的に置き換える。

- **備考**

日本語などのマルチバイト文字を１文字ごとに分解して文字コードを表す数値配列に変換します。

code パラメータで指定したエンコード形式に変換するときに、コードが見つからなかった場合などはエラーが発生して stat には false が返ります。この時 num_arr には nil が設定されます。code に 2 (Shift-JIS) を指定して且つ、sjis_char パラメータも指定していた場合には、エラーは発生しないで指定した文字に置き換えられます。

- **使用例(1)**

```
local data = "ABCこれは試験デス"
local arr = {}
local stat
stat,arr = str_to_tbl2(data)
for key,val in ipairs(arr) do
  log_msg(string.format("utf-8[%d] = %X",key,val),g_script)
  wait_time(10)
```

```
end
stat,arr = str_to_tbl2(data,1)
for key,val in ipairs(arr) do
  log_msg(string.format("unicode[%d] = %X",key,val),g_script)
  wait_time(10)
end
stat,arr = str_to_tbl2(data,2)
for key,val in ipairs(arr) do
  log_msg(string.format("sjis[%d] = %X",key,val),g_script)
  wait_time(10)
end
```

上記スクリプトの実行結果(ログ)

```
utf-8[1] = 41
utf-8[2] = 42
utf-8[3] = 43
utf-8[4] = E38193
utf-8[5] = E3828C
utf-8[6] = E381AF
utf-8[7] = E8A9A6
utf-8[8] = E9A893
utf-8[9] = E38387
utf-8[10] = E382B9
unicode[1] = 41
unicode[2] = 42
unicode[3] = 43
unicode[4] = 3053
unicode[5] = 308C
unicode[6] = 306F
unicode[7] = 8A66
unicode[8] = 9A13
unicode[9] = 30C7
unicode[10] = 30B9
sjis[1] = 41
sjis[2] = 42
sjis[3] = 43
sjis[4] = 82B1
sjis[5] = 82EA
sjis[6] = 82CD
```

```
sjis[7] = 8E8E
sjis[8] = 8CB1
sjis[9] = 8366
sjis[10] = 8358
```

- **使用例(2) SJIS に無い文字を指定する場合 ('?' で代替した例)**

```
local data = "名前はCéline"
local arr = {}
arr = stat_check(str_to_tbl2(data))
for key,val in ipairs(arr) do
  log_msg(string.format("utf-8[%d] = %X",key,val),g_script)
  wait_time(10)
end
arr = stat_check(str_to_tbl2(data,1))
for key,val in ipairs(arr) do
  log_msg(string.format("unicode[%d] = %X",key,val),g_script)
  wait_time(10)
end
arr = stat_check(str_to_tbl2(data,2,0x3f))
for key,val in ipairs(arr) do
  log_msg(string.format("sjis[%d] = %X",key,val),g_script)
  wait_time(10)
end
```

上記スクリプトの実行結果(ログ)

```
utf-8[1] = E5908D
utf-8[2] = E5898D
utf-8[3] = E381AF
utf-8[4] = 43
utf-8[5] = C3A9
utf-8[6] = 6C
utf-8[7] = 69
utf-8[8] = 6E
utf-8[9] = 65
unicode[1] = 540D
unicode[2] = 524D
unicode[3] = 306F
unicode[4] = 43
unicode[5] = E9
```

```
unicode[6] = 6C
unicode[7] = 69
unicode[8] = 6E
unicode[9] = 65
sjis[1] = 96BC
sjis[2] = 914F
sjis[3] = 82CD
sjis[4] = 43
sjis[5] = 3F
sjis[6] = 6C
sjis[7] = 69
sjis[8] = 6E
sjis[9] = 65
```

15.8 list_to_str(), tbl_to_str()

- **機能概要**

  数値データリストまたは配列を連結した文字列に変換する。

- **関数定義**

  str = list_to_str(num#1, ... ,num#n)

  str = tbl_to_str(num_arr)

- **パラメータとリターン値**

num#n:Number	#n番目の数値データ 1バイトに収まる 0から255 (0x0..0xff)の整数のみが有効
num_arr:Table [1..#max_item] of Number	変換対象の数値データ配列
str:String	変換後の文字列

- **備考**

  リスト中の数値が、マイナス値や 255 よりも大きい場合はエラーになります。

  数値をそのまま連結して文字列に変換します。そのため入力データによって、制御コード等が文字列中に入る場合があります。

- **使用例**

```
data = "test message!!"
tbl = str_to_tbl(data)
strdata = tbl_to_str(tbl)
if data == strdata then
   log_msg("success !!")
end
```

15.9 load_file_tbl()

- **機能概要**

  ローカルファイルの内容をバイナリモードで読み込んで、1バイト毎に数値に変換して配列で返す。

- **関数定義**

  stat, num_arr = load_file_tbl(filename)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  num_arr:Table [1..#max_item] of Number

  　　ファイル内容をバイト毎に分解して得られた数値データ配列

  　　配列の項目は1バイトに収まる 0から255 (0x0..0xff)の整数

  filename:String　　ファイル名。絶対パス名で指定します。

- **備考**

  ローカルファイルの内容をテーブルにロードします。ファイルはバイナリモードで読み込みます。

  パラメータで指定したローカルファイルが存在しない場合にはエラーになります。

- **使用例**

```
local tbl = stat_check(load_file_tbl(/home/pi/mydata))
```

15.10 writeUTF_hex()

- **機能概要**

  文字列を左から1バイト毎に数値に変換して16進数表記の文字列に変換する。データの先頭には元の文字列バイト数を表すための２バイト分の長さデータが追加される

- **関数定義**

  hexstr = writeUTF_hex(str)

- **パラメータとリターン値**

  hexstr:String　　16進数表記の文字列。

  　　先頭２バイトには BigEndian 形式で２バイト分の文字列バイト数が入る

  str:String　　任意の文字列

- **備考**

  文字列データ値は UTF-8 コードで数値データ変換されます。

  ここで変換した hexstr データを元の文字列に戻す場合には readUTF_hex() ライブラリ関数を使用してください。また、この関数で扱っているエンコード形式は Java の writeUTF(), readUTF() と同じものです。

- **使用例**

```
hexstr = writeUTF_hex("これは試験です")
```

```
str = readUTF_hex(hexstr)
log_msg(str,file_id)
```

15.11 readUTF_hex()

- **機能概要**

16進数表記の数値データリストを連結した文字列に変換する。データの先頭には元の文字列バイト数を表すための２バイト分の長さデータが入っていることを想定しています。

- **関数定義**

str = readUTF_hex(hexstr)

- **パラメータとリターン値**

hexstr:String	16進数表記の文字列。 先頭２バイトには BigEndian 形式で２バイト分の文字列バイト数が入る
str:String	文字列

- **備考**

16進数の数値データ値は UTF-8 コードとして文字列データに変換されます。

この関数で扱う hexstr データは、writeUTF_hex() ライブラリ関数などを使用して作成できます。また、この関数で扱っているエンコード形式は Java の writeUTF(), readUTF() と同じものです。

- **使用例**

```
hexstr = writeUTF_hex("これは試験です")
str = readUTF_hex(hexstr)
log_msg(str,file_id)
```

15.12 ssv_to_tbl()

- **機能概要**

SSV(semicolon-separated values)形式の文字列をカラムごとに分解して、文字列配列にする。

- **関数定義**

str_arr = ssv_to_tbl(ssv_str)

- **パラメータとリターン値**

str_arr:Table [1..#max_item] of String	SSVで構成されていた各カラムの文字列
ssv_str:String	SSV 形式の文字列

- **備考**

";" セミコロン区切りで表現された複数カラムから構成される文字列データを、各カラム毎に分けた文字列配列に変換します。

文字列はセミコロンで区切られ，任意で二重引用符で囲まれます。文字列に二重引用符がある場合，元の文字列と文字列を囲む引用符を区別するために続けて引用符を付けます。二重引用符で囲まれていないセミコロンは区切り記号です。

- **使用例**

```
local tbl = ssv_to_tbl("113;00000000;084;001;1002a1e;3320;0009;0054;0009;0009;S")
for akey,aval in ipairs(tbl) do
    log_msg(string.format("ssv_column[%d] = %s",akey,aval))
end
```

15.13 key_val_to_tbl()

- **機能概要**

キー名と値を "="文字で繋いだペアを複数個並べて、":" で連結した文字列を連想配列に変換する。

- **関数定義**

tbl = key_val_to_tbl(key_val_str)

- **パラメータとリターン値**

tbl:Table of String　　キー名と値から構成される連想配列(テーブル)

key_val_str:String　　<key#1>=<val#1>:<key#2>=<val#2>: ... :<key#n>=<val#n> 形式の文字列

- **備考**

キーと値のペアは "=" 文字で関連づけられていて、各ペアの間は":" コロンで区切られます。

同一のキー名を文字列データ中に複数回指定することはできません。

- **使用例**

```
local tbl =
key_val_to_tbl("rc=80000000:lq=84:ct=0001:ed=81002A1E:id=1:ba=3320:a1=0009:a2=0009:p0=001:p1=000")
for key,val in pairs(tbl) do
    log_msg(string.format("key_val[%s] = %s",key,val))
end
```

15.14 hex_to_bit()

- **機能概要**

16進数表記の文字列全体を一つの数値として扱い、ビット位置の値を持った配列に変換する。

- **関数定義**

bit_arr = hex_to_bit(hexstr)

- **パラメータとリターン値**

bit_arr:Table [0..MSB] of Number

ビット位置毎の値(0 または 1) を示す配列。

配列のサイズは 16進数文字列で表した数値のビット幅に等しくなる。

hexstr:String　　16進数表記の文字列で表した１つの数値

- **備考**

16進数で表現する値のビット幅の制限はありません。次章で説明するビット演算ライブラリでは 31ビット長までの数値しか扱えませんが、このライブラリ関数では hexstr で指定可能な数値のビット幅に制限はありません。

bit_arr 配列のインデックスは 0 から始まる点に注意してください。Lua 標準ライブラリや abs_agent ライブラリ関数で配列を返す場合のインデックスは通常 1 から始まります。この hex_to_bit()関数の仕様は bit_arr 配列のインデックス値をビット位置の値に一致させるためです。ビット位置の値は 0 から始まり、MSB(最上位ビット位置)までの値になります。例えば bit_arr = hex_to_bit("F0") を実行すると　bit_arr配列の値は下記になります。このときのビット幅(配列のサイズ)は 8 で、インデックスは 0 から 7 になります。

```
bit_arr[0] = 0
bit_arr[1] = 0
bit_arr[2] = 0
bit_arr[3] = 0
bit_arr[4] = 1
bit_arr[5] = 1
bit_arr[6] = 1
bit_arr[7] = 1
```

- **使用例**

```
local data = "8000000001"
local bit_arr = hex_to_bit(data)
for key = 0,#bit_arr,1 do
  log_msg(string.format("bit_arr[%d] = %d ",key,bit_arr[key]))
  wait_time(10)
end
```

上記のスクリプト中に記述した #bit_arr は、配列の長さではなくインデックス値の最大値を示している点に注意してください。また、bit_arrテーブルのインデックスが 0から始まるため、Lua ライブラリ関数の **ipairs() は使用できません**。

15.15　num_to_hex()

- **機能概要**

Lua の数値を IEEE-754 倍精度(または単精度)浮動小数点の16進数文字列に変換する。

- **関数定義**

  hexstr = num_to_hex(num [,single])

- **パラメータとリターン値**

hexstr:String	16進数表記の文字列(“0x” は含まれない) 16文字(倍精度64bit)または8文字(単精度32bit)の16進数文字列
single:Boolean	単精度浮動小数点(32bit) フォーマットにする場合に ture を指定する パラメータ省略時や false 指定時は倍精度浮動小数点(64bit)になる。
num:Number	実数値

- **備考**

  num で指定する値は Lua の数値型(整数値や浮動少数点値)を指定します。

  取得する16進数文字列の数値フォーマットは、IEEE-754 倍精度浮動小数点形式または単精度浮動小数点(32bit)形式になります。

  この関数で変換された16進数文字列を元の数値に戻す場合には hex_to_num() ライブラリ関数を使用します。

- **使用例**

```
hexstr = num_to_hex(1.2345678)        -- "3FF3C0CA2A5B1D5D"
hexstr = num_to_hex(1.2345678E-10)    -- "3DE0F7BFD11A605F"
hexstr = num_to_hex(-1.2345678E-10)   -- "BDE0F7BFD11A605F"
```

15.16 hex_to_num()

- **機能概要**

  IEEE-754 倍精度(または単精度)浮動小数点フォーマットの16進数文字列を Lua数値に変換する。

- **関数定義**

  stat, num = hex_to_num(hexstr)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
hexstr:String	16文字(倍精度64bit)または8文字(単精度32bit)の16進数文字列
num:Number	実数値

- **備考**

  num_to_hex() ライブラリ関数で変換された 16進数文字列を数値に変換する。

  hexstr パラメータで指定した16進数文字列を変換するときにエラーが発生した場合には、stat には false が返り num は nil になります。

- **使用例**

```
stat, x = hex_to_num("3FF3C0CA2A5B1D5D")    -- 1.2345678
stat, x = hex_to_num("3DE0F7BFD11A605F")    -- 1.2345678E-10
stat, x = hex_to_num("BDE0F7BFD11A605F")    -- -1.2345678E-10
```

16 ビット演算API

スクリプト内でビット演算を行うためのライブラリ関数です。abs_agent で提供している API のパラメータやイベントデータを処理するときに必要となる機能を提供しています。

abs_agent で独自に提供しているビット演算ライブラリ関数で扱うことのできる整数値は 0 から (2^31 - 1) です。32 ビット長で MSB が 1 の整数値 (例えば16進数の文字列表記で "FFFFFFFF" や "80000000" の値)はエラーになります。"0" から"7FFFFFFF" のみが有効になりますので注意して下さい。

Lua スクリプトで扱う数値が、(2^31－1) を超えて (2^32－1) までの整数の場合には、Lua の標準ライブラリ bit32 を使用することもできます。bit32 ライブラリで作成した整数値を 16進数文字列へ変換するときには string.format() 関数を使用してください。bit32 ライブラリ関数で取得した値が、2^31- 1 を超える場合には、abs_agent のライブラリ関数の引数として使用することはできませんので注意して下さい。

16.1 bit_tohex()

- **機能概要**

  整数値を16進数表記の文字列に変換する。

- **関数定義**

  hexstr = bit_tohex(x [,n])

- **パラメータとリターン値**

hexstr:String	16進数表記の文字列(“0x” は含まれない)
x:Number	整数値
n:Number	16進数表記の最小桁数を 1 から 8 までの間で指定する(省略時は1)

- **備考**

  変換後の 16進数表記が n で指定した値よりも桁数が多い場合には、n の指定は無視されて全ての桁が返ります。

- **使用例**

```
hexstr = bit_tohex(255,4)
```

16.2 bit_not()

- **機能概要**

  NOTビット演算の結果を返す。

- **関数定義**

y = bit_not(x)

- **パラメータとリターン値**

  y:Number　　整数値

  x:Number　　整数値

- **備考**

- **使用例**

```
y = bit_not(0)
y = bit_not(0x1234)
```

16.3 bit_or()

- **機能概要**

  ORビット演算の結果を返す。

- **関数定義**

  y = bit_or(x1[,x2,..,x<n>])

- **パラメータとリターン値**

  y:Number　　整数値

  x1,..,x<n>:Number　　整数値

- **備考**

- **使用例**

```
y = bit_or(1,2,4,8,16,32,64)
```

16.4 bit_and()

- **機能概要**

  ANDビット演算の結果を返す。

- **関数定義**

  y = bit_and(x1[,x2,..,x<n>])

- **パラメータとリターン値**

  y:Number　　整数値

  x1,..,x<n>:Number　　整数値

- **備考**

- **使用例**

```
y = bit_and(0x1234,0xff00)
```

16.5 bit_xor()

- **機能概要**

  XORビット演算の結果を返す。

- **関数定義**

  y = bit_xor(x1[,x2,...,x<n>])

- **パラメータとリターン値**

  y:Number　　整数値

  x1,...,x<n>:Number　　整数値

- **備考**

- **使用例**

```
y = bit_xor(0xffff,0xaaaa)
```

16.6 bit_lshift()

- **機能概要**

  左シフト演算の結果を返す。

- **関数定義**

  y = bit_lshift(x, n)

- **パラメータとリターン値**

  y:Number　　整数値

  x:Number　　整数値

  n:Number　　整数値(シフト数)

- **備考**

  n で指定されたビット数分、左論理シフトを行ないます。

  ビットシフト時の LSB(最下位ビット) には 0 を代入します。

- **使用例**

```
y = bit_lshift(1,4)
```

16.7 bit_rshift()

- **機能概要**

  右シフト演算の結果を返す。

- **関数定義**

  y = bit_rshift(x, n)

- **パラメータとリターン値**

  y:Number　　整数値

  x:Number　　整数値

  n:Number　　整数値(シフト数)

- **備考**

  n で指定されたビット数分、右論理シフトを行ないます。

  ビットシフト時の MSB(最上位ビット) には 0 を代入します。

- **使用例**

```
y = bit_rshift(1,4)
```

16.8 bit_tosigned()

- **機能概要**

  符号ビット付きで表現された整数を符号付整数に変換する。

- **関数定義**

  y = bit_tosigned(x, w)

- **パラメータとリターン値**

  y:Number　　符号付き整数

  x:Number　　符号ビット付き整数

  w:Number　　MSB(符号ビット) から LSB までのビット幅

- **備考**

  2 の補数で表現された符号ビット付き整数を符号付の数値に変換します。

- **使用例**

```
r = bit_tosigned(0,12)      -- r = 0
r = bit_tosigned(0x7ff,12)  -- r = 2047
r = bit_tosigned(0x800,12)  -- r = -2048
r = bit_tosigned(0xfff,12)  -- r = -1

r = bit_tosigned(0,16)      -- r = 0
r = bit_tosigned(0x7fff,16)  -- r = 32767
r = bit_tosigned(0x8000,16)  -- r = -32768
r = bit_tosigned(0xffff,16)  -- r = -1
```

17 日付時間API

スクリプト内で、日付や時間を計算するときに使用するライブラリ関数です。

17.1 now_datetime()

- **機能概要**

  現在の日付と時刻を返す。

- **関数定義**

  stat, year, month, day, hour, min, sec, millisec = now_datetime()

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
year:Number	現在の年 (1 から 9999 までの整数値)
month:Number	現在の月 (1 から 12 までの整数値)
day:Number	現在の日 (1 から 31 までの整数値)
hour:Number	現在の時 (0 から 23 までの整数値)
min:Number	現在の分 (0 から 59 までの整数値)
sec:Number	現在の秒 (0 から 59 までの整数値)
millisec:Number	現在のミリ秒 (0 から 999 までの整数値)

- **備考**

  Lua ライブラリの os.date('*t') と同様の機能ですが、この関数ではリターン値にミリ秒が追加されてリスト形式で結果が返ります。

- **使用例**

```
local y,m,d,h,mm,s,msec = stat_check(now_datetime())
```

17.2 day_of_week()

- **機能概要**

  指定された日付の曜日を取得する。

- **関数定義**

  stat, wday = day_of_week(year, month, day)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
wday:Number	1 から 7 までの整数値。(1 が日曜日を表す)
year:Number	年を西暦で指定
month:Number	月を指定(1 から12 までの整数値)

day:Number　　日を指定(1 から31 までの整数値)

- **備考**

- **使用例**

```
stat,wday = day_of_week(2009,1,31)
```

17.3 days_in_month()

- **機能概要**

指定された年の指定された月の日数を取得する。

- **関数定義**

stat, days = days_in_month(year, month)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

days:Number　　指定された年の指定された月の日数

year:Number　　年を西暦で指定

month:Number　　月を指定(1 から12 までの整数値)

- **備考**

- **使用例**

```
stat,days = days_in_month(2009,1)
```

17.4 inc_day()

- **機能概要**

指定された日数で変更された日付を返す。

- **関数定義**

stat, new_year, new_month, new_day = inc_day(days, year, month, day)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

new_year:Number　　指定した日数で更新された年

new_month:Number　　指定した日数で更新された月

new_day:Number　　指定した日数で更新された日

days:Number　　この日数を加えた新しい日付を返す。負の値を指定すると前の日付が返される

year:Number　　年を西暦で指定

month:Number　　月を指定(1 から12 までの整数値)

day:Number　　日を指定(1 から31 までの整数値)

- **備考**

- **使用例**

```
stat,y,m,d = inc_day(100,2009,1,31)
```

17.5 inc_second()

- **機能概要**

  指定された秒数で変更された日時を返す。

- **関数定義**

```
stat, new_year, new_month, new_day, new_hour, new_min, new_sec =
                        inc_second(seconds, year, month, day, hour, min, sec)
```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
new_year:Number	指定した秒数で更新された年
new_month:Number	指定した秒数で更新された月
new_day:Number	指定した秒数で更新された日
new_hour:Number	指定した秒数で更新された時
new_min:Number	指定した秒数で更新された分
new_sec:Number	指定した秒数で更新された秒
seconds:Number	この秒数を加えた新しい日時を返す。負の値を指定すると前の日時が返される
year:Number	年を西暦で指定
month:Number	月を指定(1 から12 までの整数値)
day:Number	日を指定(1 から31 までの整数値)
hour:Number	時を指定(0 から23 までの整数値)
min:Number	分を指定(0 から59 までの整数値)
sec:Number	秒を指定(0 から59 までの整数値)

- **備考**

- **使用例**

```
stat,y,m,d,h,mm,s = inc_second(1,2009,1,31,23,59,59)
```

17.6 seconds_between()

- **機能概要**

  指定された２つの日時間の秒数を返す。

- **関数定義**

```
stat, seconds = seconds_between(year1, month2, day1, hour1, min1, sec1,
                                year2, month2, day2, hour2, min2, sec2)
```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。

seconds:Number	２つの日時間の秒数を返す。パラメータの最初の日時よりも次のパラメータの日時が過去の場合には負の値が返される。
year1:Number	最初のパラメータで指定する日時の年を西暦で指定
month1:Number	最初のパラメータで指定する日時の月を指定(1 から12 までの整数値)
day1:Number	最初のパラメータで指定する日時の日を指定(1 から31 までの整数値)
hour1:Number	最初のパラメータで指定する日時の時を指定(0 から23 までの整数値)
min1:Number	最初のパラメータで指定する日時の分を指定(0 から59 までの整数値)
sec1:Number	最初のパラメータで指定する日時の秒を指定(0 から59 までの整数値)
year2:Number	次のパラメータで指定する日時の年を西暦で指定
month2:Number	次のパラメータで指定する日時の月を指定(1 から12 までの整数値)
day2:Number	次のパラメータで指定する日時の日を指定(1 から31 までの整数値)
hour2:Number	次のパラメータで指定する日時の時を指定(0 から23 までの整数値)
min2:Number	次のパラメータで指定する日時の分を指定(0 から59 までの整数値)
sec2:Number	次のパラメータで指定する日時の秒を指定(0 から59 までの整数値)

- **備考**

seconds で返される秒数の**絶対値**が 2147483647(約68年間に相当)を超えた場合にはエラーが発生して stat には false が返ります。

- **使用例**

```
function str_seconds_between(from_date,to_date)
  local stat1,y,m,d,h,min,s = str_to_datetime(from_date)
  local stat2,y2,m2,d2,h2,min2,s2 = str_to_datetime(to_date)
  if not (stat1 and stat2) then
    log_msg("seconds_between ** error **",file_id)
    error()
  end
  local stat,seconds = seconds_between(y,m,d,h,min,s,y2,m2,d2,h2,min2,s2)
  if stat then
    log_msg(string.format("between %s and %s is %d seconds",from_date,to_date,seconds),file_id)
  else
    log_msg("seconds_between ** error **",file_id)
  end
end
str1 = "2011/12/31 1:22:33"
str2 = "2011/12/31 1:22:34"
str_seconds_between(str1,str2)
```

17.7 str_to_datetime()

- **機能概要**

  日付と時刻を表した文字列を日時に変換する。

- **関数定義**

  stat, year, month, day, hour, min, sec, msec = str_to_datetime(str)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
year:Number	str で指定した年を数値に変換した値
month:Number	str で指定した月を数値に変換した値
day:Number	str で指定した日を数値に変換した値
hour:Number	str で指定した時を数値に変換した値
min:Number	str で指定した分を数値に変換した値
sec:Number	str で指定した秒を数値に変換した値
msec:Number	str で指定したミリ秒を数値に変換した値
str:String	日付時刻を表した文字列。 "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。 例: "2001/01/23 01:23:45.067","2001/1/1 12:0:0.001","2001/1/2 1:2:3.100"

- **備考**

  日付または時間部分のみを変換したい場合には、ダミーの日付または時刻を文字列に追加して使用してください。

  ミリ秒を指定しない場合のリターン値 msec は 0 になります。

- **使用例**

```
stat,y,m,d,h,mm,s = str_to_datetime("2011/1/1 13:1:25")
stat,y,m,d,h,mm,s,ms = str_to_datetime("2011/01/01 13:01:25.001")
```

18 GPS API

スクリプト内で、緯度・経度データを操作するためのライブラリ関数です。abs_agent で提供している API のパラメータやイベントデータを処理するときに必要となる機能を提供しています。

TDCP デバイスから受信した GPS イベントデータ(NMEA-0183フォーマット)を処理するときに利用できます。

18.1 gps_utc_to_local()

- **機能概要**

  UTC 日時をローカル日時に変換する。

- **関数定義**

  stat, local_date, local_time = gps_utc_to_local(utc_date, utc_time)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  utc_date:String　　ddmmyy 形式の UTC 日付

  utc_time:String　　hhmmss または hhmmss.sss 形式の UTC 時刻

  local_date:String　　YYYY/MM/DD 形式のローカル日付

  local_time:String　　HH:MM:SS 形式のローカル時刻。UTCTime に指定したミリ秒は切り捨てられる

- **備考**

  GPS NMEA-0183 形式で出力される日付データを、abs_agent の動作するコンピュータのローカル日時に変換します。

- **制限事項**

- **使用例**

```
stat,local_date,local_time = gps_utc_to_local(“160510”,”081855.916”)
```

18.2　gps_distance_course()

- **機能概要**

  ２地点の座標(緯度, 経度)から距離とコースを求める

- **関数定義**

  stat, distance, course = gps_distance_course(unit, lat1, lon1, lat2, lon2

  [,lat1compass, lon1compass, lat2compass, lon2compass])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  unit:String　　パラメータで指定する緯度経度の単位を指定する

  "deg" [-]ddd.dddddd 形式の角度(degree)。南緯もしくは、西経の場合に負の値を示す

  “rad" [-]r.rrrrrr 形式の角度(radian)。南緯もしくは、西経の場合に負の値を示す

  “dmm" GPS NMEA-0183 dddmm.mmmm 形式の座標値。compass パラメータ指定が必要

  "dms" 時分秒で表現する dddmmss.ssss 形式の座標値。compass パラメータ指定が必要

  lat1:Number　　地点１の緯度

  lon1:Number　　地点１の経度

  lat2:Number　　地点２の緯度

  lon2:Number　　地点２の経度

  lat1compass:String　　地点１緯度の方位 "N" 北緯 "S" 南緯。unit が “dmm”, ”dms” の時にのみ必要

  lon1compass:String　　地点１経度の方位 "E" 東経 "W" 西経。unit が “dmm”, ”dms” の時にのみ必要

  lat2compass:String　　地点２緯度の方位 "N" 北緯 "S" 南緯。unit が “dmm”, ”dms” の時にのみ必要

  lon2compass:String　　地点２経度の方位 "E" 東経 "W" 西経。unit が “dmm”, ”dms” の時にのみ必要

  distance:Number　　地点１から地点２までの距離(m)

course:Number地点１から地点２への方角(deg)

- **備考**

２地点間距離は"ヒュベニの公式"を使用して計算しています。また楕円体の定数は GRS80 を使用しています。

方角は大圏コースを計算しています。

地点１が緯度90(deg) の場合に course は 180, 緯度-90(deg) の場合に 0 を返します。

地点１と地点２が同一地点の場合は、course = 0, distance = 0 として値を返します。

course は True Course を表しますので、磁気コンパスの方角とは偏差があります。

- **制限事項**

浮動小数点計算誤差のため 緯度・経度のいずれかが同一の地点間であっても、方角の計算結果が整数値にならない場合があります。

- **使用例**

```
stat,distance,course = gps_distance_course(“dms”,360602.0,1400528.0,353918.0,
1394441.0,”N”,”E”,”N”,”E”)
stat,distance,course = gps_distance_course(“rad”,0.592539,-2.066470,0.709186,-1.287762)
```

18.3 gps_coordinate_deg()

- **機能概要**

座標(緯度, 経度)値を degree 形式に変換する

- **関数定義**

stat, lat_deg, lon_deg = gps_coordinate_deg(unit, lat, lon [,latcompass, loncompass])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
unit:String	パラメータで指定する緯度経度の単位を指定する
	"deg" [-]ddd.dddddd 形式の角度(degree)。南緯もしくは、西経の場合に負の値を示す
	“rad" [-]r.rrrrrr 形式の角度(radian)。南緯もしくは、西経の場合に負の値を示す
	“dmm" GPS NMEA-0183 dddmm.mmmm 形式の座標値。compass パラメータ指定が必要
	"dms" 時分秒で表現する dddmmss.ssss 形式の座標値。compass パラメータ指定が必要
lat:Number	求める地点の緯度
lon:Number	求める地点の経度
latcompass:String	求める地点 緯度の方位 "N" 北緯 "S" 南緯。unit が “dmm”, ”dms” の時にのみ必要
loncompass:String	求める地点 経度の方位 "E" 東経 "W" 西経。unit が “dmm”, ”dms” の時にのみ必要
lat_deg:Number	degree 形式に変換した緯度
lon_deg:Number	degree 形式に変換した経度

- **備考**

- **使用例**

```
stat,lat_deg,lon_deg = gps_coordinate_deg(“dms”,360602.0,1400528.0,”N”,”E”)
```

18.4 gps_coordinate_dmm()

- **機能概要**

  座標(緯度, 経度)値を dmm 形式に変換する

- **関数定義**

  stat, lat_dmm, lon_dmm, lat_dmm_compass, lon_dmm_compass =

  gps_coordinate_dmm(unit, lat, lon *[,*latcompass, loncompass*]*)

- **パラメータとリターン値**

  stat:Boolean　成功した場合は true, 失敗した場合は false が返る。

  unit:String　パラメータで指定する緯度経度の単位を指定する

  ″deg″ [-]ddd.dddddddd 形式の角度(degree)。南緯もしくは、西経の場合に負の値を示す

  “rad″ [-]r.rrrrrr 形式の角度(radian)。南緯もしくは、西経の場合に負の値を示す

  “dmm″ GPS NMEA-0183 dddmm.mmmm 形式の座標値。compass パラメータ指定が必要

  ″dms″ 時分秒で表現する dddmmss.ssss 形式の座標値。compass パラメータ指定が必要

  lat:Number　求める地点の緯度

  lon:Number　求める地点の経度

  latcompass:String　求める地点 緯度の方位 ″N″ 北緯 ″S″ 南緯。unit が “dmm”, ”dms” の時にのみ必要

  loncompass:String　求める地点 経度の方位 ″E″ 東経 ″W″ 西経。unit が “dmm”, ”dms” の時にのみ必要

  lat_dmm:Number　dmm 形式に変換した緯度

  lon_dmm:Number　dmm 形式に変換した経度

  lat_dmm_compass:String　dmm 形式に変換した地点 緯度の方位 ″N″ 北緯 ″S″ 南緯。

  lon_dmm_compass:String　dmm 形式に変換した地点 経度の方位 ″E″ 東経 ″W″ 西経。

- **備考**

- **使用例**

```
stat,lat,lon,lat_compass,lon_compass = gps_coordinate_dmm(“deg”,35.680743,139.768914)
```

19 グローバル共有データAPI

- グローバル共有領域について

グローバル共有データは、abs_agent 上で管理されているキー・バリュー・データ領域です。全てのスクリプトやイベントハンドラ等から常にアクセス可能で、全ての操作がメモリ中で行われますので高速で動作します。

ユーザースクリプトやイベントハンドラ中の Lua 変数は実行中のスクリプト内でのみ有効ですが、グローバル共有データは実行タイミングが後で実行されるスクリプトや別スレッドで実行されるスクリプトから共通にアクセスすることができます。マルチスレッドで同時に実行中のスクリプトから同じグローバル共有データにアクセスすることもできます。

ここで紹介しているライブラリ関数は全て、マルチスレッドの環境で確実に操作できるように内部でリソースの管理が行われています。このため、イベントハンドラやユーザースクリプトからスレッド間の競合を意識することなく使用することができます。

Lua スクリプトからこのライブラリ関数を使用して操作するグローバル共有データは、クライアントコマンド agent_data を使用して同様に操作することができます。

グローバル共有データは、abs_agent が再起動されると内容は消えてしまいます。永続的なデータとして保存する場合にはインストールキットで提供されている BACKUP_RESTORE スクリプトを使用することができます。このスクリプトを使用すると、現在登録中のグローバル共有データをバックアップファイルに保存したり、バックアップファイルファイルからグローバル共有データをロードすることができます。詳しくは BACKUP_RESTORE スクリプトの内容をご覧ください。

永続的なデータを使用したい場合にはこのほかにも、abs_agent のマスター機能や Lua 標準ライブラリ "io" を使用してローカルファイルシステムに保存することができます。永続的なデータを頻繁に保存したい場合には、外部の Web サービスやクラウドサービス等を HTTP クライアントや MQTT クライアント経由でアクセスして利用してください。

- グローバル共有データに保存するデータ型

xxxx_shared_data() の名前が付いたライブラリ関数では、グローバル共有データを操作するときに指定するキーとデータ値の型は共に文字列型です。データ値に数値型を使用する場合には、Lua の tostring() や tonumber() 関数を使用して文字列型との相互変換を明示的に行うか、または暗黙の型変換を使用してください。

もし、Lua の数値型で保存している値(倍精度浮動少数点型) をそのままの型でグローバル共有変数に保存・参照したい場合には、ライブラリ関数名が xxxx_shared_num() のものを使用します。これらのライブラリ関数では、内部で 倍精度浮動少数点型(64ビット長、IEEE-754フォーマット)を16進数文字列に変換した形で保存・参照します。

- リモート共有ライブラリ関数 xxxx_net_data() について

ライブラリ関数名が xxxx_net_data() のものは、リモート側の abs_agent のグローバル共有変数を操作することができます。このライブラリ関数で操作する対象のグローバル共有変数データはリモート側の abs_agent 内のスクリプトから xxxx_shared_data() ライブラリ関数で操作するものと同じものです。

xxxx_net_data() ライブラリ関数の "remote_host" パラメータには、リモート側ホストの IP アドレスまたは "ホスト名" を指定します。このパラメータを省略した場合には、サーバー設定ファイル中の "デフォルト・リモートホスト /Document/ServiceMain/DefaultRemoteHost" 設定項目で指定された値が使用されます。リモート側ホストに "ホスト名" を指定する場合には、"ホスト名"が DNS や他の手段によって適切な IP アドレスに変換できることを確認してください。また、"ホスト名" を指定する場合には IP アドレスを直接指定するよりも、リモート側へのアクセススピードが低下する場合があります。

リモートホスト間との通信データは、abs_agent 内で暗号化されていますのである程度のセキュリティを確保することができます。ただし、インターネットやWAN 間でリモートホストとの通信を行う場合には、より安全な VPN や専用線などと兼用することをお勧めします。

リモート側の abs_agent 側では、"agent_hosts" クライアントプログラムでアクセス許可を行ったホストまたは MAC アドレスを持つ機器からのリクエストのみを処理することができます。

リモート側へのソケット接続に失敗した場合や、この関数を実行したホストからのアクセスをリモート側の abs_agent 側で許可していない場合にはそれぞれエラーが発生します。

リモート側 abs_agent でアクセス許可されていない場合には、リクエスト元サーバーのログに "get_net_data:*ERROR* Command failed, sender-host is not authorized" のメッセージが記録されます。同時に、リモート側サーバーのログには "UpdateGlobalParam:*EXCEPTION* sender-host is not authorized" のメッセージが出力されます。

ソケット接続時にエラーが発生したときは、xxxx_net_data() ライブラリ関数を実行した abs_agent 側でリモートエラーフラグが設定されます。一旦、リモートエラーフラグにリモートホスト名が登録されると、それ以降そのリモートホストに対する xxxx_net_data() ライブラリ関数を含む全てのリモートアクセス関数の実行は、実際のネットワークアクセスを試みずに全て失敗するようになります。このときログには "get_shared_data:*EXCEPTION* lost connection <hostname>" のメッセージが記録されます。

リモートエラーフラグは、グローバル共有文字列リスト(チャンネル名 "$REMOTE_LOST_HOSTS")にリモートホスト名を登録することで実現しています。リモートエラーフラグをクリアするには remove_shared_strlist() ライブラリ関数または clear_shared_strlist()関数を使用します。それぞれのライブラリ関数のチャンネル名パラメータに "$REMOTE_LOST_HOSTS" を指定して、特定のホストまたは全てのホスト名エントリをリモートエラーフラグから取り除くことができます。abs_agent 再起動時には全てのリモートエラーフラグはクリアされます。abs_agent インストール時に提供されている REMOTE_LOST_HOSTS_RESET.lua スクリプトを PERIODIC_TIMER 中から定期的にコールすることで、リモートエラーフラグを一定時間経過後にクリアすることができます。詳しい使用方法はスクリプトファイルを参照してください。

イベントハンドラ等のマルチスレッドで実行するスクリプト中で、xxxx_net_xxxx() ライブラリ関数 を使用する場合には abs_agent のスクリプトプール逼迫に注意する必要があります。xxxx_net_xxxx() ライブラリ関数を実行するイベントハンドラが短時間に多数コールされてかつ、リモート側へのソケット接続が不安定な時には特に考慮しておく必要があります。このような時には、イベントハンドラスクリプトの先頭に script_free_count() ライブラリ関数を使用することで、ソケットエラー検出やソケット接続に時間がかかるときには 一時的にスクリプトの実行をアボート可能にできます。詳しくは script_free_count() ライブラリ関数の項を参照してください。

19.1 get_shared_data(), get_shared_num(), get_net_data()

- **機能概要**

  共有データ領域から key に対応する値を取得する。

- **関数定義**

  stat, value = get_shared_data(key [,delete])

  stat, num = get_shared_num(key [,delete])

  stat, value_tbl = get_shared_data(key_arr [,delete]) -- 複数のデータを同時に取得する場合

  stat, num_tbl = get_shared_num(key_arr [,delete]) -- 複数のデータを同時に取得する場合

  stat, value = get_net_data(key [,delete] [,remote_host])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  value:String　　共有データの key に対応する文字列
  共有データが見つからない場合は、空文字列 "" が設定され、stat には true が返る。

  num:Number　　共有データの key に対応する数値
  共有データが見つからない場合は、stat には false が返され num に値は返らない(nil を設定)

  key:String　　共有データのkey 文字列

  value_tbl:Table of String
  　　複数の共有データの現在の値(Keyと文字列値のペア)が入る。

  num_tbl:Table of Number
  　　複数の共有データの現在の値(Keyと数値のペア)が入る。

  key_arr:Table [1..#key_count] of String
  　　複数の Key文字列を格納した配列

  delete:Boolean　　データ取得後に共有データを自動削除する場合は trueを指定する。
  パラメータ省略時は false になる。

  remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

  複数のキー値に対応するグローバル共有データを同時に取得する場合には、キー文字列を配列に格納して get_shared_data()、get_shared_num() をコールします。リターン値には Key 値と対応する共有データ値がテーブル型で得られます。この時、他のグローバル共有データ更新・参照アクセスとは排他的に動作して、複数の共有データを同時に取得することを保障します。

  get_shared_data() コール時に、key に指定したキー値に対応するグローバル共有変数がなかった場合には stat には true が返り value には "" 空文字列が設定されます。get_shared_num() コール時に key に対応するグローバル共有変数が見つからなかった場合には stat に false が返ります。

get_shared_data() コール時に key_arr に複数のキーを指定したとき、<key#n> に対応するグローバル共有変数がなかった場合には value_tbl[<key#n>] の値は "" 空文字列になります。get_shared_num() をコールするときに、<key#n> に対応するグローバル共有変数がなかった場合には、リターン値の stat に false が設定され、num_tbl[] テーブルの値は返りません(nil を設定)。

key_arr に複数のキーを指定したときに同時に delete_flag を true にすると、指定した全てのキーに対応するグローバル共有変数はデータ取得後に削除されます。

delete_flag を true にした場合、value で値を取得した後削除するまでの間、別スレッドで実行中のスクリプトとは排他的にグローバル共有変数にアクセスします。これによって、value に空文字以外の値を取得するスクリプトは必ず一つになることが保障されます。

- **使用例１**

```
stat,val = get_shared_data(“KeyName1”)
if (stat) then
  log_msg(“data = “ .. val)
end
```

- **使用例２**

```
-----------------------------------------------------------------------------------------------------------------
-- 指定したグローバル共有変数に何か値がセットされるか、もしくは指定されたカウント値 x 10ms 経過するまで
-- 内部でウェイトする。セットされたグローバル共有変数の値を返す。タイムアウト発生時は "" 空文字列が返る。
-----------------------------------------------------------------------------------------------------------------
function global_set_wait(global_name,max_wait_cntr)
  local stat,new_val
  local cntr = 0
  repeat
    wait_time(10)
    cntr = cntr + 1
    stat,new_val = get_shared_data(global_name,true) -- 共有変数の取得を試みた後、常に対象の共有変数を削除する
  until (cntr >= max_wait_cntr) or (new_val ~= "")
  return new_val
end
```

- **使用例３**

```
-----------------------------------------------------------------------------------------------------------------
-- 指定したグローバル共有変数の値が変更されるか、もしくは指定されたカウント値 x 10ms 経過するまで内部でウェイト
-- 最後に取得したグローバル共有変数の値を返す。
```

```
--------------------------------------------------------------------------------------------------------------

function global_change_wait(global_name,start_val,max_wait_cntr)
  local stat,new_val
  local cntr = 0
  repeat
    wait_time(10)
    cntr = cntr + 1
    stat,new_val = get_shared_data(global_name)
  until (cntr >= max_wait_cntr) or (start_val ~= new_val)
  return new_val
end
```

19.2 set_shared_data(), set_shared_num(), set_net_data()

- **機能概要**

  共有データ領域の key とそれに対応する値を設定する。

  key が既に登録済みの場合には指定した値で更新される。

- **関数定義**

  stat = set_shared_data(key, value *[,no_global_watch]*)

  stat = set_shared_num(key, num *[,no_global_watch]*)

  stat = set_shared_data(value_tbl) -- 複数のデータを同時に更新する場合

  stat = set_shared_num(num_tbl) -- 複数のデータを同時に更新する場合

  stat = set_net_data(key, value *[,remote_host]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  key:String　　共有データ Key 文字列。

  value:String　　共有データの Key に対応する文字列値。

  num:Number　　共有データの Key に対応する数値。

  value_tbl:Table of String

  　　共有データに設定する値(Keyと文字列値のペア)を格納したテーブル。

  num_tbl:Table of Number

  　　共有データに設定する値(Keyと数値のペア)を格納したテーブル。

  remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

  no_global_watch:Boolean　　true を指定すると、共有データ変更時に GLOBAL_WATCH イベントの発生条件になっているかのチェックを行わないようにして高速な動作を行う。

  　　パラメータ省略時は false になる。

  　　複数のデータを同時に更新する場合(value_tbl パラメータを指定)には、このパラメータは常に ture に解釈されて GLOBAL_WATCH イベントの検出を行わない。

- **備考**

共有データを消去したい場合は、set_shared_data()をコールするときのパラメータ値 value に空文字列を指定します。複数のキーに対応する共有データを削除する場合には、value_tbl のキーに対応する値に空文字列 "" を指定します。数値型を保存している場合にも共有データ削除時には文字列型を扱うライブラリ関数 set_shared_data() に同じキーを指定して削除します。set_shared_num() ライブラリ関数では共有データを削除することはできません。

複数のキー値に対応するグローバル共有データを同時に更新する場合には、キー文字列と値のペアをテーブルに格納して set_shared_data()、set_shared_num() をコールします。この時、他のグローバル共有データ更新・参照アクセスとは排他的に動作して、複数の共有データが同時に更新されることを保障します。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
if not set_shared_data("KeyName1","新しい値") then error() end
```

19.3 inc_shared_data(), inc_net_data()

- **機能概要**

共有データ領域の key に対応するデータに対して、その現在値を数値とみなして１をインクリメントした値を文字列形式にして設定する。

- **関数定義**

stat, value = inc_shared_data(key *[,no_global_watch]*)

stat, value = inc_net_data(key *[,remote_host]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
value:String	共有データの key に対応するカウンタ値(文字列)
key:String	共有データのkey 文字列
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。
no_global_watch:Boolean	true を指定すると、共有データ変更時に GLOBAL_WATCH イベントの発生条件になっているかのチェックを行わないようにして高速な動作を行う。 パラメータ省略時は false になる。

- **備考**

共有データが見つからないか既存の共有データが数値に変換できない場合は、"1" が共有データに設定されてリターンパラメータの value に "1" 返ります。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,data = inc_shared_data("mykey")
if not stat then error() end
log_msg(string.format("val = %s",data))
```

19.4 dec_shared_data(), dec_net_data()

- **機能概要**

共有データ領域の key に対応するデータに対して、その現在値を数値とみなして１をデクリメントした値を文字列形式にして設定する。デクリメント後の値が "0" 以下("0"を含む)になった場合には共有変数自身を削除する。

- **関数定義**

stat, value = dec_shared_data(key *[,no_global_watch]*)

stat, value = dec_net_data(key *[,remote_host]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
value:String	共有データの key に対応するカウンタ値(文字列)
key:String	共有データのkey 文字列
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。
no_global_watch:Boolean	true を指定すると、共有データ変更時に GLOBAL_WATCH イベントの発生条件になっているかのチェックを行わないようにして高速な動作を行う。 パラメータ省略時は false になる。

- **備考**

ライブラリ関数をコールする前の共有データの値が "1" の場合には、デクリメント後の値が "0" になるため共有データが削除されます。

デクリメント後の値が負の値を示す場合や、key で指定した共有データが見つからないとき、既存の共有データが数値に変換できない場合にも同様に共有データは削除されます。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,data = dec_shared_data("mykey")
if not stat then error() end
log_msg(string.format("val = %s",data))
```

19.5 keys_shared_data(), keys_net_data()

- **機能概要**

  グローバル共有データ領域に保存されている key名リストを取得する。

- **関数定義**

  stat, key_list = keys_shared_data(*[key_prefix]*)

  stat, key_list = keys_net_data(*[remote_host]*)

  stat, key_list = keys_net_data(key_prefix, remote_host)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  key_list:Table [1..#count] of String　　文字列リスト中の文字列（配列）

  remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

  key_prefix:String　　検索対象の Key名（前方一致で検索する）
  空文字列 "" 指定時やパラメータ省略時は全ての Keyが検索対象となる。

- **備考**

  グローバル共有領域に保存されている全てのデータのキー名リストを取得します。

  key_prefix パラメータを指定すると、指定した文字列に前方一致する Key名のみが検索対象になります。

- **使用例**

```
stat,key_list = keys_shared_data()
if not stat then error() end
for key,val in ipairs(key_list) do
  log_msg(string.format("key[%d]= %s",key,val),file_id)
end
```

20 グローバル共有文字列リストAPI

- グローバル文字列リストについて

グローバル共有文字列リストは、文字列リスト名を表す任意の名前を付けたチャンネル(channel) に、複数の文字列エントリを格納することができるリスト形式のメモリ領域です。

グローバル共有文字列リストもキー・バリュー形式のグローバル共有変数と同様に、メモリ内で高速に動作してマルチスレッド下で安全に使用できます。作成可能なチャンネルの数やチャンネル内に格納することができる文字列エントリの数は、abs_agent が動作しているコンピュータの環境に依存します(通常の使用上では制限無く使用できると

考えて問題ありません)。

ユーザースクリプトやイベントハンドラ中で作成したリスト(Luaテーブル変数)は実行中のスクリプト内でのみ有効ですが、グローバル共有文字列リストは実行タイミングが後で実行されるスクリプトや別スレッドで実行されるスクリプトから共通にアクセスすることができます。マルチスレッドで同時に実行中のスクリプトから同じグローバル共有文字列リストにアクセスすることもできます。

ここで紹介しているライブラリ関数は全て、マルチスレッドの環境で確実に操作できるように内部でリソースの管理が行われています。このため、イベントハンドラやユーザースクリプトからスレッド間の競合を意識することなく使用することができます。

Lua スクリプトからこのライブラリ関数を使用して操作するグローバル共有文字列リストは、クライアントコマンド agent_strlist を使用して同様に操作することができます。

グローバル共有文字列リストは、abs_agent が再起動されると内容は消えてしまいます。永続的なデータとして保存する場合にはインストールキットで提供されている BACKUP_RESTORE スクリプトを使用することができます。このスクリプトを使用すると、現在登録中のグローバル共有文字列リスト全体をバックアップファイルに保存したり、バックアップファイルァイルからグローバル共有文字列リストをロードすることができます。詳しくは BACKUP_RESTORE スクリプトの内容をご覧ください。

- グローバル共有文字列リストに保存するデータエントリ型

xxxx_shared_strlist() の名前が付いたライブラリ関数では、グローバル共有文字列リストを操作するときに指定するチャンネルとエントリ値の型は共に文字列型です。エントリ値に数値型を使用する場合には、Lua の tostring() や tonumber() 関数を使用して文字列型との相互変換を明示的に行うか、または暗黙の型変換を使用してください。

もし、Lua の数値型で保存している値(倍精度浮動少数点型) をそのままの型でグローバル共有文字列リストに保存・参照したい場合には、ライブラリ関数名が xxxx_shared_numlist() のものを使用します。これらのライブラリ関数では、内部で 倍精度浮動少数点型(64ビット長、IEEE-754フォーマット)を16進数文字列に変換した形で保存・参照します。

- リモート共有ライブラリ関数 xxxx_net_strlist() について

ライブラリ関数名が xxxx_net_strlist() のものは、リモート側の abs_agent のグローバル共有文字列リストを操作することができます。このライブラリ関数で操作する対象のグローバル共有文字列リストはリモート側の abs_agent 内のスクリプトから xxxx_shared_strlist() ライブラリ関数で操作するものと同じものです。

xxxx_net_strlist() ライブラリ関数の "remote_host" パラメータには、リモート側ホストの IP アドレスまたは "ホスト名" を指定します。このパラメータを省略した場合には、サーバー設定ファイル中の "デフォルト・リモートホスト /Document/ServiceMain/DefaultRemoteHost" 設定項目で指定された値が使用されます。リモート側ホストに "ホスト名" を指定する場合には、"ホスト名"が DNS や他の手段によって適切な IP アドレスに変換できることを確

認してください。また、"ホスト名" を指定する場合には IP アドレスを直接指定するよりも、リモート側へのアクセススピードが低下する場合があります。

リモートホスト間との通信データは、abs_agent 内で暗号化されていますのである程度のセキュリティを確保することができます。ただし、インターネットやWAN 間でリモートホストとの通信を行う場合には、より安全な VPN や専用線などと兼用することをお勧めします。

リモート側の abs_agent 側では、"agent_hosts" クライアントプログラムでアクセス許可を行ったホストまたは MAC アドレスを持つ機器からのリクエストのみを処理することができます。

リモート側へのソケット接続に失敗した場合や、この関数を実行したホストからのアクセスをリモート側の abs_agent 側で許可していない場合にはそれぞれエラーが発生します。

リモート側 abs_agent でアクセス許可されていない場合には、リクエスト元サーバーのログに "get_net_data:*ERROR* Command failed, sender-host is not authorized" のメッセージが記録されます。同時に、リモート側サーバーのログには "UpdateGlobalParam:*EXCEPTION* sender-host is not authorized" のメッセージが出力されます。

ソケット接続時にエラーが発生したときは、xxxx_net_strlist() ライブラリ関数を実行した abs_agent 側でリモートエラーフラグが設定されます。一旦、リモートエラーフラグにリモートホスト名が登録されると、それ以降そのリモートホストに対する xxxx_net_strlist() ライブラリ関数を含む全てのリモートアクセス関数の実行は、実際のネットワークアクセスを試みずに全て失敗するようになります。このときログには "get_shared_data:*EXCEPTION* lost connection <hostname>" のメッセージが記録されます。

リモートエラーフラグは、グローバル共有文字列リスト(チャンネル名 "$REMOTE_LOST_HOSTS")にリモートホスト名を登録することで実現しています。リモートエラーフラグをクリアするには remove_shared_strlist() ライブラリ関数または clear_shared_strlist()関数を使用します。それぞれのライブラリ関数のチャンネル名パラメータに "$REMOTE_LOST_HOSTS" を指定して、特定のホストまたは全てのホスト名エントリをリモートエラーフラグから取り除くことができます。abs_agent 再起動時には全てのリモートエラーフラグはクリアされます。abs_agent インストール時に提供されている REMOTE_LOST_HOSTS_RESET.lua スクリプトを PERIODIC_TIMER 中から定期的にコールすることで、リモートエラーフラグを一定時間経過後にクリアすることができます。詳しい使用方法はスクリプトファイルを参照してください。

イベントハンドラ等のマルチスレッドで実行するスクリプト中で、xxxx_net_strlist() ライブラリ関数 を使用する場合には abs_agent のスクリプトプール逼迫に注意する必要があります。xxxx_net_strlist() ライブラリ関数を実行するイベントハンドラが短時間に多数コールされてかつ、リモート側へのソケット接続が不安定な時には特に考慮しておく必要があります。このような時には、イベントハンドラスクリプトの先頭に script_free_count() ライブラリ関数を使用することで、ソケットエラー検出やソケット接続に時間がかかるときには 一時的にスクリプトの実行をアボート可能にできます。詳しくは script_free_count() ライブラリ関数の項を参照してください。

20.1 add_shared_strlist(), add_shared_numlist(), add_net_strlist()

- **機能概要**

  共有文字列リスト領域の channel パラメータで指定した文字列リストに、新しいエントリを追加する。

- **関数定義**

  stat = add_shared_strlist(channel, value *[,unique [,limit]]*)

  stat = add_shared_numlist(channel, num *[,unique [,limit]]*)

  stat = add_net_strlist(channel, value *[,unique [,limit]] [,remote_host]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	共有データ中の文字列リスト名(文字列)
value:String	文字列リストに追加する文字列エントリ
num:Number	文字列リストに追加する数値エントリ
unique:Boolean	true を指定すると、指定した文字列(または数値)が既に文字列リストに存在する場合は追加しない。このパラメータを省略した場合は false になる。
limit:Number	文字列リストに保持するエントリ数の最大値を制限することができる。このパラメータを省略した場合や 0 を指定した場合には保持できるエントリ数に制限はかからない。保持できる制限数以上に文字列エントリを追加しようとした場合には、追加できる状態になるまで古いエントリから順に自動削除します。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

  channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

  abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat = add_shared_strlist("my_list","entry1",true)
if not stat then error() end
```

20.2 clear_shared_strlist(), clear_shared_numlist(), clear_net_strlist()

- **機能概要**

  共有文字列リスト領域から channel パラメータで指定した文字列リスト全体を削除する。

- **関数定義**

  stat = clear_shared_strlist(channel)

  stat = clear_shared_numlist(channel)

stat = clear_net_strlist(channel [,remote_host])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　共有データ中の文字列リスト名(文字列)

remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

- **使用例**

```
stat = clear_shared_strlist("my_list")
if not stat then error() end
```

20.3　get_shared_strlist(), get_shared_numlist(), get_net_strlist()

- **機能概要**

共有文字列リスト領域から、channel で指定した文字列リスト全体を取得する。

- **関数定義**

stat, strlist = get_shared_strlist(channel [,delete])

stat, numlist = get_shared_numlist(channel [,delete])

stat, strlist = get_net_strlist(channel [,delete] [,remote_host])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

strlist:Table [1..#count] of String　　文字列リスト中の文字列配列

numlist:Table [1..#count] of Number　　文字列リスト中の数値配列

channel:String　　共有データ中の文字列リスト名(文字列)

delete:Boolean　　true を指定すると、文字列リスト取得後に文字列リストを削除する。
このパラメータを省略した場合は false になる。

remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,strlist = get_shared_strlist("my_list",false)
if not stat then error() end
for key,val in ipairs(strlist) do
  log_msg(string.format("list[%d]= %s",key,val),file_id)
end
```

20.4 remove_shared_strlist(), remove_shared_numlist(), remove_net_strlist()

- **機能概要**

共有文字列リスト領域中の channel で指定した文字列リストから、value で指定した文字列や数値に一致するエントリを全て削除する。

- **関数定義**

stat = remove_shared_strlist(channel, value)

stat = remove_shared_numlist(channel, num)

stat = remove_net_strlist(channel, value *[,remote_host]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	共有データ中の文字列リスト名(文字列)
value:String	文字列リストから削除する文字列エントリ
num:Number	文字列リストから削除する数値エントリ
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

value, num で指定したエントリが文字列リスト中に複数個存在する場合には、一致する全てのエントリが削除されます。

channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

同一channel に対して同時に複数の操作が行われる場合には、排他制御によって片方の操作が完了するまで処理が待たされます。

- **使用例**

```
stat = remove_shared_strlist("my_list","entry1")
```

```
if not stat then error() end
```

20.5 shift_shared_strlist(), shift_shared_numlist(), shift_net_strlist()

- **機能概要**

  共有文字列リスト領域から、channel で指定した文字列リストの先頭のエントリを取り出して削除する。

- **関数定義**

  stat, value = shift_shared_strlist(channel [,peek_only])

  stat, num = shift_shared_numlist(channel [,peek_only])

  stat, value = shift_net_strlist(channel [,peek_only] [,remote_host])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	共有データ中の文字列リスト名(文字列)
value:String	取得した先頭の文字列エントリ
num:Number	取得した先頭の数値エントリ
peek_only:Boolean	true を指定するとエントリを取得するだけで、リストからは削除しない。 このパラメータを省略した場合は false になる。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

  channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

  shift_shared_strlist() コール時に文字列リストが空の場合には、value に “” 空文字列が返ります。文字列リストが存在しない場合には エラーで、stat に false が返ります。

  shift_shared_numlist() コール時に文字列リストが空の場合や、文字列リストが存在しない場合には エラーで、stat に false が返ります。

  abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,val = shift_shared_strlist("my_list")
if not stat then error() end
```

20.6 pop_shared_strlist(), pop_shared_numlist(), pop_net_strlist()

- **機能概要**

  共有文字列リスト領域から、channel で指定した文字列リストの最後尾のエントリを取り出して削除する。

- **関数定義**

  stat, value = pop_shared_strlist(channel [,peek_only])

  stat, num = pop_shared_numlist(channel [,peek_only])

  stat, value = pop_net_strlist(channel [,peek_only] [,remote_host])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	共有データ中の文字列リスト名(文字列)
value:String	取得した最後尾の文字列エントリ
num:String	取得した最後尾の数値エントリ
peek_only:Boolean	true を指定すると文字列を取得するだけで、リストからは削除しない。 このパラメータを省略した場合は false になる。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

pop_shared_strlist() コール時に文字列リストが空の場合には、value に “” 空文字列が返ります。文字列リストが存在しない場合には エラーで、stat に false が返ります。

pop_shared_numlist() コール時に文字列リストが空の場合や、文字列リストが存在しない場合には エラーで、stat に false が返ります。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,val = pop_shared_strlist("my_list")
if not stat then error() end
```

20.7 count_shared_strlist(), count_shared_numlist(), count_net_strlist()

- **機能概要**

共有文字列リスト領域から、channel で指定した文字列リストのエントリ数を取得する。

- **関数定義**

stat, count = count_shared_strlist(channel)

stat, count = count_shared_numlist(channel)

stat, count = count_net_strlist(channel [,remote_host])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　共有データ中の文字列リスト名(文字列)

count:Numberg　　文字列リスト中のエントリ数。
文字列エントリが空の場合には 0 が返ります。
文字列リスト自身が存在しない場合には stat に false が返り、count には -1 を設定します。

remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

channel には任意の文字列を指定することができます。同一の文字列リストを操作する場合には channel に指定する文字列も同一のものを使用して下さい。

abs_agentを再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat,count = count_shared_strlist("my_list")
if not stat then error() end
```

20.8　list_shared_strlist(), list_shared_numlist(), list_net_strlist()

- **機能概要**

共有文字列リスト領域に登録済みの channel 名を検索する。

- **関数定義**

stat, channel_list = list_shared_strlist(*[*channel_prefix*]*)

stat, channel_list = list_shared_numlist(*[*channel_prefix*]*)

stat, channel_list = list_net_strlist(channel_prefix *[*,remote_host*]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel_list:Table [1..#count] of String
共有文字列リストに登録されているchannel名リスト

channel_prefix:String　　検索対象のchannel名（前方一致で検索する）
空文字列 "" 指定時やパラメータ省略時は全てのchannel が検索対象となる。
list_net_strlist() では必須パラメータですので省略不可

remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

共有文字列リスト領域に登録された channel 名リストを取得する。

channel_prefix パラメータを指定すると、指定した文字列に前方一致するchannel名のみが検索対象になります。

- **使用例**

```
stat,channel_list = list_shared_strlist("SENSOR")
if not stat then error() end
for key,val in ipairs(channel_list) do
  log_msg(string.format("key[%d] %s",key,val),file_id)
end
```

20.9 find_shared_strlist(), find_shared_numlist(), find_net_strlist()

- **機能概要**

共有文字列リスト領域の channel パラメータで指定した文字列リスト中に、指定したエントリが登録されているかどうかを調べる。

- **関数定義**

stat = find_shared_strlist(channel, value)

stat = find_shared_numlist(channel, num)

stat = find_net_strlist(channel, value *[,remote_host]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	共有データ中の文字列リスト名(文字列)
value:String	文字列リスト中の検索する文字列エントリ
num:Number	文字列リスト中の検索する数値エントリ
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

channel に指定した文字列リスト中から valueや num で指定したエントリを探します。もし見つかった場合は stat に true が設定され、見つからなかった場合には false が設定されます。channel で指定した文字列リストが存在しない場合にも false が返ります。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

- **使用例**

```
stat = find_shared_strlist("my_list","entry1")
if stat then log_msg("found",g_script) end
```

21 グローバル共有メモリエリアAPI

- グローバル共有メモリエリアについて

グローバル共有メモリエリアは、任意のサイズのメモリエリアをスクリプトやイベントハンドラから共有してアクセスできる機能です。メモリエリアにはチャンネル(channel)と呼ばれる任意の名前を付けることができ、複数のチャンネルを同時に扱うことができます。

グローバル共有メモリエリアで作成可能なチャンネルの数やチャンネル内に確保するメモリエリアのサイズは、abs_agent が動作しているコンピュータのメモリサイズが上限となります。

ここで紹介しているライブラリ関数は全て、マルチスレッドの環境で確実に操作できるように内部でリソースの管理が行われています。このため、イベントハンドラやユーザースクリプトからスレッド間の競合を意識することなく使用することができます。

Lua スクリプトからこのライブラリ関数を使用して操作するグローバル共有メモリエリアは、クライアントコマンド agent_shmem を使用して同様に操作することができます。

グローバル共有メモリエリアは、abs_agent が再起動されると内容は消えてしまいます。永続的なデータとして保存する場合には、クライアントコマンド agent_shmem コマンドを使用することでメモリ内容をファイルに保存したり、ファイルからメモリエリアにデータをロードすることができます。メモリ内容を別コンピュータで動作している abs_agent に転送することもできます。詳しくは下記のライブラリ関数の項や、agent_shmem クライアントプログラムの項を参照して下さい。

21.1 shmem_list()

- **機能概要**

  グローバル共有メモリエリアに作成済みの channel 名を検索する。

- **関数定義**

  stat, channel_list, size_list = shmem_list(*[channel_prefix]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel_list:Table [1..#count] of String

  　　グローバル共有メモリエリアに登録されているchannel名リスト

  size_list:Table [1..#count] of Number

メモリエリアの現在の割り当てサイズ(bytes)

channel_prefix:String　検索対象のchannel名（前方一致で検索する）

空文字列 "" 指定時やパラメータ省略時は全てのchannel が検索対象となる。

- **備考**

グローバル共有メモリエリアに登録された channel 名と割り当てメモリサイズのリストを取得する。

channel_prefix パラメータを指定すると、指定した文字列に前方一致するchannel名のみが検索対象にる。

- **使用例**

```
stat,channel_list = shmem_list("BUFF1")
if not stat then error() end
for key,val in ipairs(channel_list) do
  log_msg(string.format("key[%d] %s",key,val),file_id)
end
```

21.2 shmem_resize()

- **機能概要**

指定したサイズのメモリエリアを新規に作成する。

既存のチェンネル名を指定することでメモリエリアサイズの取得やサイズ変更ができる。

- **関数定義**

stat, size = shmem_resize(channel *[,new_size]*)

- **パラメータとリターン値**

stat:Boolean　成功した場合は true, 失敗した場合は false が返る。

size:Number　channel で指定したメモリエリアのサイズ(bytes)。

channel:String　グローバル共有メモリエリアのチャンネル名

new_size:Number　メモリエリアの新しいサイズ、1以上のバイト数を指定する。

channel で指定したメモリエリアが存在しない場合には、新規のメモリエリアを new_size バイト数確保して作成する。

channel で指定したメモリエリアが既に存在する場合には、new_size で指定したバイト数のサイズに縮小または拡張する。縮小した場合に、既存のメモリデータの先頭から new_size 分のデータは保持される。

- **備考**

グローバル共有メモリエリアに新規のメモリエリアを作成したり、既存のメモリエリアのサイズを変更することができる。

new_size パラメータを省略することで、既存のメモリエリアサイズの取得のみを行うことができる。

new_size パラメータには 1 以上のバイト数を指定する。実際にメモリエリアが確保できるかどうかは abs_agent が動作しているコンピュータで利用可能なメモリリソースに依存する。

既存のメモリエリアを削除する場合には shmem_remove() ライブラリ関数を使用する。

このライブラリ関数で新たに確保したメモリエリアには不定値が格納される。

新規のメモリエリアを作成するときにメモリ内容を指定したい場合には、このライブラリ関数の代わりに shmem_store() ライブラリ関数を使用してください。

- **使用例**

```
stat,size = shmem_resize("BUFF1",1024)
if not stat then error() end
```

21.3 shmem_store()

- **機能概要**

パラメータで指定したバイト列データを格納する。

新規のチェンネル名を指定すると、パラメータで指定した内容で初期化されたメモリエリアを作成する。

- **関数定義**

stat = shmem_store(channel, data_arr [,offset])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　グローバル共有メモリエリアのチャンネル名

data_arr:Table [1..#max_item] of Number

　　メモリエリアに格納するデータ配列。空のテーブルは指定できない。

　　各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。

offset:Number　　data_arr で指定したデータをメモリエリアに格納するときに、メモリエリア先頭から offset バイト分ずらして格納する。このパラメータを省略した場合は 0 を指定したのと同じ。

　　-1 を offset に指定すると、メモリエリア終端から data_arr に指定されたデータを追加する。これは現在のメモリエリアのサイズを offset に指定したのと同等。

　　新規のメモリエリアの場合には offset に 0 を指定したのと同じ動作になる。

- **備考**

新規のメモリエリアを channel パラメータに指定した場合には、メモリエリアを作成した後 data_arr パラメータ指定したデータを格納する。確保するメモリエリアのサイズ(Bytes)は (data_arrの個数 +offset)になる。

既存のメモリエリアを channel パラメータに指定した場合には、offset パラメータで指定したオフセット位置から data_arr パラメータで指定したデータを格納する。既存のメモリエリアのサイズ(Bytes)が(data_arr の個数 + offset) よりも小さかった場合には自動でサイズが拡張されてからデータが格納される。

メモリエリアを作成・拡張する場合に、実際にメモリエリアが確保できるかどうかは abs_agent が動作しているコンピュータで利用可能なメモリリソースに依存する。

既存のメモリエリアを削除する場合には shmem_remove() ライブラリ関数を使用する。

- **使用例**

```
stat = shmem_store("BUFF1",hex_to_tbl("0F124F32"),100)
if not stat then error() end
```

21.4 shmem_store_num()

- **機能概要**

パラメータで指定した数値データを倍精度浮動小数点データとして格納する。

新規のチェンネル名を指定すると、パラメータで指定した内容で初期化されたメモリエリアを作成する。

- **関数定義**

stat = shmem_store_num(channel, data_arr)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　グローバル共有メモリエリアのチャンネル名

data_arr:Table [1..#max_item] of Number

メモリエリアに格納する Lua の数値型データ(倍精度浮動集数点データ)配列。空のテーブルは指定できない。各データ項目は IEEE-754 倍精度浮動小数点フォーマットで 8バイト単位に格納される。

- **備考**

新規のメモリエリアを channel パラメータに指定した場合には、メモリエリアを作成した後 data_arr パラメータ指定した数値データを格納する。確保するメモリエリアのサイズ(bytes)は (data_arrの個数 ＊8)になる。

既存のメモリエリアを channel パラメータに指定した場合には、メモリエリアの先頭から data_arr パラメータで指定したデータを格納します。既存のメモリエリアのサイズ(bytes)が(data_arrの個数 ＊ 8) と異なっていた場合には、自動でサイズが拡張・縮小されてからデータが格納される。

メモリエリアを作成・拡張する場合に、実際にメモリエリアが確保できるかどうかは abs_agent が動作してい

るコンピュータで利用可能なメモリリソースに依存する。

このライブラリ関数で格納したデータは shmem_copy_num() ライブラリ関数や DLLライブラリの AG_get_shmem_numarr()、クライアントプログラム agent_shmem に -N オプションを指定することで数値データとして取得できる。

既存のメモリエリアを削除する場合には shmem_remove() ライブラリ関数を使用する。

- **使用例**

```
local num_arr = {1,2.2,-3.3e5)
stat = shmem_store_num("NUM_BUFF1",num_arr)
if not stat then error() end
```

21.5 shmem_copy()

- **機能概要**

グローバル共有メモリエリアに格納されているデータを取得(コピー)する。

- **関数定義**

stat,data_arr = shmem_copy(channel *[,offset [,copy_len]]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　グローバル共有メモリエリアのチャンネル名

data_arr:Table [1..#max_item] of Number
　　メモリエリアに格納されていたデータ配列。
　　各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数。

offset:Number　　メモリエリア先頭から offset バイト分ずらした位置のデータを取得する。このパラメータを省略した場合は 0 を指定したのと同じ。

copy_len:Number　　offset 位置から連続して取得するデータ数(bytes) を整数値で指定する。0 を指定するとメモリエリア終端まで取り出す。このパラメータを省略した場合は 0 を指定したのと同じ。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合にはエラーとなる。

このライブラリ関数を実行してもメモリエリアのデータ内容は変化しない。

- **使用例**

```
stat,data = shmem_copy("BUFF1",100,4)
```

```
if not stat then error() end
log_msg("data_hex = " .. tbl_to_hex(data),g_script)
```

21.6 shmem_copy_num()

- **機能概要**

  グローバル共有メモリエリアに格納されている倍精度浮動小数点データを取得(コピー)する。

- **関数定義**

  stat,data_arr = shmem_copy_num(channel)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel:String　　グローバル共有メモリエリアのチャンネル名

  data_arr:Table [1..#max_item] of Number

  　　メモリエリアに格納されていたIEEE-754 倍精度浮動小数点データ配列。

- **備考**

  channel パラメータに指定したメモリエリアが存在しない場合にはエラーとなる。

  このライブラリ関数で取得するデータを格納するときは shmem_store_num() を使用する。

  取得するデータの個数は常に(メモリエリアサイズ(bytes) / 8)になる。

  このライブラリ関数を実行してもメモリエリアのデータ内容は変化しない。

- **使用例**

```
stat,data = shmem_copy("NUM_BUFF1")
if not stat then error() end
```

21.7 shmem_move()

- **機能概要**

  グローバル共有メモリエリアに格納されているデータを移動(複製)する。

- **関数定義**

  stat = shmem_move(channel ,offset_from, offset_to ,move_len)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel:String　　グローバル共有メモリエリアのチャンネル名

  offset_from:Number　　メモリエリア先頭から offset_from バイト分ずらした位置のデータを複製する。

offset_to:Number　　メモリエリア先頭から offset_to バイト分ずらした位置に複製したデータを書き込む。

move_len:Number　　offset_from 位置から連続して複製するデータ数(bytes) を整数値で指定する。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合にはエラーとなりる。

複製元と複製先のデータ領域が重ならないようにすること。

- **使用例**

```
stat_check(shmem_move("BUFF1",0,90,4))
```

21.8　shmem_load_file()

- **機能概要**

グローバル共有メモリエリアにパラメータで指定したファイルの内容を格納する。

新規のチェンネル名を指定すると、パラメータで指定した内容で初期化されたメモリエリアを作成する。

- **関数定義**

stat = shmem_load_file(channel, filename [,offset])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　グローバル共有メモリエリアのチャンネル名

filename:String　　データファイル名。絶対パス名で指定します。

offset:Number　　filename で指定したファイルの内容をメモリエリアに格納するときに、メモリエリア先頭から offset バイト分ずらして格納する。このパラメータを省略した場合は 0 を指定したのと同じ。

-1 を offset に指定すると、メモリエリア終端から filename に指定されたファイルの内容を追加する。これは現在のメモリエリアのサイズを offset に指定したのと同等。新規のメモリエリアの場合には offset に 0 を指定したのと同じ動作になる。

- **備考**

ローカルファイルの内容をメモリエリアにロードする。ファイルはバイナリモードで読み込む。

パラメータで指定したローカルファイルが存在しない場合にはエラーになる。

新規のメモリエリアを channel パラメータに指定した場合には、メモリエリアを作成した後 data_arr パラメータ指定したデータを格納する。確保するメモリエリアのサイズは (filename で指定したファイルのサイズ + offset)になる。

既存のメモリエリアを channel パラメータに指定した場合には、offset パラメータで指定したオフセット位置から data_arr パラメータで指定したデータを格納する。既存のメモリエリアのサイズが(filename で指定したファイルのサイズ + offset) よりも小さかった場合には自動でサイズが拡張されてからデータが格納される。

メモリエリアを作成・拡張する場合に、実際にメモリエリアが確保できるかどうかは abs_agent が動作しているコンピュータで利用可能なメモリリソースに依存する。

- **使用例**

```
stat = shmem_load_file("BUFF1",/home/pi/mydata,100)
if not stat then error() end
```

21.9 shmem_save_file()

- **機能概要**

グローバル共有メモリエリアの内容をファイルに書き出す。

- **関数定義**

stat = shmem_save_file(channel, filename *[,*offset *[,*copy_len*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名
filename:String	データファイル名。絶対パス名で指定します。
offset:Number	メモリエリア先頭から offset バイト分ずらした位置のデータから書き出す。このパラメータを省略した場合は 0 を指定したのと同じ。
copy_len:Number	offset 位置から連続して書き出すデータ数(bytes) を整数値で指定する。0 を指定するとメモリエリア終端まで取り出す。このパラメータを省略した場合は 0 を指定したのと同じ。

- **備考**

メモリエリアの内容をローカルファイルに書き出す。ファイルはバイナリモードで書き込む。

filename パラメータで指定したファイルが存在する場合には上書きされる。

channel パラメータに指定したメモリエリアが存在しない場合にはエラーとなる。

このライブラリ関数を実行してもメモリエリアのデータ内容は変化しない。

- **使用例**

```
stat = shmem_save_file("BUFF1",/home/pi/mydata,100,0)
if not stat then error() end
```

21.10 shmem_rename()

- **機能概要**

  グローバル共有メモリエリアのチャンネル名を変更する。

- **関数定義**

  stat = shmem_rename(channel, new_channel)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名
new_channel:String	変更する新しいチャンネル名

- **備考**

  既存の指定したグローバル共有メモリエリアのチャンネル名を変更する。

  このライブラリ関数を使用してもメモリエリアの内容は変化しない。

- **使用例**

```
stat = shmem_rename("BUFF1","NEW_CH")
if not stat then error() end
```

21.11 shmem_remove()

- **機能概要**

  グローバル共有メモリエリアを削除する。

- **関数定義**

  stat = shmem_remove(channel)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名

- **備考**

  指定したグローバル共有メモリエリアを削除する。

このライブラリ関数を使用して明示的にメモリエリアを削除しない場合でも、abs_agent シャットダウン時に全てのメモリエリアは自動的に削除される。

- **使用例**

```
stat = shmem_remove("BUFF1")
if not stat then error() end
```

21.12 shmem_fill()

- **機能概要**

グローバル共有メモリエリア全体に指定したバイトデータを書き込む。

- **関数定義**

stat = shmem_fill(channel, val)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名
val:Number	メモリエリア全体に書き込むデータ値。 1バイトに収まる 0から255 (0x0..0xff)の整数。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合はエラーとなる。

- **使用例**

```
stat = shmem_fill("BUFF1",0xff)
if not stat then error() end
```

21.13 shmem_transfer()

- **機能概要**

グローバル共有メモリエリアの指定したチャンネルのデータを別コンピュータに転送する。

- **関数定義**

stat = shmem_transfer(channel, target_host [,remote_host])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名
target_host:String	転送先コンピュータのホスト名。
remote_host:String	転送元コンピュータのホスト名または IP アドレスを指定する。パラメータ省略時

はライブラリ関数を実行したスクリプトが動作するコンピュータ("localhost")になる。

- **備考**

チャンネル名で指定したローカルメモリエリアの内容を別コンピュータに転送する。このライブラリ関数を実行したコンピュータ(転送元)のメモリエリアの内容は変化しない。転送先 abs_agent のハードウエアタイプ(X86,RASPI)が転送元と異なっていても構わない。

転送先コンピュータに作成されるメモリエリアのチャンネル名は転送元と同じになる。もし、転送先に既存のメモリエリアが存在した場合は、転送したデータ内容で置き換えられる。

remote_host パラメータを指定すると、転送操作を行う転送元コンピュータを "localhost" 以外にできる。例えば、別コンピュータから自コンピュータ(ライブラリ関数を実行したコンピュータ)にメモリエリアを転送したい場合には、 "target_host" パラメータに自コンピュータのホスト名または IP アドレスを指定する。このとき自コンピュータ名として "localhost" は使用できません。

メモリエリアを転送する場合に、実際にメモリエリアが確保できるかどうかは転送先の abs_agent が動作しているコンピュータで利用可能なメモリリソースに依存する。

- **使用例(1) 別コンピュータにメモリ領域をバックアップ**

```
stat_check(shmem_transfer("BUFF1","backup_host"))
```

- **使用例(2) 別コンピュータからメモリ領域をリストア**

```
local cmd_result = stat_check(script_exec2("OS/GET_IP",{}))
stat_check(shmem_transfer("BUFF1",cmd_result["IPAddress"],"backup_host"))
```

21.14 shmem_set()

- **機能概要**

グローバル共有メモリエリア中の指定した位置のデータを更新する。

- **関数定義**

stat = shmem_set(channel, offset, val)

stat = shmem_set(channel, offset_arr, val_arr *[,offset_adj]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	グローバル共有メモリエリアのチャンネル名
offset:Number	メモリエリア先頭から offset バイト分ずらした位置のデータを更新する。
val:Number	offset 位置にセットするデータ値。

1バイトに収まる 0から255 (0x0..0xff)の整数。

offset_arr:Table [1..#max_item] of Number

複数のデータをメモリエリアに格納するときに指定する、offset バイトの位置を指定したデータ配列。

val_arr:Table [1..#max_item] of Number

複数のデータをメモリエリアに格納するときに指定する、メモリエリアに格納するデータ配列。

各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。

offset_adj:Number　　offset_arr 配列で指定したオフセットからさらに offset_adj 分ずらした位置のデータを更新する。パラメータ省略時は 0 になる。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合や、offset に指定したデータが存在しない場合にはエラーとなる。

offset_arr, val_arr パラメータを指定するときには、オフセットバイトとその場所に格納する値がテーブル中の各エントリに対応する。また、offset_arr, val_arr パラメータで指定するエントリの数は等しくしてください。

- **使用例**

```
stat = shmem_set("BUFF1",100,0xff)
if not stat then error() end
```

```
index_arr = {10,20,30}
val_arr = {0xaa,0xbb,0xcc}
stat = shmem_set("BUFF1",index_arr,val_arr)
if not stat then error() end
```

21.15 shmem_get()

- **機能概要**

グローバル共有メモリエリア中の指定した位置のデータを取得する。

- **関数定義**

stat, val = shmem_get(channel, offset)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　グローバル共有メモリエリアのチャンネル名

offset:Number　　メモリエリア先頭から offset バイト分ずらした位置のデータを取得する。

val:Number　　メモリエリアの offset 位置にあるデータ値。
1バイトに収まる 0から255 (0x0..0xff)の整数。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合や、offset に指定したデータが存在しない場合にはエラーとなる。

- **使用例**

```
stat,val = shmem_get("BUFF1",100)
if not stat then error() end
```

21.16 shmem_bit()

- **機能概要**

グローバル共有メモリエリア中の指定した位置のデータのビット値を更新する。

- **関数定義**

stat, new_val = shmem_bit(channel, offset, bitnum, flag)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
new_val:Number　　ビット値を更新した後のメモリエリアのoffset 位置にあるデータ値。
1バイトに収まる 0から255 (0x0..0xff)の整数。
channel:String　　グローバル共有メモリエリアのチャンネル名
offset:Number　　メモリエリア先頭から offset バイト分ずらした位置のデータを更新する。
bitnum:Number　　更新するデータ値のビット位置。 0 から 7までの整数。
flag:Boolean　　true を指定した場合には指定したビット値を 1 にする。 false を指定した場合にはビット値を 0 にする。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合や、offset に指定したデータが存在しない場合にはエラーとなる。

bitnum パラメータで指定したビット位置以外の値は変化しない。

- **使用例**

```
stat = shmem_bit("BUFF1",100,7,true)
if not stat then error() end
```

21.17 shmem_or(), shmem_and(), shmem_not(), shmem_xor()

- **機能概要**

  グローバル共有メモリエリア中の指定した位置のデータのビット演算を行う。

- **関数定義**

```
stat, new_val = shmem_or(channel, offset, val)          OR 演算
stat, new_val = shmem_and(channel, offset, val)         AND 演算
stat, new_val = shmem_not(channel, offset)              NOT 演算
stat, new_val = shmem_xor(channel, offset, val)         XOR 演算
```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
new_val:Number	ビット演算を実行した後のメモリエリアの offset 位置にあるデータ値。 1バイトに収まる 0から255 (0x0..0xff)の整数。
channel:String	グローバル共有メモリエリアのチャンネル名
offset:Number	メモリエリア先頭から offset バイト分ずらした位置のデータを更新する。
val:Number	メモリエリアのデータとビット演算を行う対象の値。 1バイトに収まる 0から255 (0x0..0xff)の整数。

- **備考**

  channel パラメータに指定したメモリエリアが存在しない場合や、offset に指定したデータが存在しない場合にはエラーとなる。

- **使用例**

```
stat,val = shmem_and("BUFF1",100,0x0f)
if not stat then error() end
```

21.18 shmem_lshift(), shmem_rshift()

- **機能概要**

  グローバル共有メモリエリア中の指定した位置のデータのビットシフト演算を行う。

- **関数定義**

```
stat, new_val = shmem_lshift(channel, offset, shift)    左ビットシフト演算
stat, new_val = shmem_rshift(channel, offset, shift)    右ビットシフト演算
```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
new_val:Number	ビットシフト演算を実行した後のメモリエリアの offset 位置にあるデータ値。 1バイトに収まる 0から255 (0x0..0xff)の整数。
channel:String	グローバル共有メモリエリアのチャンネル名

offset:Number　　メモリエリア先頭から offset バイト分ずらした位置のデータを更新する。

shift:Number　　シフトするビット数。0 から 7までの整数。

- **備考**

channel パラメータに指定したメモリエリアが存在しない場合や、offset に指定したデータが存在しない場合にはエラーとなる。

左シフト演算を行った場合には LSB に 0 が、右シフト演算を行った場合には MSB に 0 がそれぞれ格納される。

- **使用例**

```
stat,val = shmem_lshift("BUFF1",100,5)
if not stat then error() end
```

21.19　shmem_get_fontx()

- **機能概要**

グローバル共有メモリエリアに格納した FONTX 形式のデータを検索して、指定した文字コードに対応するビットマップデータを取得する。

- **関数定義**

stat,font_arr = shmem_get_fontx(channel ,code)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

channel:String　　FONTX データが格納されているグローバル共有メモリエリアのチャンネル名。

code:Number　　検索する文字コード。
FONTX データ中に記述されている文字コード。マルチバイト文字の場合には、文字コード順にビッグエンディアン形式として表現した１つの整数値を指定する。

font_arr:Table [1..#max_item] of Number
code に指定した文字コードを表すフォントビットマップデータ。
テーブルには文字のフォントサイズ分のビットマップデータ部分が格納される。
各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数。

- **備考**

abs_agent インストールキットで提供されているデフォルトフォントを使用する場合には、フォントライブラリで提供されている初期化関数 fontx_try_init() をコールすることで共有メモリにロードできる。フォントライブラリ詳細とフォント一覧は scripts/preload/013_FONTX に格納されているファイルを参照してください。

channel パラメータに指定したメモリエリアが存在しない場合にはエラーとなる。

channel パラメータに指定するフォントデータは、半角(ASCII)、全角(Shift-JIS)のフォントデータに対応している。Lua スクリプト中で使用している UTF-8形式の文字列から1文字毎の Shift-JIS コードを取得する場合には str_to_tbl2() ライブラリ関数を利用できます。

このライブラリ関数で取得可能なフォントビットマップデータの最大サイズは 512bytes です。(64 x 64ピクセル相当)

正常に処理が終了した場合には stat に true が返り、font_arr にはフォントデータが格納される。指定したフォントが見つからなかった場合などエラー発生時には stat値に false が返り font_arr の値は nil になる。

このライブラリ関数を実行してもグローバル共有メモリエリアのデータ内容は変化しない。

フォントテーブルに格納されているフォントの幅と高さは、グローバル共有変数のオフセット位置(14, 15) のデータを取得することで得られます。(下記例参照)

- **使用例**

```
log_msg("start..",g_script)
fontx_try_init()
local fontx_name = "MISAKI_FONTX"
local stat, x = shmem_copy(fontx_name,14,2)
if not stat then error() end
log_msg("font width = " .. tostring(x[1]) .. " height = " .. tostring(x[2]),g_script)
local str = "試験"
local cstat,ch_arr = str_to_tbl2(str,2) -- Shift-JIS 並びの BigEndian に変換する
if (not cstat) or (#ch_arr == 0) then error() end
for k,v in ipairs(ch_arr) do
  fdata = {}
  stat, fdata = shmem_get_fontx(fontx_name,v)
  if not stat then error() end
  for i,d in ipairs(fdata) do
    log_msg("shmem_get_fontx() tbl[" .. i .. "] = 0x" .. bit_tohex(d,2),g_script)
    wait_time(1)
  end
end
```

22　スクリプト実行API

スクリプト実行ライブラリ関数は、abs_agent の scripts フォルダに格納されている任意のユーザースクリプトやイベントハンドラスクリプトを実行します。ライブラリ関数コール時にスクリプトパラメータを指定すると、実行時にスクリプト内からパラメータ値を取得できます。script_exec2(),script_rexec2() ライブラリ関数では、スクリプト中から設定した複数のリターンパラメータをライブラリ関数をコールした側に返すこともできます。

abs_agent ではライブラリ関数を使用して起動したスクリプト中から、ネストしてスクリプトを実行することができます。このとき、abs_agent ではスクリプト・エンジンのスプールエントリを複数同時に使用します。

スクリプト実行用のライブラリ関数をコールした側では、コールしたスクリプト実行の終了まで待つか、別スレッドでスクリプト実行を行ってその終了を待たずに制御を戻すこともできます。

平行して動作するスクリプト間や、実行タイミングが異なるスクリプト間でデータをやり取りしたい場合には、スクリプトパラメータやリターンパラメータだけではなく、abs_agent で提供されている各種の共有データ機能(グローバル共有データ、グローバル共有文字列リスト、グローバル共有メモリエリア)を利用可能です。これらの機能はマルチスレッド下で安全に操作できるように管理されています。

別のコンピュータで動作している abs_agent のスクリプトを実行することもできます。このときは、script_rexec(), script_rexec2(), script_fork_rexec() ライブラリ関数を使用します。script_net_list() ライブラリ関数を使用すると、リモート側のスクリプトファイル一覧を取得できます。これらのリモートコンピュータにアクセスするライブラリ関数を使用する場合には、リモート側の abs_agent 側で、クライアントプログラム "agent_hosts -a <hostname>" コマンドを使用してライブラリ関数を実行する側のホスト名を追加しておく必要があります。
リモート側への接続に失敗した場合や、この関数を実行したホストからのアクセスをリモート側の abs_agent 側で許可していない場合にはエラーが発生します。

リモート側 abs_agent でアクセス許可されていない場合には、リクエスト元サーバーのログに "script_rexec:*EXCEPTION* ExecuteScript failed, Command failed, sender-host is not authorized" のメッセージが記録されます。同時に、リモート側サーバーのログには "ExecuteScript:*EXCEPTION* [<ScriptName>] sender-host is not authorized" のメッセージが出力されます。

ネットワーク接続時にエラーが発生したときは、ライブラリ関数を実行した abs_agent 側でリモートエラーフラグが設定されます。一旦、リモートエラーフラグにリモートホスト名が登録されると、それ以降そのリモートホストに対する script_rexec(), script_rexec2(), script_fork_rexec(), script_net_list()ライブラリ関数を含む全てのリモートアクセスを行うライブラリ関数の実行は、実際のネットワークアクセスを試みずに全て失敗するようになります。このときログには "script_rexec:*EXCEPTION* lost connection <hostname>" のメッセージが記録されます。

リモートエラーフラグは、リモートアクセスを実行した abs_agent のグローバル共有文字列リスト(チャンネル名 "$REMOTE_LOST_HOSTS")にリモートホスト名が登録されているかどうかで判断します。リモートエラーフラグをクリ

アするには remove_shared_strlist() ライブラリ関数または clear_shared_strlist()関数を使用してグローバル共有文字列リストからリモート側のホスト名エントリを削除します。それぞれのライブラリ関数のチャンネル名パラメータに "$REMOTE_LOST_HOSTS" を指定して、特定のホストまたは全てのホスト名エントリをリモートエラーフラグから取り除くことができます。abs_agent 再起動時には全てのリモートエラーフラグはクリアされます。また、abs_agent インストール時に提供されている REMOTE_LOST_HOSTS_RESET.lua スクリプトを PERIODIC_TIMER 中から定期的にコールすることで、リモートエラーフラグを一定時間経過後にクリアすることができます。詳しい使用方法はスクリプトファイルを参照してください。インストール直後の設定では10分経過すると自動的にリモートエラーフラグがクリアされて、ライブラリ関数コール時に再びネットワークアクセスを試みます。

イベントハンドラ等のマルチスレッドで実行するスクリプト中で、script_rexec(), script_rexec2(), script_fork_rexec(),script_net_list() ライブラリ関数 を使用する場合には abs_agent のスクリプトプール逼迫に注意する必要があります。script_rexec(), script_rexec2(), script_fork_rexec(),script_net_list() ライブラリ関数を実行するイベントハンドラが短時間に多数コールされてかつ、リモート側へのソケット接続が不安定な時には特に考慮しておく必要があります。このような時には、イベントハンドラスクリプトの先頭に script_free_count() ライブラリ関数を使用することで、ソケットエラー検出やソケット接続に時間がかかるときには 一時的にスクリプトの実行をアボート可能にできます。詳しくは script_free_count() ライブラリ関数の項を参照してください。

22.1 script_exec()

- **機能概要**

  スクリプトを実行する。

- **関数定義**

  stat = script_exec(name, key_list, value_list)

  stat = script_exec(name, param_tbl)　　スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  name:String　　スクリプト名。スクリプトがサブフォルダ内にある場合には、'/' 文字でフォルダ名を区切って指定する

  key_list:String　　スクリプトパラメータ Key リスト(CSV形式)
  パラメータが無い場合は、空文字列 “” を指定します。

  value_list:String　　スクリプトパラメータ Value リスト(CSV形式)
  パラメータが無い場合は、空文字列 “” を指定します。

  param_tbl:Table of String
  スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。
  テーブルのキーと値は文字列型で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

  呼び出したスクリプト実行が終了するまで呼び出し元のスクリプトは待ち状態になります。

abs_agent で同時実行可能なスクリプトの数には制限があります。(デフォルト値 16)

- **使用例**

```
local key = list_to_csv("aa¥"¥"","bb''","cc##")
local val = list_to_csv("これは¥"¥"","てすと''","です##")
if not script_exec("SAMPLE",key,val) then error() end
if not script_exec("SAMPLE","","") then error() end
local tbl = {}
tbl["key_for_table_param"] = "value_for_table_param";
tbl["aa¥"¥""] = "これは¥"¥""
tbl["bb''"] = "てすと''"
tbl["cc##"] = "です##"
if not script_exec("SAMPLE",tbl) then error() end
```

22.2 script_exec2()

- **機能概要**

スクリプトを実行して、スクリプト中で指定したリターン値を取得する。

- **関数定義**

stat,result = script_exec2(name, key_list, value_list)

stat,result = script_exec2(name, param_tbl) スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る
result:Table of String	スクリプト中から指定したリターン値(キーと値のペア)が入る。 script_resut() 関数を参照の事。
name:String	スクリプト名。スクリプトがサブフォルダ内にある場合には、'/' 文字でフォルダ名を区切って指定する
key_list:String	スクリプトパラメータ Key リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。
value_list:String	スクリプトパラメータ Value リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。
param_tbl:Table of String	スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。 テーブルのキーと値は文字列型で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

script_exec() の機能と同等ですが、スクリプト内で指定した複数のリターン値を取得できます。リターン値は、スクリプト内で script_result() 関数を実行することで設定できます。リターン値を使用しない場合は、

実行スピードが script_exec2()関数よりも速い script_exec()を使用してください。

呼び出したスクリプト実行が終了するまで、呼び出し元のスクリプトは待ち状態になります。

abs_agent で同時実行可能なスクリプトの数には制限があります。（デフォルト値 16）

- **使用例**

```
stat,result = script_exec2("SAMPLE",list_to_csv("param1","param2"),list_to_csv("xx","yy"))
if not stat then error() end
for key,val in pairs(result) do
  log_msg(string.format("result[%s] = %s",key,val),file_id)
end
```

22.3 script_rexec()

- **機能概要**

リモートコンピュータで動作している abs_agent でスクリプトを実行する。

- **関数定義**

stat = script_rexec(remote_host, name, key_list, value_list)

stat = script_rexec(remote_host, name, param_tbl) スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。
name:String	スクリプト名。スクリプトがサブフォルダ内にある場合には、'/' 文字でフォルダ名を区切って指定する
key_list:String	スクリプトパラメータ Key リスト(CSV形式) パラメータが無い場合は、空文字列 "" を指定します。
value_list:String	スクリプトパラメータ Value リスト(CSV形式) パラメータが無い場合は、空文字列 "" を指定します。
param_tbl:Table of String	スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。 テーブルのキーと値は文字列型で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

リモートコンピュータで動作している abs_agent に設定されたスクリプトを実行します。

セキュリティの為に、リモートコンピュータで動作している abs_agent のマスタ(HostsEquiv)には、呼び出し

元コンピュータのホスト名エントリが登録されている必要があります。これは、クライアントプログラム "agent_hosts -a <hostname>" コマンドで設定できます。

name パラメータで指定するスクリプト名は、リモートコンピュータ側の abs_agent にセットアップされているものを指定します。

呼び出したスクリプト実行が終了するまで、呼び出し側のスクリプトは待ち状態になります。

abs_agent で同時実行可能なスクリプトの数には制限があります。(デフォルト値 16)

- **使用例**

```
local key = list_to_csv("aa¥"¥"","bb''","cc##")
local val = list_to_csv("これは¥"¥"","てすと''","です##")
stat = script_rexec(“sv2","SAMPLE",key,val)
if not stat then error() end
stat = script_rexec(“sv2","SAMPLE","","")
if not stat then error() end
```

22.4 script_rexec2()

- **機能概要**

リモートコンピュータの abs_agent でスクリプトを実行して、スクリプト中から指定したリターン値を取得する。

- **関数定義**

stat, result = script_rexec2(remote_host, name, key_list, value_list)

stat, result = script_rexec2(remote_host, name, param_tbl)

スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。
result:Table of String	スクリプト中から指定したリターン値(キーと値のペア)が入る。 script_resut() 関数を参照の事。
name:String	スクリプト名。スクリプトがサブフォルダ内にある場合には、'/' 文字でフォルダ名を区切って指定する
key_list:String	スクリプトパラメータ Key リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。
value_list:String	スクリプトパラメータ Value リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。

param_tbl:Table of String

スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。
テーブルのキーと値は文字列型で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

リモートコンピュータで動作している abs_agent に設定されたスクリプトを実行します。

セキュリティの為に、リモートコンピュータで動作している abs_agent のマスタ(HostsEquiv)には、呼び出し元コンピュータのホスト名エントリが登録されている必要があります。これは、クライアントプログラム "agent_hosts -a <hostname> " コマンドで設定できます。

name パラメータで指定するスクリプト名は、リモートコンピュータ側の abs_agent にセットアップされているものを指定します。

script_rexec() の機能と似ていますが、script_rexec2() ではリモート側のスクリプト内で指定した複数のリターン値を取得できます。リターン値は、スクリプト内で script_result() 関数を実行することで設定できます。リターン値を使用しない場合は、実行スピードが script_rexec2()関数よりも速い、script_rexec()を使用してください。

呼び出したスクリプト実行が終了するまで、呼び出し側のスクリプトは待ち状態になります。

abs_agent で同時実行可能なスクリプトの数には制限があります。（デフォルト値 16）

- **使用例**

```
stat,result =script_rexec2("sv2","SAMPLE",list_to_csv("param1","param2"),list_to_csv("xx","yy"))
if not stat then error() end
for key,val in pairs(result) do
  log_msg(string.format("result[%s] = %s",key,val),file_id)
end
```

22.5 script_result()

- **機能概要**

script_exec2()、script_rexec2() でコールされるスクリプト中で、リターン値(キーと値のペア) を設定する。
スクリプト中からscript_result() を複数回使用するとリターン値を複数設定できる。

- **関数定義**

stat = script_result([taskid,] key, value)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

taskid:String　　タスクID（指定する場合にはグローバル変数 g_taskidを使用する）
　　(ver1.30 以降の abs_agent ではこのパラメータを省略することができます)
key:String　　リターン値のキー名。
value:String　　リターン値のキーに対応する値。

- **備考**

スクリプトからリターンパラメータを呼び出し元に返すときに使用します。このライブラリ関数を複数回コールして、任意の個数のキー値と値を返すことができます。

- **使用例**

```
if not script_result(g_taskid,"key1","val1") then error() end
if not script_result("key2","val2") then error() end
if not script_result(g_taskid,"リターン#2","値#2") then error() end
if not script_result("リターン#3","値#3") then error() end
```

22.6　script_fork_exec()

- **機能概要**

スクリプトを別スレッドで実行する。スクリプトの終了を待たずに制御が返る。

- **関数定義**

stat, taskid = script_fork_exec(name, key_list, value_list)
stat, taskid = script_fork_exec(name, param_tbl)　スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
taskid:String　　別スレッドで実行開始したスクリプトの g_taskid が返る。
name:String　　スクリプト名
key_list:String　　スクリプトパラメータ Key リスト(CSV形式)
　　パラメータが無い場合は、空文字列 “” を指定します。
value_list:String　　スクリプトパラメータ Value リスト(CSV形式)
　　パラメータが無い場合は、空文字列 “” を指定します。
param_tbl:Table of String
　　スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。
　　テーブルのキーと値は文字列形で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

現在実行中のスクリプトインスタンスとは別のインスタンス＆別スレッドでスクリプトを実行します。呼び出したスクリプト実行の終了を待たずに、呼び出し側のスクリプトに制御が戻ります。

呼び出したスクリプト実行中のエラーは、呼び出し側で検出はできません。コールされる側でエラー処理を行

ってください。（この場合でも、ログにはエラーメッセージが記録されます）

abs_agent で同時実行可能なスクリプトの数には制限があります。（デフォルト値 16）

- **使用例**

```
local key = list_to_csv("aa¥"¥"",",bb''",",cc##")
local val = list_to_csv("これは¥"¥"",",てすと''",",です##")
stat,taskid = script_fork_exec("SAMPLE",key,val)
if not stat then error() end
```

22.7 script_fork_rexec()

- **機能概要**

リモートコンピュータの abs_agent でスクリプトを別スレッドで実行する。スクリプトの終了を待たずに制御が返る。

- **関数定義**

stat, taskid = script_fork_rexec(remote_host, name, key_list, value_list)

stat, taskid = script_fork_rexec(remote_host, name, param_tbl)

スクリプトパラメータをテーブルで指定する場合

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。
taskid:String	別スレッドで実行開始したスクリプトの g_taskid が返る。
name:String	スクリプト名
key_list:String	スクリプトパラメータ Key リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。
value_list:String	スクリプトパラメータ Value リスト(CSV形式) パラメータが無い場合は、空文字列 “” を指定します。
param_tbl:Table of String	スクリプトパラメータが格納されたテーブル(キーと値のペア)を指定する。 テーブルのキーと値は文字列形で、スクリプトパラメータのキーと値にそれぞれ対応する。パラメータが無い場合は、空テーブル {} を指定します。

- **備考**

リモートコンピュータで動作している abs_agent に設定されたスクリプトを実行します。

セキュリティの為に、リモートコンピュータで動作している abs_agent のマスタ(HostsEquiv)には、呼び出し元コンピュータのホスト名エントリが登録されている必要があります。これは、クライアントプログラム

"agent_hosts -a <hostname>" コマンドで設定できます。

name パラメータで指定するスクリプト名は、リモートコンピュータ側の abs_agent にセットアップされているものを指定します。

現在実行中のスクリプトインスタンスとは別のスレッドでスクリプトを実行します。呼び出したスクリプト実行の終了を待たずに、呼び出し側のスクリプトに制御が戻ります。

呼び出したスクリプト実行中のエラーは、呼び出し側で検出はできません。コールされる側でエラー処理を行ってください。（この場合でも、ログにはエラーメッセージが記録されます）

abs_agent で同時実行可能なスクリプトの数には制限があります。（デフォルト値 16）

- **使用例**

```
local key = list_to_csv("aa¥"¥"","bb''","cc##")
local val = list_to_csv("これは¥"¥"","てすと''","です##")
stat,taskid = script_fork_rexec("sv2","SAMPLE",key,val)
if not stat then error() end
```

22.8 script_kill()

- **機能概要**

  実行中のスクリプトを強制終了させる。

- **関数定義**

  stat = script_kill(taskid)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  taskid:String　　実行中のスクリプトを示す taskid を指定する。

- **備考**

  taskid パラメータには、script_fork_exec() ライブラリ関数等をコールした時に得られた、実行中のスクリプトタスクID (taskid) を指定する。指定した taskid に該当するタスクが存在しない場合にはエラーになります。

  この関数を実行すると、実行中のスクリプトタスクに停止指示(signal 値 9 を設定)を与えますが、もしスクリプトがライブラリ関数内部でウェイト状態になっていた場合には、そのライブラリ関数を抜けた後でタスクが強制終了されます。この時でも script_kill() 関数は直ぐに終了して呼び出し側に制御を戻します。

  script_kill() ライブラリ関数で実行中のスクリプトを強制終了させると、ログには下記の様なメッセージが記録されます。

```
ScriptHookFunc:*WARNING* signal[9] received, g_taskid = LUA040AC798254616
TForkScriptExecThread:*EXCEPTION* lua_pcall error detected
```

- **使用例**

```
stat = script_kill("LUA031E554F44520F")
if not stat then error() end
```

22.9 script_signal()

- **機能概要**

  実行中のスクリプトにシグナル値を設定する。

- **関数定義**

  stat = script_signal(taskid_or_name, signal)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
taskid_or_name:String	実行中のスクリプトを示す taskid またはスクリプト名を指定する。
signal:Number	設定するシグナル値を 1 から 255 の整数値で指定する。

- **備考**

  taskid_or_name パラメータには、script_fork_exec() ライブラリ関数等をコールした時に得られた、実行中のスクリプトタスクID (taskid) を指定する。指定した taskid に該当するタスクが存在しない場合にはエラーになります。スクリプト名を指定することもでき、この場合スクリプト名に一致する実行中の全てのタスクにシグナルが設定されます。スクリプト名に '*' を指定した場合は全てのタスクが対象となります。

  signalに 1 から 8 までの値を指定した場合はシグナル設定時のログメッセージが同時に出力されます。10以上 255 以下の値を指定した場合はロメッセージは出力しません。signal 値 9はスクリプト強制終了を意味していて、script_kill() と同じ動作になります。

  この関数を実行すると実行中のスクリプトにシグナル値が設定され、実行中のスクリプト内で g_signal Lua 変数にアクセスすることで、設定したシグナル値を取得できます。シグナルが未設定の場合に g_signal 変数にアクセスすると nil が返ります。

  実行中のスクリプトがライブラリ関数内部でウェイト状態になっていた場合には、そのライブラリ関数を抜けた後でシグナル値が設定されます。この時でも script_signal() 関数は直ぐに終了して呼び出し側に制御を戻します。

  シグナル値はスクリプトインスタンス毎に独立して最後に設定した１つ分が保存されます。シグナルを設定すると以前に設定されていたシグナル値は上書きされます。シグナルを受け取るスクリプト側で g_signal 変数が設定されると同時に設定したシグナル値はクリア(0に設定)されます。

システム関連 API章中の "g_signal" の項も参照してください。

- **使用例**

```
stat_check(script_signal("LUA05A4280F30F48D",10))
```

22.10 script_task_list()

- **機能概要**

現在 abs_agent で実行中のスクリプトのタスクID(g_taskid) リストを取得する。

- **関数定義**

stat, tasklist, namelist = script_task_list()

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

tasklist:Table [1..#max_tasks] of String

　　実行中のスクリプトやイベントハンドラにアサインされた TaskID(g_taskid) 値の配列。

namelist:Table [1..#max_tasks] of String

　　実行中のスクリプトの名前。API 関数やクライアントプログラムからスクリプト実行時に指定したスクリプト名の配列。

- **備考**

tasklist 配列中に格納された各タスクID は、script_fork_exec() 関数実行時に返された g_taskid 文字列や、実行中のスクリプトのグローバル共有変数 g_taskid に格納されている値になります。

- **使用例**

```
local stat,idlist,namelist = script_task_list()
if not stat then error() end
for key,val in ipairs(idlist) do
  log_msg(string.format("task %s name %s is running",val,namelist[key]),file_id)
  wait_time(10)
end
```

22.11 script_exists()

- **機能概要**

スクリプトファイルが存在するかどうかを調べる。

- **関数定義**

stat, path = script_exists(name)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

name:String　　スクリプト名。スクリプトがサブフォルダ内にある場合には、'/' 文字でフォルダ名を区切って指定する

path:String　　スクリプトが存在する場合には、スクリプトファイルのローカルコンピュータ上でのファイルパス名が設定される。

- **備考**

ローカルコンピュータ上に指定したスクリプトファイルが存在するかどうかを調べます。

22.12　script_file_list(), script_net_list()

- **機能概要**

abs_agent に格納されているスクリプトファイル名(Lua ファイル) リストを取得する。

- **関数定義**

stat, filelist = script_file_list(*[*script_folder*]*)

stat, filelist = script_net_list(*[*script_folder *[*, remote_host*]]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

script_folder　　指定したフォルダ以下のスクリプトを検索対象にする。このパラメータを省略した場合や、"" 空文字列を指定した場合には全スクリプトファイルが対象になる。script_folder 中に複数のサブフォルダ名を指定する場合にはフォルダ名の間を "/" で区切ること。("¥" は使えません)

script_net_list() の場合には、remote_host パラメータと共に全てのパラメータを省略するときのみ、script_folder パラメータを省略できます。全てのリモート側のスクリプト名を取得するときには、script_folder に "" 空文字列を指定します。

remote_host:String　　リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

filelist:Table [1..#max_files] of String

(script_folder パラメータを省略した場合)

abs_agent をインストールしたディレクトリ内の "scripts" ディレクトリに格納されている全てのスクリプトファイル名のリスト。ファイル名の拡張子(.lua) はスクリプトファイル名からは除かれる。スクリプトファイルがフォルダに格納されている場合には、フォルダ名が "/" で区切られてスクリプトファイル名にアペンドされる。

(script_folder パラメータを指定した場合)

検索対象のフォルダを、
abs_agent をインストールしたディレクトリ + "/scripts/ + "script_folder パラメータで指定したフォルダ" 以下に内に格納されている全てのスクリプトファイル名のリスト。
ファイル名は script_folder 以下の相対パス名で得られる。ファイル名の拡張子(.lua) はスクリプトファイル名からは除かれる。スクリプトファイルがフォルダに格納されている場合には、フォルダ名が “/” で区切られてスクリプトファイル名にアペンドされる。

- **備考**

この関数で取得したスクリプトファイル名は script_exec() ライブラリ関数等のスクリプトファイル名パラメータに指定できます。ただし、script_folder パラメータを指定した場合には filelist で得られるスクリプトファイルパス名は相対パスになりますので、スクリプトファイル名パラメータに指定するときには、下記の様にして下さい。

script_file_name = "script_folderに指定した文字列" .. "/" .. "filelist[n]で得られた相対パス名"

- **使用例**

```
stat,filelist = script_file_list()
if not stat then error() end
for key,val in ipairs(filelist) do
  log_msg(string.format("script file %s ",val),file_id)
end
```

22.13 script_free_count()

- **機能概要**

スクリプトやイベントハンドラ実行時に必要となる、スクリプトプール中の未使用エントリ数を取得する。

- **関数定義**

count = script_free_count()

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
count:Number　　スクリプトプール中の未使用エントリ数。

- **備考**

スクリプトプール中で現在使用可能な未使用のエントリ数を取得します。abs_agent ではイベントハンドラやユーザースクリプト実行時には、スクリプトプール中から未使用のエントリを使用して実行しています。同時に実行中のイベントハンドラやユーザースクリプトが増えると、この関数で取得するカウント数は少なくなります。実行中のイベントハンドラやユーザースクリプト実行が完了した場合や、エラーで実行が中止されたときに、使用中だったエントリは再びスクリプトプール中で未使用状態に戻ります。

この関数で得られる未使用カウントが 0 になると、新規のイベントハンドラやユーザースクリプトの実行が、ペンディング状態になります。このときログには下記のメッセージが記録されます。

```
2014/11/05 21:19:24 eagle       LUASessionPool       0 SelectFreeLUA:*WARNING* could not find the free index.
```

この時ペンディング状態になっているスクリプトは、スクリプトプール中の未使用エントリが増加(実行中の他のスクリプトが終了)した時点で自動的に実行を再開します。

abs_agent 中で予め用意されているスクリプトプールの総エントリ数は、サーバー設定ファイル(abs_agent.xml)で設定されています。この値はフリー版ライセンスの場合には 16 で固定ですが、スタンダードライセンスの場合には任意の値に設定できます。上記のメッセージが頻繁に記録される場合には、スクリプトプールのエントリ数を増やすか、バックグランドで常に実行されているデーモンタイプのスクリプトをイベント駆動タイプに変更するなどの対応を検討してください。OS 起動と同時に abs_agent を自動起動している場合に上記のメッセージが出力される場合には、abs_agent 起動前に 60秒程度ディレイを確保することで防止できる場合があります。詳しくは "abs_agentインストール"章内の "abs_agent自動起動設定" の項を参照して下さい。

script_free_count() 関数は、イベントハンドラなど同時に複数実行される可能性のあるスクリプト中で時間が掛かる処理を行う場合に使用します。スクリプトの最初の部分で script_free_count()関数で得られた未使用のエントリ数が一定値以下の場合には、以降の時間が掛かる部分の処理を中断して未使用エントリ数をこれ以上増加するのを防ぐようにプログラムできます。このようにすることで、他の新規のイベントハンドラ実行やユーザースクリプト実行に悪影響が及ば無いようなシステムを構築できます。

- **使用例**

23 マスターファイルAPI

abs_agent では、スクリプトやイベントハンドラからマスターファイル機能を利用できます。
マスターファイルは変換テーブルなど、表(XML)の形で表現されたアイテムリストの集合です。abs_agent では１つの XML ファイルで複数のマスターを管理しています。マスターファイルはデフォルトで abs_agent をインストールしたディレクトリ中にファイル名 "masters.xml" で保存されています。

マスターファイル機能は、頻繁に参照アクセスされる情報を管理するのに適しています。マスターの内容はメモリ中にキャッシュ情報として保存されますので、高速にアクセスすることができます。反対に、マスターファイル機能は頻繁に更新される情報を管理する場合には適していません。これらの目的にはグローバル共有変数 API を利用してください。

マスターファイルを、この章で説明しているライブラリ関数を経由しないで、エディタプログラム等を使用して、直接ファイルを編集することもできます。この場合は、マスター項目やアイテムの追加・削除などをファイルを手動で更新することができます。利用シーンとしては、他のシステムから情報をインポートして大量のマスター情報を利用する時に応用できます。エディタプログラムでマスターファイルを手動で修正した場合には、必ず master_reload() ライブラリ関数をコールして実行中の abs_agent に変更を通知する必要があります。agent_stat クライアントプログラムを "-m" オプション付きで実行すると、同様に実行中の abs_agent に最新のマスター情報を反映できます。（詳しくは master_reload() 関数と agent_stat クライントプログラムの項を参照してください）

マスターファイル例：

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
    <Description>application master file</Description>
    <!--                                                              -->
    <!-- ** 注意 **                                                  -->
    <!-- マスターファイルは AppServer プログラムで管理されています。    -->
    <!-- エディタ等で手動で修正した場合は、AppServer を再起動するか、   -->
    <!-- スクリプト中から master_reload() を実行して、                -->
    <!-- キャッシュの更新を行ってください。                           -->
    <!--                                                              -->
    <LastUpdate>2011/11/29 12:57:36</LastUpdate>
    <Master>
      <Sample>
        <Description>Sample master</Description>
        <List>
              <Item>
                  <Name>1</Name>
                  <Code>First</Code>
              </Item>
              <Item>
                  <ID>1234</ID>
                  <Name>All Blue System</Name>
              </Item>
           </List>
        </Sample>
        <HostsEquiv>
            <Description>Equivalent hosts list</Description>
            <List>
```

```
            <Item>
                <HostName>sumomo</HostName>
            </Item>
            <Item>
                <HostName>falcon</HostName>
            </Item>
        </List>
    </HostsEquiv>
</Master>
</Document>
```

⚠ マスターファイル中の文字コードは UTF-8N(BOM無し)を使用して下さい

日本語をマスターファイル中に記述する場合には、UTF-8N(BOM無し)を使用してください。

上記のマスターファイル例では、2つのマスター(“Sample”, ”HostsEquiv”)が定義されています。
各マスター毎に、<Item> タグで囲まれたマスターアイテムが複数格納されています。

マスターの名前やマスターに格納されるアイテムのフィールド(XML タグ名) はユーザーが自由に決めることができます。同一マスター内の各アイテムが、それぞれ異なったフィールドを持つこともできます。上記 ”Sample” マスターを参照してください。

23.1 master_reload()

- **機能概要**

  マスターファイル(masters.xml) のメモリキャッシュをクリアして、最新の状態に更新する。

- **関数定義**

  stat = master_reload()

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

- **備考**

  abs_agent のマスターファイル(masters.xml) をエディタ等で修正した場合に、abs_agent にマスターファイルが修正された事を通知します。

  この関数をコールすると、サーバー起動時にメモリ中に作成したマスターファイルキャッシュ情報が破棄されて、最新のマスターファイルを元にしてメモリ中にキャッシュが作成されます。

  ライブラリ関数 API を使用してマスターを更新する場合には、自動でキャッシュ情報が最新の状態に更新されますので、この関数をコールする必要はありません。

- **使用例**

```
if not master_reload() then error() end
```

23.2 master_create()

- **機能概要**

  マスターファイル中に新規マスターを作成する。

- **関数定義**

  stat = master_create(master_name [,description])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  master_name:String　　マスター名、アルファベットと数字が使用可能。masters.xml ファイル中の〈master〉タグの中に、master_nameで指定したタグが作成されるため、XML のタグで利用可能な文字列のみを使用してください。

  description:String　　マスターの説明文。
  各マスターの中の〈Description〉タグ内に格納する文字列。

- **備考**

  abs_agent のマスターファイル(masters.xml) 中に、マスターを新規に作成します。

  パラメータで指定したマスターが既に存在する場合には、ライブラリ関数はエラーを返します。

- **使用例**

```
if not master_create("customers","顧客テーブル") then error() end
```

23.3 master_exist()

- **機能概要**

  マスターファイル中にマスターが存在するかどうかを調べる。

- **関数定義**

  stat = master_exist(master_name)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  master_name:String　　マスター名、アルファベットと数字が使用可能。masters.xml ファイル中の〈master〉タグの中に、master_nameで指定したタグが存在するかどうかを調べる。

- **備考**

  abs_agent のマスターファイル(masters.xml) 中に、マスターが既に作成済みの場合には true が返り、存在

しない場合は false が返ります。

- **使用例**

```
if not master_exist(“customers”) then
  master_create(“customers”,”顧客テーブル”)
end
```

23.4 master_delete()

- **機能概要**

マスターファイルから指定したマスターを削除する。

- **関数定義**

stat = master_delete(master_name)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

master_name:String　　マスター名。

- **備考**

abs_agentのマスターファイル(masters.xml) から指定したマスターを削除します。

マスターに含まれる全てのアイテムエントリも同時に削除されます。

- **使用例**

```
if not master_delete(“customers”) then error() end
```

23.5 master_item_add()

- **機能概要**

マスター中に新規アイテムエントリを追加する。

- **関数定義**

stat = master_item_add(master_name, tag_list, val_list)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

master_name:String　　マスター名

tag_list:String　　マスターに追加するアイテムエントリのタグ名リスト(CSV形式で指定する)

val_list:String　　マスターに追加するアイテムエントリの値リスト(CSV形式で指定する)

- **備考**

master_name で指定したマスターに、アイテムエントリを新規に追加します。

Item タグ中に作成されるフィールドのタグ順序は、tag_listに指定したタグ名リストの順序と同じになります。tag_list の CSV の各カラムに対応するタグの内容は、val_list の対応する同一カラムに指定された値に設定されます。この時、tag_list と val_list のCSV 項目数は必ず等しくしてください。

- **使用例**

```
if not master_item_add(“Casting”,
                       list_to_csv(“ID”,”Name”),
                       list_to_csv(“0001”,”Tom Cruise”)) then error() end
```

上記の例では、"Casting" の名前で作成されている既存のマスターに新規の "Item" エントリを追加します。"Item" エントリ中には 2 つのフィールドが作成されて、それぞれ "ID" と "Name" タグが作られます。またそのタグの内容にはそれぞれ "0001" と "Tom Cruise" が設定されます。

23.6 master_item_delete()

- **機能概要**

マスター中から指定したアイテムエントリを削除する。

- **関数定義**

stat = master_item_delete(master_name, item_tag1 *[,silent]*)

stat = master_item_delete(master_name, idx *[,silent]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
master_name:String	マスター名
item_tag1:String	Item タグ中の第１フィールド(チャイルドタグ)の内容が、このパラメータで指定した内容と一致する最初のアイテムエントリを削除する
item_idx:Number	マスター中の指定したエントリ番号のアイテムエントリを削除する。 1 以上の整数を指定する。
silent:Boolean	true を指定すると削除対象のエントリが見つからなかった場合でもエラーを検出しない。パラメータ省略時は false が設定される。

- **備考**

master_name に指定したマスタから、パラメータで指定したアイテムエントリを削除します。

item_tag1 パラメータを使用する場合で、内容に一致するアイテムエントリが複数見つかった場合には、最初に見つかったアイテムエントリのみが削除されます。

idx に指定する番号は、master_item_list() 関数でアイテムエントリのリストを取得したときの、要素番号と同じものを指定します。１から始まる整数で、指定可能な最大数は全アイテムエントリ数と同一です。

- **使用例**

```
if not master_item_delete(“Casting”,”0001”) then error() end
```

23.7 master_item_list()

- **機能概要**

マスターにある全アイテムエントリの、チャイルドタグの値リストを取得する。

アイテムエントリ毎に存在するマスター項目数が増減する場合には、項目数が一番多いエントリに合わせたリストを返す。

- **関数定義**

stat, item_tag1, item_tag2, item_tag3 *[, ... ,item_tag#n]* = master_item_list(master_name)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

master_name:String　　マスター名

item_tag1:array [1..#max_item] of String　　指定したマスター中にある各Item タグ内の第１チャイルドタグの内容リスト。

item_tag2:array [1..#max_item] of String　　指定したマスター中にある各Item タグ内の第２チャイルドタグの内容リスト。

item_tag3:array [1..#max_item] of String　　指定したマスター中にある各Item タグ内の第３チャイルドタグの内容リスト。

- **備考**

item_tag<n>(<n> = 1, 2, 3) テーブルに入るタグ内容のデータ数は、指定したマスターの全アイテムエントリ数に等しくなります。

Item タグ内の、チャイルドタグの最初のものが常に第１タグとみなして、タグ自身の名称は考慮しないので注意してください。Item タグ内のチャイルドタグ数が３未満の場合は、対応するitem_tag<n>リスト項目値に、空文字 '' が設定されます。

item_tag1, item_tag2, item_tag3 は必ずリターン値に含まれます。もし、アイテムタグ中のチャイルドタグが 3 以上ある場合には全てのチャイルドタグ数が収まるだけのテーブルがリターン値に返されます。もし、アイテムエントリ毎のタグ数よりもリターン値で返すタグが多い場合には、そのリスト項目の値は空文字 '' になります。

このライブラリ関数はアイテムエントリ中の各マスター項目を表すチャイルドタグ位置のみに着目して、リストを作成します。このため、アイテムエントリ中でマスター項目タグ名が同じでもチャイルドタグ位置が変化している場合には、リターン値で得られる各項目の値リストが正確になりません。この場合には master_item_tagval() ライブラリ関数を使用して下さい。

- **使用例**

```
local stat,tag1,tag2,tag3 = master_item_list("HostsEquiv")
if not stat then error() end
for key,val in ipairs(tag1) do
   log_msg(string.format("item[%d] =1:%s 2:%s 3:%s",key,tag1[key],tag2[key],tag3[key]))
end
```

アイテムタグ数が多い場合の例(関数のリターン値をテーブルに格納して処理を行う)

```
function master_print(master)
  local res_list = {master_item_list(master)}
  if not res_list[1] then error() end
  for key,val in ipairs(res_list[2]) do
    log_msg("---------------------------------------")
    for i = 2,#res_list,1 do
      log_msg(string.format(master .. "[%d] tag[%d] = %s",key,(i - 1),res_list[i][key]))
    end
  end
end
master_print("Sample")
master_print("HostsEquiv")
```

23.8 master_item_find()

- **機能概要**

マスター中の、指定したタグの名前と内容に一致するアイテムエントリを取得する。

- **関数定義**

stat, item_tbl = master_item_find(master_name, item_tag_name, item_tag_value)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

master_name:String　　マスター名

item_tag_name:String　　検索するタグの名前

item_tag_value:String　　検索するタグの値。

item_tbl:Table of String 最初に見つかったマスターエントリ(Itemタグ)の内容

　item_tbl[<item_tag_name#1>"]:String　　Item タグ中の第１フィールド(チャイルドタグ)の内容

　item_tbl[<item_tag_name#2>"]:String　　Item タグ中の第２フィールド(チャイルドタグ)の内容

　…

　item_tbl[<item_tag_name#n>"]:String　　Item タグ中の第nフィールド(チャイルドタグ)の内容

- **備考**

master_name に指定したマスタの全アイテムエントリから、item_tag_name に指定したタグ名の値が、

item_tag_value と一致するアイテムエントリを探します。もし見つかった場合は、stat に true が設定され、item_tbl にマスタのアイテムエントリ内容が設定されます。一致するアイテムエントリが見つからなかった場合は、 stat に false が設定されます。

指定したタグと名前に一致する アイテムエントリがマスター内に複数見つかった場合には、最初に見つけたアイテムエントリをリターン値として返します。

item_tbl はテーブル型で、キーにアイテムエントリ(Itemタグ)のチャイルドタグ名(アイテムフィールド名)が入り、値にそのタグの内容が入ります。

- **使用例**

```
local stat,item = master_item_find("Sample","Entry1","1")
if not stat then error() end
for key,val in pairs(item) do
  log_msg(string.format("master_item[%s] = %s",key,val),file_id)
end
```

23.9 master_item_tagval()

- **機能概要**

マスターにある全アイテムエントリの、指定したタグの値リストを取得する。
リストの長さはアイテムエントリ数と同じになる。

- **関数定義**

stat, item_tag = master_item_tagval(master_name, item_tag_name)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
master_name:String　　マスター名
item_tag_name:String　　検索するタグの名前
item_tag:array [1..#max_item] of String
　　指定したマスター中にある、<item_tag_name>タグの内容リスト。

- **備考**

master_name に指定したマスタの全アイテムエントリから、item_tag_name に指定したタグの値をリストにして返します。

アイテムエントリ中に item_tag_name で指定した名前のタグが見つからない場合には、空文字 "" がリスト項目に格納されます。

このライブラリ関数では、アイテムエントリ中のチャイルドタグの位置に関わらず item_tag_name に指定したタグ名を検索してリストに格納します。

- **使用例**

```
local stat,item = master_item_tagval("Sample","Entry1")
if not stat then error() end
for key,val in ipairs(item) do
  log_msg(string.format("master_item[%d] = %s",key,val),file_id)
end
```

24 HTTP クライアント, JSON変換

abs_agent のスクリプト中から、外部のアプリケーションサーバーやアプリケーションプログラムに対して HTTP プロトコルでアクセスしてレスポンス文字列(JSON)を取得できます。

Web API サービスでよく使用される HTTP-GET, HTTP-POST, HTTP-PUT コマンドをサポートしています。
HTTP-GET コマンドの場合にはURL パラメータを指定できます。HTTPプロトコルで送信されるヘッダ情報には、任意のカスタムヘッダを追加することもできます。

24.1 http_get()

- **機能概要**

  HTTP プロトコル GET コマンドで URL にアクセスしてリプライ文字列を取得する。(JSON 形式でのリプライ取得用)

- **関数定義**

  stat, reply_str = http_get(host, port, path *[,*param_tbl *[,*header_tbl *[,*ucs2utf*]]])*

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  host:String　　Webサービスを公開しているホスト名

  port:Number　　HTTP-GET リクエスト時のポート番号

  path:String　　Webサービスのパス名

  param_tbl:Table of String
  　　URLパラメータが格納されたテーブル(キーと値のペア)を指定する。
  　　テーブルのキーと値は文字列型で、URLパラメータのキーと値にそれぞれ対応する。

  header_tbl:Table of String
  　　HTTP カスタムヘッダ情報が格納されたテーブル(キーと値のペア)を指定する。
  　　テーブルのキーと値は文字列型で、HTTP ヘッダパラメータのキーと値にそれぞれ対応する。

  ucs2utf8:Boolean　　リプライ文字列が Unicode(UCS-2)文字で “¥uxxxx” エンコード表記されていた場合に UTF-8 コードに変換する。(省略時は true に指定される)

  reply_str:String　　HTTP-GET リクエストを送信したサーバーから返されたレスポンス文字列

レスポンス文字列中に改行コードが含まれていた場合でも、全体を１つの文字列として取得します

- **備考**

パラメータで指定された値を元に下記の URL に対して HTTP GET コマンドを実行します。

リプライデータは１つの文字列として取得しますので、JSON 形式のリプライデータの取得に適しています。HTML や XML など、複数行にまたがるリプライデータの取得には適していません。

```
http://<host>[:<port>]<path>[?[key#1=val#1[&key2=val#2]...]]
```

HTTP GET コマンドでアクセスするときの URL は上記になります。<host> は、host パラメータで指定された文字列が入ります。<port> は portパラメータで指定されたポート番号を文字列に変換したものが入ります。80 を指定した場合にはこの部分は省略されます。<path> は、path パラメータで指定された文字列が入ります。<key#n>,<val#n> には param_tbl パラメータで指定したキーと値のペアがそれぞれ URL パラメータに指定しされます。尚、各 URLパラメータは自動的に URL エンコードされます。

下記はライブラリ関数の使用例です。

```
local host = "localhost"
local port = 8080
local path = "/command/json/script"
local params = {}    -- URL パラメータ
params["session"] = "1234"
params["name"] = "PARAM_ECHO"
params["key1"] = "val1"
params["キー２"] = "値２"
local stat,rpl = http_get(host,port,path,params)
if not stat then error() end
local rpl_json = g_json.decode(rpl)
log_msg(string.format("Result    = %s",rpl_json.Result),file_id)
log_msg(string.format("ErrorText = %s",rpl_json.ErrorText),file_id)
log_msg(string.format("TaskID    = %s",rpl_json.TaskID),file_id)
if (rpl_json.Result == "Success") then
  for key,val in pairs(rpl_json.ResultParams) do
    log_msg(string.format("ResultParam[%s] = %s",key,val),file_id)
  end
end
```

このときには下記の URL がアクセスされます。

```
http://localhost:8080/command/json/script?session=ST02983095510584&name=PARAM_ECHO&
key1=val1&%E3%82%AD%E3%83%BC%EF%BC%92=%E5%80%A4%EF%BC%92
```

- **使用例**

  上記記事中の Lua スクリプトをご覧ください

24.2 http_post(), http_put()

- **機能概要**

  HTTP プロトコル POST または PUTコマンドで、指定したデータをURL に送信してリプライ文字列を取得する。(JSON 形式でのリプライ取得用)

- **関数定義**

  stat, reply_str = http_post(host, port, path [,data_tbl [,header_tbl [,ucs2utf]]])

  stat, reply_str = http_put(host, port, path [,data_tbl [,header_tbl [,ucs2utf]]])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  host:String　　Webサービスを公開しているホスト名

  port:Number　　HTTP POST, HTTP PUTリクエスト時のポート番号

  path:String　　Webサービスのパス名

  data_tbl:Table [1..#max_str_line] of String

  　　URL に送信するデータが格納されたテーブル(文字列配列)を指定する。
  　　任意の数の文字列を配列形式(テーブル)に格納して指定することが出来ます。
  　　配列中の各文字列を送信するときに 終端に 0x0d, 0x0a を付加します。

  header_tbl:Table of String

  　　HTTP カスタムヘッダ情報が格納されたテーブル(キーと値のペア)を指定する。
  　　テーブルのキーと値は文字列型で HTTP ヘッダパラメータのキーと値にそれぞれ対応する。

  ucs2utf8:Boolean　　リプライ文字列が Unicode(UCS-2)文字で “¥uxxxx” エンコード表記されていた場合に UTF-8 コードに変換する。(省略時は true に指定される)

  reply_str:String　　HTTP-GET リクエストを送信したサーバーから返されたレスポンス文字列
  　　レスポンス文字列中に改行コードが含まれていた場合でも、全体を１つの文字列として取得します

- **備考**

  指定した パラメータの値を元に求めた URL アドレスに対して HTTP POST または PUTコマンドを実行します。

  リプライデータは１つの文字列として取得しますので、JSON 形式で返されるリプライデータの取得に適してい

ます。(HTML や XML など、複数行にまたがるリプライデータの取得には適していません)

HTTP プロトコルでサーバーにアクセスした時のリクエストデータは下記のようになります

```
PUT <path> HTTP/1.1
Connection: close
Content-Length: xxxxxx
<header_tbl#1 key>: <header_tbl#1 value>
<header_tbl#2 key>: <header_tbl#2 value>
..
<header_tbl#n key>: <header_tbl#n value>
Host: <host>
Accept: */*
User-Agent: Mozilla/3.0 (compatible; Indy Library)

<data_tbl#1>
<data_tbl#2>
..
<data_tbl#n>
```

上記は、http_put() ライブラリ関数を使用したときにサーバー側に送信されるデータ例です。TCP/IP ソケット通信で接続するときのホスト名とポート番号は、それぞれ host パラメータと port パラメータに指定した値を使用します。

送信データ中の <host> 部分には host パラメータで指定された文字列が入ります。<path> 部分には path パラメータで指定された文字列が入ります。<header_tbl#n key>と<header_tbl#n value>には、header_tblパラメータで指定されたカスタムヘッダテーブル(連想配列)のキー名と値が入ります。<data_tbl#n>には data_tbl パラメータで指定されたテーブル(文字列配列)の値が入ります。

- **使用例**

```
local data = {}
table.insert(data,"str1")
table.insert(data,"str2")
local header = {}
header["X-ABS-MyKey"] = "KeyData"
stat,rpl = http_post("server_host_name",80,"/api/cmd",data,header)
if not stat then error() end
```

24.3 g_json.decode() ,g_json.encode()

- **機能概要**

JSON 文字列を Lua テーブルに変換する。g_json.decode()

Lua テーブルを JSON 文字列に変換する。g_json.encode()

- **関数定義**

tbl = g_json.decode(json_str)

json_str = g_json.encode(tbl)

- **パラメータとリターン値**

tbl:Table　　JSON を Lua のテーブルに変換したもの。

json_str:String　　JSON 文字列。

- **備考**

abs_agent 起動時に preload 機能によって、"preload/011_JSON" フォルダに格納されたファイルをロードすることで定義されます。フォルダには下記のライブラリ(json.lua)が配置されています。

```
JSON4Lua: JSON encoding / decoding support for the Lua language.json Module.
Author: Craig Mason-Jones
http://json.luaforge.net/
```

g_json.decode() 関数で JSON 文字列をテーブルに変換する場合に、"¥uxxxx" 形式のエンコードフォマットの文字列を自動的に UTF-8 にデコードすることができません。このため、abs_agent の http_get() 関数で取得した JSON 文字列をこの関数で変換する場合には、http_get() 関数の "ucs2utf8" パラメータを "true" に設定して予め UTF-8 に変換しておくことで、日本語を含む JSON 文字列を正しく Lua のテーブルに変換できます。

g_json テーブルの定義は、上記ライブラリ関数と同じフォルダに格納された z_globalload.lua ファイル中で下記の様に定義されます。

```
g_json = require('json')
```

- **使用例(Yahoo API を使用して座標から住所を取得する)**

```
file_id = "WEBAPI_REVERSEGEO"
log_msg("start..",file_id)
local host = "reverse.search.olp.yahooapis.jp"
local port = 80
local path = "/OpenLocalPlatform/V1/reverseGeoCoder"
local params = {}
params["appid"] = "<Your_Yahoo_API_ID>"
params["lon"] = "135.677805"
params["lat"] = "35.013636"
```

```
params["output"] = "json"
local header = {}


local stat,rpl = http_get(host,port,path,params,header,true)
if not stat then error() end
local rpl_json = g_json.decode(rpl)
log_msg(string.format("Count=  %d",rpl_json.ResultInfo.Count),file_id)
for i = 1,rpl_json.ResultInfo.Count do
    log_msg(string.format("住所 =  %s",rpl_json.Feature[i].Property.Address),file_id)
end
```

実行結果(ログ出力)

```
2010/12/11 16:40:19 falcon     WEBAPI_REVERSEGEO    0 start..
2010/12/11 16:40:19 falcon     WEBAPI_REVERSEGEO    0 Count=  1
2010/12/11 16:40:19 falcon     WEBAPI_REVERSEGEO    0 住所 =  京都府京都市右京区嵯峨天龍寺芒ノ馬場町
```

- **使用例(JSON を使用して Lua テーブルをシリアライズしてパラメータに渡す)**

Luaテーブルを JSON でシリアライズしてパラメータに渡すスクリプト(TEST/JSON_PARAM_SEND)

```
-- Lua テーブルを JSON 文字列にシリアライズして、スクリプトパラメータに渡す
log_msg("start..",g_script)
local test = {
  one='first',two='second',three={2,3,5},four='テーブルデータ',five={a='入れ子テーブルデータ',b=999}
}
local jsonTest = g_json.encode(test)
local param = {} -- スクリプトパラメータ
param["p1"] = jsonTest
if not script_exec("TEST/JSON_PARAM_RECV",param) then error() end
```

パラメータを受け取って Lua テーブルに戻すスクリプト(TEST/JSON_PARAM_RECV)

```
-- スクリプトパラメータで JSON 文字列でシリアライズ化された Lua テーブルを受けてテーブルに戻す
log_msg("start..",g_script)
function sprint(s)
  log_msg(s,g_script);
end
local tbl = g_json.decode(g_params["p1"])
sprint('resut.one = ' .. tbl.one)
sprint('resut.two = ' .. tbl.two)
sprint('resut.three[2] = ' .. tbl.three[2])
sprint('resut.four = ' .. tbl.four)
```

```
sprint('resut.five.a = ' .. tbl.five.a)
sprint('resut.five.b = ' .. tbl.five.b)
```

実行結果(ログ出力)

```
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_SEND 0 start..
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 start..
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.one = first
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.two = second
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.three[2] = 3
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.four = テーブルデータ
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.five.a = 入れ子テーブルデータ
2018/06/28 08:30:33 raspberryp TEST/JSON_PARAM_RECV 0 resut.five.b = 999
```

25 WebSocket サーバーAPI

WebSocket サーバー機能を有効にしている場合に使用できるライブラリ関数です。

WebSocket クライアント接続に対してテキストフレームやバイナリフレームを送信することができます。

25.1 websocket_emit_text(), websocket_emit_binary()

- **機能概要**

  WebSocket クライアントにテキストフレームまたはバイナリフレームを送信する。

- **関数定義**

  stat = websocket_emit_text([channel_or_wsid,] msg)

  stat = websocket_emit_binary([channel_or_wsid,] data_arr)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel_or_wsid:String

  送信先WebSocket クライアントのチャンネル名もしくは WebSocketID 文字列。
  チャンネル名は WebSocket クライアントが接続するときに指定した URL パス名 "/<channel>/<SessionToken>" 中の <channel> に指定した文字列。複数のクライアント接続で同じチャンネル名を使用している場合には、チャンネル名が一致する全てのクライアントにフレームを送信する。

  WebSocketID 文字列は、WEBSOCKET_DATA イベントハンドラ中に渡されたパラメータ値 g_params["WebSocketID"] に格納されている。WebSocketID をパラメータに指定すると、WEBSOCKET_DATA イベント発生時に送信してきたクライアント接続に対してのみフレームを送信する。WebSocketID 文字列は WebSocketクライアント接続が持

続している間は変化しませんが、クライアント側から再接続を行った場合には別の文字列に変更されます。

channel_or_wsid パラメータを省略するか空文字列 '' を指定した場合には、現在接続中の全ての WebSocket クライアントに対してフレームを送信する。

msg:String　　送信する文字列データ

data_arr:Table [1..#max_item] of Number

送信するバイナリデータが格納されたデータ配列。
配列中の各データは1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。

- **備考**

msg 文字列または data_arr 配列に指定したバイナリデータを WebSocketフレームに格納して WebSocket クライアントに送信します。

送信対象の WebSocket クライアントは channel_or_wsid パラメータで指定します。チャンネル名を指定した時に送信対象の WebSocket クライアントが見つからなかった場合には、フレームデータは送信されずに stat に true が返ります。WebSocketID 文字列を指定したときに送信対象が見つからなかった場合には、エラーを検出して stat に false が返ります。

実際のフレームデータ送信は WebSocket クライアント接続毎に作成されたワーカースレッドで行います。このため、通信に時間が掛かるWebSocketクライアントが送信対象に含まれていた場合でも、他のクライアントへの通信に影響を及ぼしません。

通信に時間が掛かるクライアント接続では、このライブラリ関数を頻繁にコールした場合に送信データが上書きされて一部送信されないメッセージが発生する場合があります。上書きされたデータが発生した場合にはログに下記の様なメッセージが記録されます。abs_agent のコンフィギュレーションファイル(abs_agent.xml)中の "MaxWaitToPreventOverwrite" の値(デフォルト値は 20ms)を大きくすると、上書きするまでの最大待ち時間を増やすことができます。この場合、前回の送信処理が未だ完了していない通信速度の遅いクライアントへの送信依頼時にウェイトが挿入され、これが他のクライアントへの送信処理に影響するため、ライブラリ関数処理全体のパフォーマンスが悪くなる場合があります。

```
2018/12/20 09:04:53 raspberryp WebSocketWorkerThrea 0 *WARNING* SetSendDataString: previous message had not been sent.
```

- **使用例**

WEBSOCKET_DATA イベントハンドラ中に、送信されてきたデータを単に送り返すエコー機能を作成した例

```
--------------------------------------------------------------------------------
-- 送信してきたメッセージ内容をそのままクライアント側に送り返す(Echo)例
--------------------------------------------------------------------------------
if PayloadString ~= "" then
```

```
  if not websocket_emit_text(g_params["WebSocketID"],PayloadString) then error() end
end
```

文字列送信例

```
for i =1,100,1 do
  if not websocket_emit_text('全クライアント向けメッセージ count = ' .. tostring(i)) then error() end
end

if not websocket_emit_text('app1','これは /app1/xxxxx クライアント向けメッセージ') then error() end
```

バイナリ送信例

```
local tbl0 = {}
for i = 1,100,1 do
  table.insert(tbl0,i)
end
if not websocket_emit_binary('app1',tbl0) then error() end

local tbl
for i =1,50,1 do
  tbl = new_tbl(100,i)
  if not websocket_emit_binary(tbl) then error() end
end
```

26 TCP, UDPクライアント API

abs_agentのスクリプト中から、外部のアプリケーションサーバーやアプリケーションプログラムに対してネットーワーク経由でイベントデータの通知や、リクエスト処理を行いたい場合に使用します。

HTTP プロトコルで WebAPI をアクセスする場合は、”HTTPクライアント API”の章を参照して下さい。また、abs_agentの機能を WebAPI 経由でアクセスしたい場合には、”HTTPサーバー & Web API” の章を参照して下さい。

26.1 udp_send_str()

- **機能概要**

  文字列データを UDP データグラムパケットで送信する。

- **関数定義**

  stat = udp_send_str(host, port, msg *[,encode]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
host:String	送信先ホスト名もしくはIP アドレス文字列
port:Number	UDPポート番号
msg:String	送信する文字列データ
encode:Number	文字列を指定したエンコード形式で送信する。パラメータ省略時は 1 が設定される
	1: UTF-8
	2: UCS2 (Unicode文字で “¥uxxxx” エンコード表記される)

- **備考**

msg で指定された文字列を UDPデータグラムパケットで送信します。

host に空文字列 "" を指定した場合は "localhost" が送信先になります。

host には LAN内で有効なブロードキャストアドレスを指定することもできます。

- **使用例**

```
for cnt = 1,10,1 do
   stat = udp_send_str("localhost",8091,"TestMessage #" .. tostring(cnt))
   if not stat then error() end
end
```

26.2 udb_send_binary()

- **機能概要**

バイナリデータを UDP データグラムパケットで送信する。

- **関数定義**

stat = udp_send_binary(host, port, data_arr)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
host:String	送信先ホスト名もしくはIP アドレス文字列
port:Number	UDPポート番号
data_arr:Table [1..#max_item] of Number	
	送信するバイナリデータを格納した配列。
	各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。
	各データ項目に255よりも大きな数を指定した場合には 256の剰余値を使用する。

- **備考**

data_arr のバイナリデータを UDPデータグラムパケットで送信します。

host に空文字列 "" を指定した場合は "localhost" が送信先になります。

host には LAN内で有効なブロードキャストアドレスを指定することもできます。

26.3 udb_send_recv_binary()

- **機能概要**

  文字列データやバイナリデータを UDP データグラムパケットで送信した後、ホスト側からデータを受信する。

- **関数定義**

  stat, rdata_arr = udp_send_recv_binary(host, port, data_arr ,read_len [,timeout])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  host:String　　送信先ホスト名もしくはIP アドレス文字列

  port:Number　　UDPポート番号

  data_arr:Table [1..#max_item] of Number

  　　送信するバイナリデータを格納した配列。

  　　各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。

  　　各データ項目に255よりも大きな数を指定した場合には 256の剰余値を使用する。

  read_len:Number　　受信データの最大バイト数。実際に受信したバイト数は #rdata_arr になります。

  timeout:Number　　受信時のタイムアウト時間をミリセカンド[ms]で指定する。

  　　省略時は 10,000(10秒) が指定されます。

  rdata_arr:Table [1..#max_item] of Number

  　　ホストから受信したバイナリデータを格納した配列。

  　　各データ項目は1バイトに収まる 0から 255 (0x0..0xff)の整数が格納される。

- **備考**

  data_arr のバイナリデータを UDPデータグラムパケットで送信します。送信後に、ソケット・ピア(ホスト側)から最大 read_len バイト数分のデータを受信します。

  host に空文字列 "" を指定した場合は "localhost" が送信先になります。

- **使用例**

```
local echo_host = "192.168.100.50"
local echo_port = 8090
local data = {}
for x = 1,1000,1 do
  table.insert(data,x)
end
function calc_checksum(tbl)
```

```
  local checksum = 0
  for i,v in ipairs(tbl) do
    checksum = checksum + v
    checksum = bit_and(0xff,checksum)
  end
  return checksum
end
local rbuff
for i = 1,50,1 do
  log_msg("sent(" .. tostring(i) .. "): checksum=0x" .. bit_tohex(calc_checksum(data),2))
  rbuff = stat_check(udp_send_recv_binary(echo_host,echo_port,data,2000))
  log_msg("received " .. tostring(#rbuff) .. " bytes first=0x" .. bit_tohex(rbuff[1],2) .. " last=0x" ..
bit_tohex(rbuff[#rbuff],2) .. " checksum=0x" .. bit_tohex(calc_checksum(rbuff),2))
end
```

26.4 tcp_send_str()

- **機能概要**

  文字列データをTCP ストリームソケットで送信する。

- **関数定義**

  stat = tcp_send_str(host, port, data *[,*encode *[,*trailing_data *[,*wait_for_disconnect*]]])*

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
host:String	送信先ホスト名もしくはIP アドレス文字列
port:Number	TCPポート番号
data:String	送信する文字列データ
encode:Number	文字列を指定したエンコード形式で送信する。パラメータ省略時は 1 が設定される 1: UTF-8 2: UCS2 (Unicode文字で “¥uxxxx” エンコード表記される)
trailing_data:Number	文字列の終端に指定した改行コードを追加する。省略時は 2 が指定される 0: 文字列の終端に改行コードを追加しない 1: 文字列の終端に CR (0x0D) を追加する 2: 文字列の終端に LF/NL (0x0A) を追加する 3: 文字列の終端に CR と LF/NL (0x0D, 0x0A) を追加する
wait_for_disconnect:Number	TCP ストリームデータ送信後にソケットをクローズするまでの待ち時間(ms) 0 以上の整数値を指定する。省略時は 10(ms) が指定される

- **備考**

data で指定された文字列を abs_agentから指定されたホストへ TCP データストリームで送信します。
host に空文字列 "" を指定した場合は "localhost" が送信先になります。

trailing_data パラメータを指定した場合には、文字列の終端に指定した改行コードを付加してから送信されます。デフォルト値は 2 で、0x0A が文字列の終端に付加してから送信します。長い文字列を送信する場合にはデータ受信側のプログラムが終端文字コードをチェックすることで、確実にデータストリーム全体を受信することができます。

wait_for_disconnect パラメータを指定して、送信後にクライアント側のソケットをクローズするまでの待ち時間を入れることができます。送信先のホスト側プログラムによっては、この待ち時間を 0 にした場合にはうまく受信できない場合があります。このパラメータを調整して、ソケットのクローズ処理を遅らせてもデータを受信できない場合には、代わりに tcp_send_recv_str() ライブラリ関数を使用して、ホスト側からAck 文字列を送信する方法を検討してください。

- **使用例**

```
for cnt = 1,10,1 do
   stat = tcp_send_str("localhost",8091,"TestMessage #" .. tostring(cnt),1,2,10)
   if not stat then error() end
end
```

26.5 tcp_send_recv_str()

- **機能概要**

文字列データをTCP ストリームソケットで送信した後、ホスト側から文字列データを受信する。

- **関数定義**

stat, rdata = tcp_send_recv_str(host, port, data *[,*encode *[,*trailing_data*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
host:String	送信先ホスト名もしくはIP アドレス文字列
port:Number	TCPポート番号
data:String	送信する文字列データ
encode:Number	文字列を指定したエンコード形式で送信する。パラメータ省略時は 1 が設定される 1: UTF-8 2: UCS2 (Unicode文字で “¥uxxxx” エンコード表記される)
trailing_data:Number	文字列の終端に指定した改行コードを追加する。省略時は 2 が指定される tcp_send_data() のtrailing_data パラメータと違って0 は指定できません。 1: 文字列の終端に CR (0x0D) を追加する 2: 文字列の終端に LF/NL (0x0A) を追加する 3: 文字列の終端に CR と LF/NL (0x0D, 0x0A) を追加する

rdata:String　　送信先からリプライで返された文字列データ

- **備考**

data で指定された文字列を abs_agent から指定されたホストへ TCP データストリームで送信します。host に空文字列 "" を指定した場合は "localhost" が送信先になります。データ送信後に同一ソケットを使用して、ホストから送信された文字列を受信します。

trailing_data パラメータで指定した改行コードを文字列の終端に自動的に付加してから送信します。デフォルト値は 2 で、0x0A を文字列の終端に付加してから送信します。長い文字列を送信する場合にはデータ受信側のプログラムが終端文字コードをチェックすることで、確実にデータストリーム全体を受信することができます。リプライ文字列をホスト側から送信する時には、abs_agent側から送信する時にtrailing_data パラメータで指定したのと同じ終端文字を追加して送信してください。tcp_send_recv_str() 関数は終端文字が送られてくるまでのデータをリプライ文字列(rdata)に設定します。同様に、encode パラメータで指定したのと同じ文字形式でリプライ文字列を送信してください。tcp_send_recv_str() 関数はそれぞれの文字形式から自動的にデコードします。

- **使用例**

```
for cnt = 1,10,1 do
   stat,rdata = tcp_send_recv_str("localhost",8091,"TestMessage #" .. tostring(cnt),1,2)
   if not stat then error() end
end
```

26.6　tcp_send_recv_binary()

- **機能概要**

バイナリデータをストリームソケットで送信する。

送信後のリプライとしてホスト側からバイナリデータを受信することもできる。

- **関数定義**

stat *[,*rdata_hexstr*]* = tcp_send_recv_binary(host, port, data_hexstr ,read_len *[,*timeout *[,*retain*]]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

host:String　　送信先ホスト名もしくはIP アドレス文字列

port:Number　　TCPポート番号。
-1 を指定すると abs_agent コンフィギュレーションファイル中で設定された NETSERVER/TCPPort 値を使用する。デフォルトは 502に設定されている。

data_hexstr:String　　送信するバイナリデータ。16進数の文字列形式で指定する。

read_len:Number　　サーバーからリプライを受信する時のデータバイト数を指定する。
read_len に 0 を指定するとサーバーからリプライデータは受信しない。

read_len に -1 を指定すると、サーバーからリプライパケットを abs_agent コンフィギュレーションファイル中で設定された固定フレームと可変フレームを合わせたバイト数分受信する。この時、固定フレーム部分を最初に受信した後、固定フレーム内のデータから可変フレームサイズを計算して、残りの可変フレーム部分を次に受信する。その後固定フレームと可変フレームを合わせたデータを rdata_hexstr に設定する。

timeout:Number　受信時のタイムアウト時間をミリセカンド[ms]で指定する。
省略時は 500(500ミリ秒) が指定されます。

retain:Boolean　true を指定するとTCP サーバーとのソケット接続を維持したままにする。
複数回の tcp_send_recv_binary() コール時に既に接続済みのソケットを使用することで、コネクション処理を省略した高速動作が可能になる。また、サーバー側の接続負荷を低減することもできる。ソケット接続は host と port のペア毎に内部で保持されています。関数コール時に既存のソケット接続で既にエラーを検出していた場合は、そのエントリは使用されないで新しいソケット接続を作成します。データ送・受信時にソケットエラーを検出した場合には stat に false が返り、使用中のソケット接続はエラーにマークされます。この場合にはライブラリ関数利用側でリトライすることで新しいソケット接続で送受信を試みることができます。

false を指定すると、関数コール時にサーバー接続を新規に開始してデータ送受信を行った後ソケットを切断する。
省略時は true が指定されます。

rdata_hexstr:String
read_len パラメータに 0 以外を指定した場合に、サーバーから受信したリプライパケットのバイナリデータ。16進数の文字列形式で格納される。

- **備考**

data_hexstr のバイナリデータを TCPストリームパケットで送信します。

送信後に、オプションでサーバー側から read_len バイト数分のデータを受信します。

通常 TCPストリームパケットでデータ受信する場合には、予め決められたフレーム構造のパケットを受信する場合が殆どです。このとき read_len パラメータで読み込みバイト数を指定する場合は固定長フレームの場合になります。

可変長フレームを扱う場合には、予め abs_agent のコンフィギュレーション(NETSERVER)に、フレーム長計算の設定を行っておきます。read_len パラメータに -1 に指定することで、可変長フレームに対応したデータ取得が行われます。また、このコンフィギュレーション設定は、汎用 TCPサーバー機能でクライアントからのデータ受信時にも適用されます。詳しくは "サーバープログラム・設定ファイル"章中の NETSERVER 項と、"イベント"章中の TCPSERVER_DATA 項を参照してください。

retain を true に設定したときに、既存のコネクションが存在してかつ現在使用中の場合には、ライブラリ関数内で待ち状態になります。この時は、最大6秒間でタイムアウトエラーを検出します。

retain を true にした場合のクライアント接続の状態を確認したり、古い接続を削除したい場合は、tcp_client_purge() ライブラリ関数の説明や、"クライアントプログラム" 章中の "agent_net" の項を参照して下さい。

- **使用例**

```
local host = "192.168.100.18"
local port = 502
local retain_mode = true
local read_len = -1
local timeout = 500
local data = '000000000006010300180001'
local stat,rdata
for i = 1,10 do
  stat,rdata = tcp_send_recv_binary(host,port,data,read_len,timeout,retain_mode)
  if not stat then
    if retain_mode then -- retain モード時のみ１回だけリトライを行う
      rdata = stat_check(tcp_send_recv_binary(host,port,data,read_len,timeout,retain_mode))
    else
      error()
    end
  end
  if rdata then log_msg("rdata=" .. rdata) end
end
```

26.7 tcp_client_purge()

- **機能概要**

  retain モードで、接続を維持しているクライアント接続を削除する。

- **関数定義**

  stat = tcp_client_purge(*[,timestamp_until]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  timestamp_until:String　削除対象のタイムスタンプ値

  指定した日付時刻よりも古い最終アクセス時刻を持つクライアント接続を削除する。

  'YYYY/MM/DD HH:MM:SS' の形式で指定する

  パラメータ省略時は既存のクライアント接続全てを削除する

- **備考**

tcp_send_recv_binary() ライブラリ関数コール時の ratain パラメータ を true にした時に作成されたクライアント接続の中で、最後に使用された時のタイムスタンプが timestamp_until よりも古い(過去)ものを削除する。パラメータ省略時は、内部で保持している全てのクライアント接続を削除する。

retain で維持しているクライアント接続で、通信エラーを検出しているエントリは timestamp_until パラメータ指定に関係なく全て削除される。

インストールキットで提供されるスクリプト "TCPCLIENT_PURGE" では、このライブラリ関数をコールして古い接続を自動的に削除できます(下記参照)。インストール直後は "PERIODIC_TIMER" イベントハンドラから このスクリプトが１分毎にコールされていて、使用されていない古い接続を自動削除しています。

- **使用例(TCPCLIENT_PURGE スクリプト)**

```
local sec_before = 60
local val = stat_check(get_shared_data("$TCPCLIENT_PURGE_BEFORE"))
if val ~= "" then sec_before = tonumber(val) end
-- 削除対象日付を取得
local year, month, day, hour, min, sec, millisec = stat_check(now_datetime())
local new_year, new_month, new_day, new_hour, new_min, new_sec =
                            stat_check(inc_second(-1 * sec_before, year, month, day, hour, min, sec))
local timestamp =
string.format("%4.4d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d",new_year,new_month,new_day,new_hour,new_min,new_sec)
--log_msg("data purge until " .. timestamp)
-- 古い接続を削除
stat_check(tcp_client_purge(timestamp))
```

27 シリアルデバイスAPI

abs_agent の SERIAL モジュールでは、シリアルポートに接続したデバイスからのデータ受信やデバイスへのデータ送信機能を提供しています。

abs_agent の SERIAL モジュールでサポートするデータは、文字列を扱うための “STRING” タイプと、Arduinoデバイスでサポートしている “FIRMATA” タイプ、XBee モジュールの API フレームを扱う "XBEE", "ZB"等があり、専用のバイナリフォーマットを扱うことができます。もちろん、任意のフォーマットで作成されたバイナリデータを送信することもできます。

27.1 serial_all_list()

- **機能概要**

  abs_agent に登録された全てのシリアルデバイスリストを取得する。

- **関数定義**

  stat, com_port, com_title, com_type, buffered = serial_all_list()

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  com_port:Table [1..#max_device] of String　　シリアルデバイスのポート名 (“COMxx”)

  com_title:Table [1..#max_device] of String　　シリアルデバイスのタイトル名

  　　タイトルを登録していない場合には空文字列 “” が入る

  com_type:Table [1..#max_device] of String　　シリアルデバイスのタイプ名

  buffered:Table [1..#max_device] of Boolean　　シリアルデバイスがバッファーモードかどうかを示す

- **備考**

  serial_all_list() は abs_agentに登録されている全てのシリアルデバイスのリストを取得します。

  シリアルデバイスの登録や削除・修正を行うときは、”サーバー設定ファイル” 直接編集して下さい。

- **使用例**

```
file_id = "SERIAL_DEV_LIST"
local stat,com,title,type,buff = serial_all_list()
if not stat then error() end
for key,val in ipairs(com) do
  log_msg(string.format("device[%d] com = %s title = %s type = %s buffered  = %s",
                        key,val,title[key],type[key],tostring(buff[key])),file_id)
end
```

27.2 serial_find_device()

- **機能概要**

  SERIAL サービスに登録済みのデバイスタイトル名またはポート名を指定してCOMポート名を取得する

- **関数定義**

  stat, com_port = serial_find_device(com_title)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  com_title:String　　SERIAL サービスに登録済みのデバイスタイトル名。COMポート名 (“/dev/ttyxxx”)

  　　をパラメータに指定することもできる。

  com_port:String　　SERIAL サービスに登録済みのCOMポート名 (“/dev/ttyxxxx”)

- **備考**

com_title で指定するシリアルデバイスのタイトル名は、予め SERIAL サービスモジュールに登録しておく必要があります。

com_title に指定したシリアルデバイスが見つかった場合には、stat は true に設定されて com_port には COMポート名("/dev/ttyxxxxx") が設定されます。見つからなかった場合には stat に false が返ります。

- **使用例**

```
local stat,port = serial_find_device("Arduinoデバイス#1")
```

27.3 serial_readln()

- **機能概要**

シリアルポートから受信したデータが格納された文字列バッファから文字列を取り出す。

- **関数定義**

stat, value, remain = serial_readln(com_port)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
value:String	文字列バッファから取り出した先頭の文字列
remain:Number	文字列を取り出した後の、残りの文字列バッファに格納されている文字列数 残りの文字列バッファが空の場合には 0 が設定される。
com_port:String	SERIAL サービスに登録済みのポート名 (“COMxx”) シリアルデバイスのタイトルが設定されている場合には、タイトル文字列を com_port パラメータに指定することもできる。

- **備考**

com_port で指定するシリアルデバイスのポート名は、予め SERIAL サービスモジュールに登録しておく必要があります。また、シリアルデバイスは BufferedMode が True に設定されている必要があります。"XBEE" と "ZB" デバイスタイプは BufferdMode が常に False に設定されるのでこのライブラリ関数を使用できません。

文字列バッファに複数の文字列が格納されていた場合には、最初の文字列が取り出されて文字列バッファからは削除されます。

文字列バッファに格納する文字列数がサーバー設定ファイル中の MaxBufferRecordCount(デフォルト値:100) を超えた場合には、最初に登録したデータから順に自動削除されています。そのため、文字列バッファからは常に MaxBufferRecordCount 分の直近データを取り出すことができます。

文字列バッファが空の場合には、エラーとなって stat には false が返ります。この関数をコールする前に serial_available() 関数を使用して、予め文字列バッファに格納されている文字列数を取得することができま

す。

文字列バッファに格納されている内容はデバイスタイプ毎に下記の様に決まっています。

デバイスタイプ	文字列バッファに格納される内容
STRING	シリアルポートから入力した１文字列が格納されています。文字列は NULL (0x00) または改行文字ごとに区切られてから１文字列づつバッファに格納されています。
FIRMATA	FIRMATA プロトコルで定義された１パケット分のバイナリデータを16進数文字列に変換したものが格納されています。1つの FIRMATA パケット毎にバッファにまとめて格納されます。
RAW	シリアルポートから読み込んだバイナリデータを16進数文字列に変換したものが格納されています。シリアルドライバとライブラリ経由で取得したバイナリデータの固まりをそのままのバッファにまとめて格納されます。
XBEE	使用できません
ZB	使用できません

- **使用例**

```
local com = "/dev/ttyUSB0"
-----------------------------------------------------------------
-- COM ポートから受信した文字列バッファの内容を出力する
-----------------------------------------------------------------
local stat,count,val
stat,count = serial_available(com)
if not stat then error() end
while count > 0 do
  stat,val,count = serial_readln(com)
  if not stat then error() end
  log_msg("read string = " .. val)
end
```

```
file_id = "SERIAL_AR8200_RX"
------------------------------------------------------------------
-- AOR AR8200 レシーバーで地元ラジオ局を受信する
-- AR8200 とPCシリアルポートの接続には AOR 製 CC8200 ケーブルを使用する
-- 予め本体のスケルチとボリューム位置を適当な値に設定しておくこと
-- 接続する COM ポートのコンフィギュレーションは下記の通り
--      <COMPort>           COMxx (実際に使用する COM ポート名を指定する)
--      <Type>                      STRING
--      <BufferedMode>      True
--      <BaudRate>          9600  (AR8200 本体の CONFIG から同じボーレートを設定しておく)
--      <DataBits>          8
--      <StopBits>          2
--      <ParityBit>                 NONE
--      <FlowControl>       SOFT  (XON/XOFF)
------------------------------------------------------------------


------------------------------------------------------------------
-- シリアルポートから１文字列を取得する関数 serial_input() の定義
-- read_string = serial_input(com_port)
------------------------------------------------------------------
function serial_input(com_port)
        ------------------------------------
        -- waiting for serial data
        ------------------------------------
        local stat,count,val,max_wait;
        local max_wait = 300; -- about 4..5 seconds
        repeat
                  wait_time(10);
                  max_wait = max_wait - 1;
                  stat,count = serial_available(com_port);
                  if not stat then error() end;
        until (count > 0) or (max_wait < 0);
        ------------------------------------
        -- get a string from the buffer
        ------------------------------------
        if max_wait >= 0 then
                  stat,val,count = serial_readln(com_port);
                  if not stat then error() end;
                  return    val;
```

```
        else
                log_msg("serial_input:*ERROR* max_wait exceeded");
                error();
        end;
end;

local com = "COM1"
-- スケルチオープン時のレスポンス抑止
if not serial_print(com,"LC0",1) then error() end;
serial_input(com);
-----------------------------------------------------------------
-- STVラジオ AM,1440kHz
-----------------------------------------------------------------
if not serial_print(com,"RF1.440",1) then error() end;
serial_input(com);
if not serial_print(com,"MD7",1) then error() end;
serial_input(com);
if not serial_print(com,"RX",1) then error() end;
log_msg("AR8200 RX:" .. serial_input(com));
-- 5 秒視聴 --
wait_time(5000);
-----------------------------------------------------------------
-- HBCラジオ AM,1287kHz
-----------------------------------------------------------------
if not serial_print(com,"RF1.287",1) then error() end;
serial_input(com);
if not serial_print(com,"MD7",1) then error() end;
serial_input(com);
if not serial_print(com,"RX",1) then error() end;
log_msg("AR8200 RX:" .. serial_input(com));
-- 5 秒視聴 --
wait_time(5000);
-----------------------------------------------------------------
-- エフエム北海道 FM,80.4MHz
-----------------------------------------------------------------
if not serial_print(com,"RF80.4",1) then error() end;
serial_input(com);
if not serial_print(com,"MD0",1) then error() end;
serial_input(com);
```

```
if not serial_print(com,"RX",1) then error() end;
log_msg("AR8200 RX:" .. serial_input(com));
-- 5 秒視聴 --
wait_time(5000);
------------------------------------------------------------
-- エフエム・ノースウエーブ FM,80.4MHz
------------------------------------------------------------
if not serial_print(com,"RF82.5",1) then error() end;
serial_input(com);
if not serial_print(com,"MD0",1) then error() end;
serial_input(com);
if not serial_print(com,"RX",1) then error() end;
log_msg("AR8200 RX:" .. serial_input(com));
-- 5 秒視聴 --
wait_time(5000);
------------------------------------------------------------
-- AR8200リモートコントロールモード終了
------------------------------------------------------------
if not serial_print(com,"EX",1) then error() end
```

27.4 serial_available()

- **機能概要**

  シリアルポートから受信した文字列フレームバッファの保存レコード数を取得したり、フレームバッファをクリアする。

- **関数定義**

  stat, count = serial_available(com_port *[,clear]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
count:Number	文字列バッファに格納されている文字列数 文字列バッファが空の場合には 0 が設定される。
com_port:String	SERIAL サービスに登録済みのポート名 (“COMxx”) シリアルデバイスのタイトルが設定されている場合には、タイトル文字列を com_port パラメータに指定することもできる。
clear:Boolean	バッファ中に保存されている文字列を全て消去する。true に指定した場合は count の値は常に 0 が返る。省略時は false が指定される

- **備考**

  com_port で指定するシリアルデバイスのポート名は、予め SERIAL サービスモジュールに登録しておく必要が

あります。また、シリアルデバイスはBufferedMode が True に設定されている必要があります。"XBEE" と "ZB" デバイスタイプは BufferdMode が常に False に設定されるのでこのライブラリ関数を使用できません。

count パラメータには、文字列バッファに格納中の文字列数が入ります。例えばcount が 2 の場合には、serial_readln() 関数を２回使用して、文字列を合計２つ取得することができます。

文字列バッファに格納する文字列数が、サーバー設定ファイル中の MaxBufferRecordCount(デフォルト値 100) を超えた場合には、最初に登録したデータから順に自動削除されています。このため、このライブラリ関数で得られる count の値は常に MaxBufferRecordCount 以下になります。

- **使用例**

```
local com = "/dev/ttyUSB0"
------------------------------------------------------------
-- COM ポートから受信した文字列バッファの内容を出力する
------------------------------------------------------------
local stat,count,val
stat,count = serial_available(com)
if not stat then error() end
while count > 0 do
  stat,val,count = serial_readln(com)
  if not stat then error() end
  log_msg("read string = " .. val)
end
```

27.5 serial_print()

- **機能概要**

シリアルポートに文字列を送信する。

- **関数定義**

stat = serial_print(com_port, str *[,trailing_data]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
com_port:String	SERIAL サービスに登録済みのポート名 (“COMxx”) シリアルデバイスのタイトルが設定されている場合には、タイトル文字列を com_port パラメータに指定することもできる。
str:String	シリアルポートから送信する文字列
trailing_data:Number	文字列の終端に指定した終端コードを追加してから送信する。 省略時は 0 が指定される 0: 文字列の終端に終端コードを追加しない 1: 文字列の終端に CR (0x0D) を追加する

2: 文字列の終端に LF/NL (0x0A) を追加する

3: 文字列の終端に CR と LF/NL (0x0D, 0x0A) を追加する

4: 文字列の終端に Null (0x00) を追加する

- **備考**

com_port で指定するシリアルデバイスのポート名は、予め SERIAL サービスモジュールに登録しておく必要があります。この関数は シリアルデバイスのタイプや BufferedMode に関係なく使用できます。

送信するデータは、文字列のみが指定できます。日本語の文字データは、UTF-8 コードで送信されます。
バイナリデータを送信する場合には、serial_write() ライブラリ関数を使用してください。

trailing_data パラメータを指定した場合には、文字列の終端に指定した終端コードを付加してから送信します。

- **使用例**

```
if not serial_print("/dev/ttyUSB0","Hello World!!",4) then error() end
```

27.6 serial_command()

- **機能概要**

シリアルポートに文字列を送信した後、同一ポートからリプライ文字列を受信する。

- **関数定義**

stat, reply_str = serial_command(com_port, request_str *[,trailing_data]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
reply_str:String	シリアルポートから受信した文字列
com_port:String	SERIAL サービスに登録済みのポート名 ("COMxx") デバイスタイプは "STRING" に設定すること。 シリアルデバイスのタイトルが設定されている場合には、タイトル文字列を com_port パラメータに指定することもできる。
request_str:String	シリアルポートから送信する文字列
trailing_data:Number	文字列の終端に指定した改行コードを追加する。省略時は 1 が指定される 0: 文字列の終端に改行コードを追加しない 1: 文字列の終端に CR (0x0D) を追加する 2: 文字列の終端に LF/NL (0x0A) を追加する 3: 文字列の終端に CR と LF/NL (0x0D, 0x0A) を追加する

- **備考**

com_port で指定するシリアルデバイスのポート名は、予め SERIAL サービスモジュールに登録しておく必要が

あります。この関数は シリアルデバイスタイプが "STRING" のものにだけに使用することができます。

このライブラリ関数を使用すると、コマンド文字列送信とリプライ文字列受信を１度に実行できます。BufferedMode を使用しないときには、シリアルデバイスから任意のタイミングで送信される文字列データはイベントハンドラスクリプト(SERIAL_STRING) で処理できます。これによってコマンドとレスポンスの処理は serial_command() ライブラリ関数で行い、その他のイベント処理は SERIAL_STRING.lua イベントハンドラに記述してシステム全体の処理を簡略化できます。

trailing_data パラメータに指定した改行コードを、コマンド文字列の終端に付加して送信します。

BufferedMode を有効にすると、この関数のリターンパラメータ reply_string に返されるリプライ文字列と、シリアルデバイスから任意のタイミングで送信される文字列データを serial_readln() ライブラリ関数で読み込むことができます。

コマンド送信とリプライ受信を別々に実行したい場合には、このライブラリ関数の代わりに serial_print() と serial_readln() を組み合わせて使用するか(BufferedMode が有効の場合)、SERIAL_STRING イベントハンドラスクリプト(BufferedMode が無効の場合)を使用してリプライデータを取得できます。

送信するデータは、文字列のみが指定できます。日本語の文字データは、UTF-8 コードで送信されます。また、リプライで受信するデータも文字列形式のみ使用できます。

リプライで受信する文字列データはコマンド文字列送信後に、同一ポートから取得した最初の文字列データです。もし、複数のリプライ文字列を受信する場合には続けて serial_readln() を使用するか(BufferedMode が有効の場合) または、SERIAL_STRING イベントハンドラスクリプト(BufferedMode が無効の場合)を使用して続きのリプライデータを取得して下さい。

コマンド文字列を送信してからリプライ文字列受信までの最大待ち時間は 5 秒です。これを超えた場合にはタイムアウトエラーが発生して stat には false が返ります。

- **使用例**

```
local stat,reply = serial_command(“/dev/ttyUSB0”,”cmd param1 param2”)
if stat then
  log_msg("result = " .. reply,"COMMAND_TEST")
else
  error()
end
```

27.7 serial_write()

- **機能概要**

パラメータで指定したシリアルポートにバイナリデータを送信する。

デバイスタイプが "FIRMATA" の場合にはデータ送信後に、FIRMATA プロトコルで定義された sysex タイプのリプライデータパケットを受信するまで内部でウェイトした後、リプライデータを取得する。

デバイスタイプが "XBEE" または "ZB" の場合には XBee API コマンドフレーム送信後に、リプライフレームを受信するまで内部でウェイトした後その結果を取得する。

- **関数定義**

(デバイスタイプが "STRING", "RAW" の場合)

stat = serial_write(com_port, hexstr)

(デバイスタイプが "FIRMATA" の場合)

stat *[,*sysex_reply_hexstr*]* = serial_write(com_port, hexstr *[,*sysex_reply_command_hexstr*]*)

(デバイスタイプが "XBEE" の場合)

stat, frame_data *[,*mframe_data_arr*]* = serial_write(com_port, hexstr)

(デバイスタイプが "ZB" の場合)

stat, frame_data = serial_write(com_port, hexstr *[,*ignore_timeout*]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

com_port:String　　SERIAL サービスに登録済みのポート名 (“COMxx”)
シリアルデバイスのタイトルが設定されている場合には、タイトル文字列を com_port パラメータに指定することもできる。

hexstr:String　　シリアルポートから送信するバイナリデータ。16進数の文字列形式で指定する。

sysex_reply_hexstr:String
sysex_reply_command_byte パラメータを指定した場合に、"FIRMATA" プロトコルでシリアルポートから受信したリプライパケットのバイナリデータ。16進数の文字列形式で格納される。

sysex_reply_command_hexstr:String
data で指定するバイトデータが "FIRMATA" の “sysex” タイプのリクエストパケットの場合に、このパラメータで指定した “sysex command” バイトに一致するリプライパケットを受信する。FIRMATA デバイスタイプのデバイスのみ有効なパラメータ。16進数の文字列形式で指定する。

frame_data:String　　XBee 802.15.4 API または XBee-ZB プロトコルでシリアルポートから受信したリプ

ライフレームのバイナリデータ。16進数の文字列形式で格納される。XBee API コマドデータフレームを hexstr に格納した場合に、XBee モジュールから返されたリプライフレームデータが設定されます。

mframe_data_arr:Table [1..#max_multiframe] of String

"XBEE" デバイスタイプで hexstr にマルチフレームを返す AT コマンドフレームを設定した場合に、XBee 802.15.4 モジュールから返されたマルチフレーム部分のータが 16進数文字列配列で格納されます。コマンド実行結果ステータスが格納されたフレームデータ部は frame_dataに設定されます。

ignore_timeout:Boolean

"ZB" デバイスタイプの場合に、10 秒以内にリプライフレームを受信できなかった場合にはタイムアウトを検出して stat には false が返ります。ignore_timeout を true に指定すると、タイムアウトが発生した場合でも stat には true が返りエラーメッセージもログに出力しません。マルチフレームを返すコマンドを送信する場合には常にステータスフレームは返されないので、このパラメータを true に設定して、SERIAL_ZB イベントハンドラ側でマルチフレームを受信します。省略時は false が設定されます。

- **備考**

com_port で指定するシリアルデバイスのポート名は、予め SERIAL サービスモジュールに登録しておく必要があります。

hexstr で指定した 16進数形式の文字列データをバイナリ値に変換した後、シリアルポートから送信します。

"FIRMATA" デバイスタイプのデバイスに送信する場合には、FIRMATA プロトコルで定義されたデータフォーマットに従って、送信データを作成してください。

"FIRMATA" デバイスに対して、Query -> Reply タイプのパケットを送・受信する場合には、FIRMATA(sysex) Query パケットを送信するときに、sysex_reply_command_hexstrパラメータを指定します。これはリプライで受信予定の sysex パケットの “sysex command” バイト値を指定することで、この値に一致した FIRMATA (sysex) プロトコルのパケットを受信するまで関数内部でウェイトします。FIRMATA(sysex) リプライパケットを受信したら、その内容が sysex_reply_hexstr リターン値に設定されます。ウェイトの最大待ち時間は 5 秒で、これを超えた場合にはタイムアウトエラーが発生して stat には false が返ります。

"XBEE" や "ZB" デバイスタイプに設定したシリアルポートにデータを送信する場合には、XBee 802.15.4 API Mode1 または XBee-ZB API Mode1 で定義されたコマンドデータフレームのフォーマットに従って、hexstr データを作成します。このとき、hexstr にセットするコマンドデータフレーム中の FrameID 部分のバイト値は必ず 0x00 以外を指定する必要があります。

XBEE, ZB デバイス操作について

XBee 802.15.4 や XBee-ZB Zigbee モジュールに対して ATコマンド実行やパケット送信等を行う時には、serial_write() ライブラリ関数よりも便利な xbee_at_command(),zb_at_command(),xbee_transmit_data(), zb_transmit_data() を使用してください。

これらの専用ライブラリ関数ではリクエストフレームデータの構築やコマンドパラメータ取得、リモート側デバイスのアドレス変換等の処理を自動で行います。これらのライブラリ関数も内部では serial_write() をコールしています。詳しくは "XBee 802.15.4 API" や "XBee-ZB API" の章を参照して下さい。

- **使用例**

FIRMATA フレーム送信例

```
if not serial_write("/dev/ttyUSB0","414243") then error() end


-- Query firmware name and version (FIRMATA protocol)
local stat,reply = serial_write("COM1","F079F7","79")
if not stat then error() end;
log_msg("reply = " .. reply)
```

XBee コマンドフレーム送信例

```
-----------------------------------------------------------------
-- COM ポートから XBee API コマンドフレームを送信する
-----------------------------------------------------------------
local com = "/dev/ttyUSB0"
function debug_frame_dump(stat,frame,mframe)
  if not stat then error() end
  log_msg("-- frame --",g_script)
  log_hexdump(frame)
  if mframe then
    for k,v in ipairs(mframe) do
      log_msg("-- mframe#" .. k .. " --",g_script)
      log_hexdump(v)
    end
  end
end

-- Xbee AT "NI" command
debug_frame_dump(serial_write(com,"7E000408524E490E"))

-- Xbee AT "ND" command
```

```
debug_frame_dump(serial_write(com,"7E000408524E4413"))


-- Xbee RemoteAT "NI" command
debug_frame_dump(serial_write(com,"7E000F17520013A200404AC397FFFE024E4967"))


-- Transmit 16bit addr (dest:0C03)
debug_frame_dump(serial_write(com,"7E000D01520C030011223344AABBCCDDE5"))


-- Transmit 64bit addr (dest:0013A200404AC398)
debug_frame_dump(serial_write(com,"7E001300520013A200404AC3980011223344AABBCCDD5B"))
```

上記　XBee コマンドフレーム送信時のログ出力例

```
2019/05/11 08:35:52 raspberryp TEST/XBEE_WRITE      0 -- frame --
2019/05/11 08:35:52 raspberryp LUA Script           0  length = 16 bytes
2019/05/11 08:35:52 raspberryp LUA Script           0
2019/05/11 08:35:52 raspberryp LUA Script           0  data[000]-[007] 7E 00 0C 88 52 4E 49 00
2019/05/11 08:35:52 raspberryp LUA Script           0                  ~          R  N  I
2019/05/11 08:35:52 raspberryp LUA Script           0  data[008]-[00F] 44 65 76 69 63 65 35 09
2019/05/11 08:35:52 raspberryp LUA Script           0                  D  e  v  i  c  e  5
2019/05/11 08:35:55 raspberryp TEST/XBEE_WRITE      0 -- frame --
2019/05/11 08:35:55 raspberryp LUA Script           0  length = 9 bytes
2019/05/11 08:35:55 raspberryp LUA Script           0
2019/05/11 08:35:55 raspberryp LUA Script           0  data[000]-[007] 7E 00 05 88 52 4E 44 00
2019/05/11 08:35:55 raspberryp LUA Script           0                  ~          R  N  D
2019/05/11 08:35:55 raspberryp LUA Script           0  data[008]-[008] 93
2019/05/11 08:35:55 raspberryp LUA Script           0
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE      0 -- mframe#1 --
2019/05/11 08:35:56 raspberryp LUA Script           0  length = 28 bytes
2019/05/11 08:35:56 raspberryp LUA Script           0
2019/05/11 08:35:56 raspberryp LUA Script           0  data[000]-[007] 7E 00 18 88 52 4E 44 00
2019/05/11 08:35:56 raspberryp LUA Script           0                  ~          R  N  D
2019/05/11 08:35:56 raspberryp LUA Script           0  data[008]-[00F] 0B 02 00 13 A2 00 40 4A
2019/05/11 08:35:56 raspberryp LUA Script           0                                    @  J
2019/05/11 08:35:56 raspberryp LUA Script           0  data[010]-[017] C3 97 3C 44 65 76 69 63
2019/05/11 08:35:56 raspberryp LUA Script           0                        <  D  e  v  i  c
2019/05/11 08:35:56 raspberryp LUA Script           0  data[018]-[01B] 65 32 00 2F
2019/05/11 08:35:56 raspberryp LUA Script           0                  e  2     /
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE      0 -- mframe#2 --
2019/05/11 08:35:56 raspberryp LUA Script           0  length = 28 bytes
```

```
2019/05/11 08:35:56 raspberryp LUA Script          0
2019/05/11 08:35:56 raspberryp LUA Script          0  data[000]-[007] 7E 00 18 88 52 4E 44 00
2019/05/11 08:35:56 raspberryp LUA Script          0                  ~          R  N  D
2019/05/11 08:35:56 raspberryp LUA Script          0  data[008]-[00F] 0D 04 00 13 A2 00 40 55
2019/05/11 08:35:56 raspberryp LUA Script          0                                  @  U
2019/05/11 08:35:56 raspberryp LUA Script          0  data[010]-[017] 80 26 42 44 65 76 69 63
2019/05/11 08:35:56 raspberryp LUA Script          0                     &  B  D  e  v  i  c
2019/05/11 08:35:56 raspberryp LUA Script          0  data[018]-[01B] 65 34 00 CC
2019/05/11 08:35:56 raspberryp LUA Script          0                  e  4
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE     0 -- mframe#3 --
2019/05/11 08:35:56 raspberryp LUA Script          0  length = 28 bytes
2019/05/11 08:35:56 raspberryp LUA Script          0
2019/05/11 08:35:56 raspberryp LUA Script          0  data[000]-[007] 7E 00 18 88 52 4E 44 00
2019/05/11 08:35:56 raspberryp LUA Script          0                  ~          R  N  D
2019/05/11 08:35:56 raspberryp LUA Script          0  data[008]-[00F] 0C 03 00 13 A2 00 40 4A
2019/05/11 08:35:56 raspberryp LUA Script          0                                  @  J
2019/05/11 08:35:56 raspberryp LUA Script          0  data[010]-[017] C3 98 42 44 65 76 69 63
2019/05/11 08:35:56 raspberryp LUA Script          0                        B  D  e  v  i  c
2019/05/11 08:35:56 raspberryp LUA Script          0  data[018]-[01B] 65 33 00 25
2019/05/11 08:35:56 raspberryp LUA Script          0                  e  3     %
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE     0 -- mframe#4 --
2019/05/11 08:35:56 raspberryp LUA Script          0  length = 28 bytes
2019/05/11 08:35:56 raspberryp LUA Script          0
2019/05/11 08:35:56 raspberryp LUA Script          0  data[000]-[007] 7E 00 18 88 52 4E 44 00
2019/05/11 08:35:56 raspberryp LUA Script          0                  ~          R  N  D
2019/05/11 08:35:56 raspberryp LUA Script          0  data[008]-[00F] 0A 01 00 13 A2 00 40 4A
2019/05/11 08:35:56 raspberryp LUA Script          0                                  @  J
2019/05/11 08:35:56 raspberryp LUA Script          0  data[010]-[017] C3 9C 3E 44 65 76 69 63
2019/05/11 08:35:56 raspberryp LUA Script          0                        >  D  e  v  i  c
2019/05/11 08:35:56 raspberryp LUA Script          0  data[018]-[01B] 65 31 00 2B
2019/05/11 08:35:56 raspberryp LUA Script          0                  e  1     +
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE     0 -- frame --
2019/05/11 08:35:56 raspberryp LUA Script          0  length = 26 bytes
2019/05/11 08:35:56 raspberryp LUA Script          0
2019/05/11 08:35:56 raspberryp LUA Script          0  data[000]-[007] 7E 00 16 97 52 00 13 A2
2019/05/11 08:35:56 raspberryp LUA Script          0                  ~          R
2019/05/11 08:35:56 raspberryp LUA Script          0  data[008]-[00F] 00 40 4A C3 97 0B 02 4E
2019/05/11 08:35:56 raspberryp LUA Script          0                     @  J             N
2019/05/11 08:35:56 raspberryp LUA Script          0  data[010]-[017] 49 00 44 65 76 69 63 65
```

```
2019/05/11 08:35:56 raspberryp LUA Script          0                 I    D e v i c e
2019/05/11 08:35:56 raspberryp LUA Script          0  data[018]-[019] 32 57
2019/05/11 08:35:56 raspberryp LUA Script          0                 2  W
2019/05/11 08:35:56 raspberryp TEST/XBEE_WRITE     0 -- frame --
2019/05/11 08:35:56 raspberryp LUA Script          0  length = 7 bytes
2019/05/11 08:35:56 raspberryp LUA Script          0
2019/05/11 08:35:56 raspberryp LUA Script          0  data[000]-[006] 7E 00 03 89 52 00 24
2019/05/11 08:35:56 raspberryp LUA Script          0                 ~          R     $
2019/05/11 08:35:57 raspberryp TEST/XBEE_WRITE     0 -- frame --
2019/05/11 08:35:57 raspberryp LUA Script          0  length = 7 bytes
2019/05/11 08:35:57 raspberryp LUA Script          0
2019/05/11 08:35:57 raspberryp LUA Script          0  data[000]-[006] 7E 00 03 89 52 00 24
2019/05/11 08:35:57 raspberryp LUA Script          0                 ~          R     $
```

27.8 firmata_str_encode()

- **機能概要**

  文字列を FIRMATA のバイナリ形式に変換する。

- **関数定義**

  stat, hexdata = firmata_str_encode(str)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
str:String	変換対象の文字列。
hexstr:String	FIRMATA プロトコルで使用される16進数表記の文字列

- **備考**

  パラメータで指定された文字列を、１文字づつ FIRMATA プロトコルで定義された、１つの数値を 7bit幅の 2 バイトデータ(LSB + MSB)として表現するバイナリ形式に変換します。

  日本語などのマルチバイト文字を変換する場合には、UTF-8 エンコート形式で表現された数値を1 バイト毎に FIRMATAバイナリ形式に変換します。

- **使用例**

```
log_msg("start..",file_id)
local com = "/dev/ttyUSB0"
------------------------------------------------------------
-- FIRMATA エンコード試験
------------------------------------------------------------
local stat,val,val2
```

```
stat,val = firmata_str_encode("A")
if not stat then error() end
log_msg("encoded = " .. val,file_id)

stat,val2 = firmata_str_decode(val)
if not stat then error() end
log_msg("decoded = " .. val2,file_id)

stat,val = firmata_str_encode("HelloWorld")
if not stat then error() end
log_msg("encoded = " .. val,file_id)

stat,val2 = firmata_str_decode(val)
if not stat then error() end
log_msg("decoded = " .. val2,file_id)

stat,val = firmata_str_encode("これは試験メッセージです")
if not stat then error() end
log_msg("encoded = " .. val,file_id)

stat,val2 = firmata_str_decode(val)
if not stat then error() end
log_msg("decoded = " .. val2,file_id)
```

実行結果(ログ出力)

```
start..
encoded = 4100
decoded = A
encoded = 480065006C006C006F0057006F0072006C006400
decoded = HelloWorld
encoded = 6301010113016301020100C01630101012F016801290126016901280113016301030121016301...(省略)
decoded = これは試験メッセージです
```

27.9 firmata_str_decode()

- **機能概要**

  FIRMATA のバイナリ形式を文字列に変換する。

- **関数定義**

  stat, str = firmata_str_decode(hexdata)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

hexstr:String　　変換対象の FIRMATA バイナリ形式 16進数表記の文字列

str:String　　変換後の文字列。

- **備考**

この関数では FIRMATA プロトコルで定義された、7bit幅の 2バイトデータ(LSB + MSB)を１つの数値に変換した後、各々の数値を文字として連結した文字列に変換します。

- **使用例**

firmata_str_encode() の項を参照して下さい

27.10　hex72_to_tbl()

- **機能概要**

FIRMATA プロトコルで使用される、7ビット幅の複数バイト列の16進数表記文字列を数値配列に変換する。

- **関数定義**

value = hex72_to_tbl(hexstr *[,digits]*)

- **パラメータとリターン値**

hexstr:String　　16進数表記の文字列

value:Table [1..#max_item] of Number

　　取得した数値データ

digits:Number　　数値を7 ビット幅のバイトで表現する時のバイト数。

　　1 から 5 までの整数(省略時は 2 が指定される)

- **備考**

16進数文字列は左から４文字(digits＝2 の場合)ごとに区切って数値に変換した後、数値配列に設定されます。

この関数では１つの数値を 7bit幅の 2バイトデータ(LSB + MSB)として表現するFIRMATA 16進数文字列形式から、数値に変換しています。(digits = 2の場合)

digits の値を指定することで、数値を下記の様な複数バイトに分けて変換できます。

digits	Byte#	各Byte# に対応する数値のビット位置	扱える数値の最大値
1	#1	bits 0 – 6	127 (0x7F)
2	#1(LSB)	bits 0 – 6	16383 (0x3FFF)
	#2(MSB)	bits 7 - 13	
3	#1(LSB)	bits 0 – 6	2097151 (0x1FFFFF)
	#2	bits 7 - 13	

	#3(MSB)	bits 14 - 20	
4	#1(LSB)	bits 0 – 6	268435455 (0xFFFFFFF)
	#2	bits 7 - 13	
	#3	bits 14 - 20	
	#4(MSB)	bits 21 - 27	
5	#1(LSB)	bits 0 – 6	2147483647 (0x7FFFFFFF) (bit31 – 34 は 常に 0 になります。スクリプトで扱える整数が 0 から 0x7FFFFFFF の為)
	#2	bits 7 - 13	
	#3	bits 14 - 20	
	#4	bits 21 - 27	
	#5(MSB)	bits 28 – 34	

- **使用例**

```
data = "010002000300"
tbl = hex72_to_tbl(data)
for key,val in ipairs(tbl) do
   log_msg(string.format("tbl[%d] = %d %2.2X",key,val,val))
   wait_time(10)
end
```

実行結果(ログ出力)

```
tbl[1] = 1 01
tbl[2] = 2 02
tbl[3] = 3 03
```

27.11 tbl_to_hex72()

- **機能概要**

数値配列をFIRMATA プロトコルで使用される16進数表記の文字列に変換する。

- **関数定義**

hexstr = tbl_to_hex72(value [,digits])

- **パラメータとリターン値**

value:Table [1..#max_item] of Number　　変換対象の数値データ配列

hexstr:String　　16進数表記の文字列

digits:Number　　数値を7 ビット幅のバイトで表現する時のバイト数。
1 から 5 までの整数(省略時は 2 が指定される)

- **備考**

パラメータで指定した配列中の各々の数値は、2 バイトデータとして 4 文字の16進数文字列(LSB + MSB) に変換された後、16 進数表記の文字列として連結された文字列に変換します。(digits = 2 の場合)

digits の値を指定することで、数値を下記の様な複数バイトに分けて変換できます。

digits	Byte#	各Byte# に対応する数値のビット位置	扱える数値の最大値
1	#1	bits 0 – 6	127 (0x7F)
2	#1(LSB)	bits 0 – 6	16383 (0x3FFF)
	#2(MSB)	bits 7 - 13	
3	#1(LSB)	bits 0 – 6	2097151 (0x1FFFFF)
	#2	bits 7 - 13	
	#3(MSB)	bits 14 - 20	
4	#1(LSB)	bits 0 – 6	268435455 (0xFFFFFFF)
	#2	bits 7 - 13	
	#3	bits 14 - 20	
	#4(MSB)	bits 21 - 27	
5	#1(LSB)	bits 0 – 6	2147483647 (0x7FFFFFFF) (bit31 – 34 は 常に 0 になります。スクリプトで扱える整数が 0 から 0x7FFFFFFF の為)
	#2	bits 7 - 13	
	#3	bits 14 - 20	
	#4	bits 21 - 27	
	#5(MSB)	bits 28 – 34	

- **使用例**

```
local hexdata = tbl_to_hex72({1,2,3,1000,2000,3000})
log_msg("hexstr = " .. hexdata)
```

実行結果(ログ出力)

```
hexstr = 0100020003006807500F3817
```

28 MQTT クライアントAPI

abs_agent の MQTT モジュールでは、MQTT ブローカとの接続を管理するための機能を提供しています。

サーバー設定ファイルにエンドポイントを登録すると、サーバー起動時に自動的に MQTT ブローカーに接続して、購読対象になっているトピックを受信することができます。

また、MQTT ブローカに対して PUBLISH メッセージを送信するためのライブラリ関数も提供しています。トピック名とペイロードデータ、QoS (MQTT 送達品質レベル) を指定してスクリプト中から任意のタイミングでメッセージを送信できます。

エンドポイント登録時に自動で購読対象にしたトピックは、この章で提供するライブラリ関数をサーバー起動時に使用しています。ユーザーが任意のタイミングでこれらのライブラリ関数を使用してトピックの購読や購読停止を指示することもできます。

ライブラリ関数を使用して、エンドポイントでソケット通信エラーが発生しているかどうかを任意のタイミングで確認することができます。また、エラーが発生したエンドポイントについて、MQTT ブローカとの再接続を試みるためのライブラリ関数も提供しています。

MQTT クライアントライブラリ関数を使用する前に abs_agent 側に MQTTエンドポイントを作成する必要があります。abs_agent の MQTT エンドポイントの設定例については "MQTTブローカ接続設定" の章を参照してください。また、各エンドポイントの設定項目については "サーバー設定ファイル(abs_agent.xml)" 章中の MQTT 項を参照してください。

28.1 mqtt_all_list()

- **機能概要**

  abs_agent に登録された全てのMQTTクライアント・エンドポイントリストを取得する。

- **関数定義**

```
stat, enable, ready, title, id, host, port, subscribe_topic_list, subscribe_qos_list,
  user, password, will_topic, will_message, will_qos, will_retain, recv_buff_size,
  log = mqtt_all_list()
```

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

enable:Table [1..#max_endpoint] of Boolean
　　エンドポイントが作成済みであるかどうかを示す。エンドポイントを作成する時は mqtt_open() ライブラリ関数を使用する。

ready:Table [1..#max_endpoint] of Boolean
　　エンドポイントのソケット通信が利用可能かどうかを示す。MQTT ブローカーとの通信時にエラーを検出した場合には false が設定される。この場合には mqtt_close(), mqtt_open() ライブラリ関数を使用することで、エンドポイントを作り直してブローカーとの再接続を試みることができる。

title:Table [1..#max_endpoint] of String
　　エンドポイントのタイトル名。
　　タイトルを登録していない場合には空文字列 "" が入る

id:Table [1..#max_endpoint] of String
　　エンドポイントの ClientID文字列

host:Table [1..#max_endpoint] of String
　　エンドポイントが接続する MQTT ブローカホスト名または IPアドレス

port:Table [1..#max_endpoint] of Number

エンドポイントが接続する MQTT ブローカのポート番号

subscribe_topic_list:Table [1..#max_endpoint] of String

mqtt_subscribe() ライブラリ関数を使用して購読リクエスト(SUBSCRIBE) メッセージを送信する時にトピック名パラメータを省略したときに使用されるトピック名が入る。トピック名はカンマ区切りで複数設定される場合がある。

subscribe_qos_list:Table [1..#max_endpoint] of String

mqtt_subscribe() ライブラリ関数を使用して購読リクエスト(SUBSCRIBE) メッセージを送信する時にトピック QoS を省略したときに使用される QoSが文字列形式で入る。QoS はカンマ区切りで複数設定される場合がある。

user:Table [1..#max_endpoint] of String

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用されるユーザーユーザー名が入る。

password:Table [1..#max_endpoint] of String

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用されるパスワード文字列が入る。

will_topic:Table [1..#max_endpoint] of String

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用される Willトピック名が入る。

will_message:Table [1..#max_endpoint] of String

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用される Willメッセージ文字列が入る。

will_qos:Table [1..#max_endpoint] of Number

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用される Will QoS が数値で入る。

will_retain:Table [1..#max_endpoint] of Boolean

mqtt_connect() ライブラリ関数を使用して接続リクエスト(CONNECT) メッセージを送信する時に使用される WillRetainフラグの値が入る。

recv_buff_size:Table [1..#max_endpoint] of Number

エンドポイントでソケット受信を行うときに、内部で確保している受信バッファサイズ初期値が入る。

log:Table [1..#max_endpoint] of Boolean

エンドポイントでメッセージの送受信を行ったときに、詳細ログメッセージを出力するかどうかを設定するフラグ値が入る。

- **備考**

mqtt_all_list() は abs_agent に登録されている全ての MQTT クライアント・エンドポイントのリストを取得します。

MQTT クライアント・エンドポイントの登録や削除・修正を行うときは、”サーバー設定ファイル” を直接編集

して下さい。

- **使用例**

```
file_id = "MQTT/LIST"
-----------------------------------------------
-- エンドポイントリストを取得
-----------------------------------------------
local stat, enable, ready, title, id, host, port, subscribe_topic_list, subscribe_qos_list,
user, password, will_topic, will_message, will_qos, will_retain, recv_buff_size,log = mqtt_all_list()
if not stat then error() end
-----------------------------------------------------------------
-- JSON形式のリストを作成
-----------------------------------------------------------------
local dev_list = "["
local cnt = 0
for key,val in ipairs(id) do
  if  cnt ~= 0 then
    dev_list = dev_list .. ","
  end
  dev_list = dev_list .. '{"Enabled":' .. tostring(enable[key]) .. ',"Ready":' ..
        tostring(ready[key]) .. ',"Title":"' .. title[key] .. '","ClientID":"' .. id[key] ..
        '","BrokerHostName":"' .. host[key] .. '","PortNumber":' .. tostring(port[key]) ..
        ',"SubscribeTopic":"' .. subscribe_topic_list[key] ..
        '","SubscribeQoS":"' .. subscribe_qos_list[key] .. '","UserName":"' .. user[key] ..
        '","Password":"' .. password[key] ..
        '","WillTopic":"' .. will_topic[key] .. '","WillMessage":"' .. will_message[key] ..
        '","WillQoS":' .. tostring(will_qos[key]) ..
        ',"WillRetain":' .. tostring(will_retain[key]) .. ',"RecvBuffInit":' ..
        tostring(recv_buff_size[key]) .. ',"DetailLog":' .. tostring(log[key]) .. '}'
        cnt = cnt + 1
end
dev_list = dev_list .. "]"
-----------------------------------------------
-- エンドポイントリストをリターンパラメータに格納
-----------------------------------------------
script_result(g_taskid,"EndPointList",dev_list)
```

28.2 mqtt_find_endpoint()

- **機能概要**

  MQTTサービスに登録済みのエンドポイントタイトル名または ClientID を指定して ClientID を取得する

- **関数定義**

  stat, id = mqtt_find_endpoint(title_or_id)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
title_or_id:String	MQTT サービスに登録済みのエンドポイントタイトル名。ClientID をパラメータに指定することもできる。
id:String	MQTT サービスに登録済みのエンドポイント ClientID

- **備考**

  title で指定するエンドポイントのタイトル名は、予め MQTT サービスモジュールに登録しておく必要があります。

  title に指定したエンドポイントが見つかった場合には、stat は true に設定されて id には ClientID が設定されます。見つからなかった場合には stat に false が返ります。

  検索対象のエンドポイントはブローカに接続中であるかどうかに関係なく、MQTT サービスモジュールに登録されている全てのエンドポイントが対象になります。

- **使用例**

```
local stat,id,title
title="タイトル"
stat,id = mqtt_find_endpoint(title)
if stat then
  log_msg("id=" .. id,file_id)
else
  log_msg("error",file_id)
end
```

28.3 mqtt_open()

- **機能概要**

  MQTT エンドポイントで設定された MQTT ブローカにソケット接続を開始する

- **関数定義**

  stat = mqtt_open(title_or_id)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  title_or_id:String　　MQTT サービスに登録済みのエンドポイントタイトル名または ClientID

- **備考**

  title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。

  abs_agent 起動時には、自動的に全ての MQTT エンドポイントに対してこの関数がコールされて、ソケット接続状態になっています。このため、普通はユーザー側でこの関数をコールする必要はありません。
  "SERVER_START", "MQTT_CONNECT_ALL" Lua スクリプトでこの動作が記述されています。

  このライブラリ関数は、サーバー設定ファイル中のエンドポイントに設定した下記のパラメータを使用して MQTT ブローカにソケット接続を行います。接続に成功した場合には stat に true が返ります。このとき、mqtt_all_list() ライブラリ関数で返る enable 値も true になります。接続に失敗した場合には false が返ります。

mqtt_open() で使用されるエンドポイント設定値	
設定項目	**説明**
BrokerHostName	ソケット接続を行うホスト名または IP アドレス
Port	ソケット接続時のポート番号

  接続に成功した場合には続けて、同一エンドポイントに対して mqtt_connect() ライブラリ関数をコールしてください。mqtt_connect() コールを実行するまでに時間が掛かりすぎた場合には MQTT ブローカ側でソケット接続が切断される場合がありますので注意してください。

- **使用例**

```
if mqtt_open("EndPointタイトル#1") then
..
..
end
```

28.4　mqtt_close()

- **機能概要**

  MQTT エンドポイントで設定された MQTT ブローカとのソケット接続を終了する

- **関数定義**

  stat = mqtt_close(title_or_id)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  title_or_id:String　　MQTT サービスに登録済みのエンドポイントタイトル名または ClientID

- **備考**

  title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。

  mqtt_open() ライブラリ関数で開始された MQTT ブローカとのソケット接続を終了します。正常に終了した場合には stat に true が返ります。このとき、mqtt_all_list() ライブラリ関数で返る enable 値 は false になります。

  abs_agent 終了時には、自動的にソケット接続中の全ての MQTT エンドポイントに対してこの関数がコールされます。このため、ユーザー側でこの関数をコールする必要はありません。"SERVER_STOP", "MQTT_DISCONNECT_ALL" Lua スクリプトでこの動作が記述されています。

  この関数はソケット接続中にエラーが発生した場合など、明示的にソケットの再接続を行いたいときに使用します。この場合には mqtt_close() をコールした後 mqtt_open() をコールすることでソケットを再接続できます。

  この関数は、ソケット切断前に MQTT ブローカに対して "DISCONNECT" メッセージを自動的に送信しません。そのため、このエンドポイント接続時に指定した Will メッセージが MQTT ブローカから送信されます。"DISCONNECT" メッセージを送信したい場合には、ソケット切断前に mqtt_disconnect() ライブラリ関数をコールしてください。

- **使用例**

```
mqtt_close("EndPointタイトル#1")
```

28.5　mqtt_connect()

- **機能概要**

  MQTT ブローカに CONNECT メッセージを送信する

- **関数定義**

  stat, connack = mqtt_connect(title_or_id *[,clean]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  connack:Number　　CONNACK レスポンスメッセージの ConnectRetuenCode が入る。
  0 の場合に CONNECT メッセージの処理が正常終了したことを示す。

title_or_id:String	MQTT サービスに登録済みのエンドポイントタイトル名または ClientID
clean:Boolean	CONNECT メッセージの CleanSession フラグの値を指定する。 パラメータ省略時は true が設定される。

- **備考**

title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続しておいてください。

abs_agent 起動時には、自動的に全ての MQTT エンドポイントに対して mqtt_open() を実行した後、この関数がコールされて MQTT ブローカに CONNECT メッセージ送信済みになっています。このため、普通はユーザー側でこの関数をコールする必要はありません。"SERVER_START", "MQTT_CONNECT_ALL" Lua スクリプトでこの動作が記述されています。

このライブラリ関数は MQTT ブローカに CONNECT メッセージを送信します。このとき、サーバー設定ファイル中のエンドポイントに設定した下記のパラメータをメッセージ中に格納して MQTT ブローカに送信します。

CONNECTメッセージ送信時に使用されるエンドポイント設定値	
設定項目	**説明**
ClientID	クライアントID 文字列
UserName	ユーザー名文字列。Password項目と共に未設定(空文字列)の場合には CONNECT メッセージには格納されません。
Password	パスワード文字列。UserName項目と共に未設定(空文字列)の場合には CONNECT メッセージには格納されません。
WillTopic	Willトピック文字列。WillMessage項目と共に未設定(空文字列)の場合には CONNECT メッセージには格納されません。
WillMessage	Willメッセージ文字列。WillTopic項目と共に未設定(空文字列)の場合には CONNECT メッセージには格納されません。
WillQoS	Will QoSフラグ値
WillRetain	Will retainフラグ値

MQTT ブローカから CONNACK メッセージを受信すると、CONNACK メッセージ中の ConnectReturnCode を connack リターン値に返します。一定時間(約1秒) 以内に CONNACK を受信出来なかった場合には、エラーとなって stat には false が返ります。CONNACK メッセージを受信した場合には connack(ConnectReturnCode) の内容に関わらず常に stat は true になります。

- **使用例**

```
local cstat,connack = mqtt_connect("EndPointタイトル#1")
```

```
if cstat then
  if (connack == 0) then
    if subscribe_topic_list[key] ~= "" then
      ..
      ..
    end
  else
    ------------------------------------------------
    -- Brokerから CONNECT を拒否された時はソケット削除
    ------------------------------------------------
    log_msg("connack is not 0, closing socket ")
    mqtt_close("EndPointタイトル#1")
  end
end
```

28.6 mqtt_disconnect()

- **機能概要**

  MQTT ブローカに DISCONNECT メッセージを送信する

- **関数定義**

  stat = mqtt_disconnect(title_or_id)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  title_or_id:String　　MQTT サービスに登録済みのエンドポイントタイトル名または ClientID

- **備考**

  title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続しておいてください。

  abs_agent 終了時には、自動的にソケット接続中の全ての MQTT エンドポイントに対してこの関数がコールされます。このため、ユーザー側でこの関数をコールする必要はありません。"SERVER_STOP", "MQTT_DISCONNECT_ALL" Lua スクリプトでこの動作が記述されています。

  この関数はソケット接続中にエラーが発生した場合など、明示的にソケットの再接続を行いたいときに使用します。この場合には mqtt_disconnect() をコールした後 mqtt_close() と、続けて mqtt_open() 、mqtt_connect() をコールすることでソケットの再接続と MQTTブローカへの接続が行えます。

  このライブラリ関数は MQTT ブローカに DISCONNECT メッセージを送信します。MQTTブローカ側のソケットは

直後に切断されますので、クライアント・エンドポイント側のソケットも mqtt_close() ライブラリをコールしてすみやかに削除してください。

- **使用例**

```
mqtt_disconnect("EndPointタイトル#1")
```

28.7 mqtt_ping()

- **機能概要**

MQTT ブローカに PINGREQ メッセージを送信する

- **関数定義**

stat = mqtt_ping(title_or_id)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

title_or_id:String　　MQTT サービスに登録済みのエンドポイントタイトル名または ClientID

- **備考**

title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続した後、mqtt_connect() 関数を使用して MQTT ブローカに対する CONNECT メッセージの送信が正常に終了している必要があります。

abs_agent は定期的に現在接続中の全ての MQTT エンドポイントに対して mqtt_ping() を実行しています。このため、普通はユーザー側でこの関数をコールする必要はありません。送信間隔は サーバー設定ファイル中の "/Document/Class/MQTT/KeepAliveTimer" で設定した時間になります。"MQTT_KEEP_ALIVE_TIMER" Lua スクリプトにこれらの動作が記述されています。

このライブラリ関数は MQTT ブローカに PINGREQ メッセージを送信します。MQTT ブローカから PINGRESP メッセージを受信すると stat に true が返ります。一定時間(約1秒) 以内に PINGRESP を受信出来なかった場合には、エラーとなって stat には false が返ります。

- **使用例**

```
mqtt_ping("EndPointタイトル#1")
```

28.8 mqtt_subscribe()

- **機能概要**

MQTT ブローカに SUBSCRIBE メッセージを送信する

- **関数定義**

stat, granted_qos = mqtt_subscribe(title_or_id [,topic ,qos [,message_id]])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
granted_qos:String	SUBACK レスポンスメッセージの GrantedQos が入る。 topc, qos パラメータに複数のトピックを指定した場合には、対応する GrantedQos がカンマ区切りで複数返る
title_or_id:String	MQTT サービスに登録済みのエンドポイントタイトル名または ClientID
topic:String	SUBSCRIBE メッセージのトピック名を指定する。カンマ区切りで複数のトピックを指定することができる。 パラメータ省略時はエンドポイントで設定した "起動時に購読するトピック" 項目で設定したトピック名が使用される。
qos:String	SUBSCRIBE メッセージのトピック QoSを指定する。カンマ区切りで複数のトピック QoSを指定することができる。**複数指定できるように文字列型**である点に注意して下さい。パラメータ省略時はエンドポイントで設定した "起動時に購読するトピックのQoS" 項目で設定したトピック QoSが使用される。
message_id:Number	メッセージID を指定する。パラメータ省略時はエンドポイント毎にユニークな番号が自動で指定される。

- **備考**

title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続した後、mqtt_connect() 関数を使用して MQTT ブローカに対する CONNECT メッセージの送信が正常に終了している必要があります。

abs_agent 起動時には、自動的に全ての MQTT エンドポイントに対して mqtt_open() を実行した後、mqtt_connect()関数がコールされて MQTT ブローカに CONNECT メッセージ送信済みになっています。その後 mqtt_subscribe() 関数を topic、qos、message_id パラメータを省略した形でコールされています。このため、エンドポイントに設定したデフォルトのトピックのみを購読する場合には、この関数をコールする必要はありません。デフォルトで設定した topic 以外に、追加でMQTT ブローカに対して購読を指定したい場合にこの関数をコールします。"SERVER_START", "MQTT_CONNECT_ALL" Lua スクリプトでこの動作が記述されています。

このライブラリ関数は MQTT ブローカに SUBSCRIBE メッセージを送信します。このとき、サーバー設定ファイル中のエンドポイントに設定した下記のパラメータをメッセージ中に格納して MQTT ブローカに送信します。

SUBSCRIBEメッセージ送信時に使用されるエンドポイント設定値	
設定項目	**説明**
起動時に購読するトピック	topic パラメータを省略した場合に使用されるトピック名

起動時に購読するトピックのQoS	qos パラメータを省略した場合に使用されるトピック QoS

MQTT ブローカから SUBACK メッセージを受信すると、SUBACK メッセージ中の GrantedQos を granted_qos リターン値に返します。一定時間(約1秒) 以内に SUBACK を受信出来なかった場合には、エラーとなって stat には false が返ります。SUBACK メッセージを受信した場合には granted_qos の内容に関わらず常に stat は true になります。

- **使用例**

```
mqtt_subscribe("EndPointタイトル#1","/sensor/#","2")
```

28.9 mqtt_unsubscribe()

- **機能概要**

  MQTT ブローカに UNSUBSCRIBE メッセージを送信する

- **関数定義**

  stat = mqtt_unsubscribe(title_or_id *[*,topic *[*,message_id*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
title_or_id:String	MQTT サービスに登録済みのエンドポイントタイトル名または ClientID
topic:String	UNSUBSCRIBE メッセージ中の購読解除するトピック名を指定する。カンマ区切りで複数のトピックを指定することができる。 パラメータ省略時はエンドポイントで設定した "起動時に購読するトピック" 項目で設定したトピック名が使用される。
message_id:Number	メッセージID を指定する。パラメータ省略時はエンドポイント毎にユニークな番号が自動で指定される。

- **備考**

  title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続した後、mqtt_connect() 関数を使用して MQTT ブローカに対する CONNECT メッセージの送信が正常に終了している必要があります。

  topic パラメータで指定するトピック名は、事前に MQTT ブローカに対して mqtt_subscribe() 関数で購読を指定したトピックを指定します。

  このライブラリ関数は MQTT ブローカに UNSUBSCRIBE メッセージを送信します。このとき、サーバー設定ファイル中のエンドポイントに設定した下記のパラメータをメッセージ中に格納して MQTT ブローカに送信します。

UNSUBSCRIBEメッセージ送信時に使用されるエンドポイント設定値	
設定項目	説明
起動時に購読するトピック	topic パラメータを省略した場合に使用される購読解除するトピック名

MQTT ブローカから UNSUBACK メッセージを受信すると stat に true が返ります。一定時間(約1秒) 以内に UNSUBACK を受信出来なかった場合には、エラーとなって stat には false が返ります。

- **使用例**

```
mqtt_unsubscribe("EndPointタイトル#1","/sensor/#")
```

28.10 mqtt_publish(), mqtt_publish_hex()

- **機能概要**

  MQTT ブローカに PUBLISH メッセージを送信する

- **関数定義**

  stat = mqtt_publish(title_or_id ,topic ,message *[,qos [,retain [,message_id]]]*)

  stat = mqtt_publish_hex(title_or_id ,topic ,hexstr *[,qos [,retain [,message_id]]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
title_or_id:String	MQTT サービスに登録済みのエンドポイントタイトル名または ClientID
topic:String	PUBLISH メッセージのトピック名を指定する。
message:String	PUBLISH メッセージのペイロードに格納する文字列データを指定する。
hexstr:String	PUBLISH メッセージのペイロードに格納するバイナリデータ列を16進数文字列で指定する。
qos:Number	PUBLISH メッセージの QoS を指定する。0, 1, 2 の何れかを指定する。 パラメータ省略時は 0 が指定される。
retain:Boolean	PUBLISH メッセージのRetainフラグ値を指定する。 パラメータ省略時は false が指定される。
message_id:Number	メッセージID を指定する。パラメータ省略時はエンドポイント毎にユニークな番号が自動で指定される。

- **備考**

  title_or_id で指定する MQTTクライアント・エンドポイントは、サーバー設定ファイル中の MQTTサービスに登録しておく必要があります。また、このライブラリ関数をコールする前に mqtt_open() 関数を使用して MQTT ブローカにソケット接続した後、mqtt_connect() 関数を使用して MQTT ブローカに対する CONNECT メッセージの送信が正常に終了している必要があります。

qos に 0 を指定した場合には PUBLISH メッセージの送信が完了すると stat に true が返ります。送信に失敗した場合には false が返ります。

qos に 1 を指定した場合には PUBLISH メッセージの送信が完了した後、MQTT ブローカから PUBACK メッセージを受信すると stat に true が返ります。一定時間(約20秒) 以内に PUBACK を受信出来なかった場合には、エラーとなって PUBLISH メッセージのリトライ送信を１回行います。 リトライを行っても PUBACK を受信できなかった場合には stat に false が返ります。

qos に 2 を指定した場合には PUBLISH メッセージの送信が完了した後、MQTT ブローカから PUBREC メッセージを受信すると PUBREL メッセージを続けて送信します。一定時間(約20秒) 以内に PUBREC を受信出来なかった場合には、エラーとなって PUBLISH メッセージのリトライ送信を１回行います。 リトライを行っても PUBREC を受信できなかった場合には stat に false が返ります。

PUBREL メッセージを送信した後、MQTT ブローカから PUBCOMP メッセージを受信すると stat に true が返ります。一定時間(約20秒) 以内に PUBCOMP を受信出来なかった場合には、エラーとなって PUBREL メッセージのリトライ送信を１回行います。 リトライを行っても PUBCOMP を受信できなかった場合には stat に false が返ります。

- **使用例**

```
mqtt_publish("EndPointタイトル#1","/sensor/Node1/temperature","18.1",1)
mqtt_publish_hex("EndPointタイトル#1","/sensor/Node1/raw","1220",1)  -- 0x12,0x20 の２バイト送信
```

28.11 topic_to_tbl()

- **機能概要**

MQTT のPUBLISHメッセージで使用されるトピック文字列を "/" ごとに分解して、文字列配列にする。

- **関数定義**

arr = topic_to_tbl(topic)

- **パラメータとリターン値**

arr:Table [1..#max_item] of String
　　トピック文字列を構成していた各カラムの文字列

topicr:String　　MQTT ブローカから送信された PUBLISH メッセージの topic 文字列

- **備考**

"/" スラッシュ区切りで表現された複数カラムから構成される文字列データを、各カラム毎に分けた文字列配列に変換します。

分解した文字列が空文字列の場合にはその文字列は配列に格納されません。

例えば、"abc/def/ghi" を変換すると、arr[1] = "abc", arr[2] = "def",arr[3] = "ghi" になります。また、"/abc/def/ghi" や "/abc/def/ghi/" の変換結果も同じ配列になります。

- **使用例**

```
local tbl = topic_to_tbl("/request/ESP32Node0/anonymous/REQ_23FEF99F9")
for akey,aval in ipairs(tbl) do
    log_msg(string.format("topic_column[%d] = %s",akey,aval))
end
```

29 Raspberry Pi ハードウエアAPI (Raspberry Pi 専用)

Raspberry Pi 用にビルドされた abs_agent では Raspberry Pi ハードウエアが持つ機能を API 経由で利用することができます。利用可能なハードウエア機能は下記になります。

- GPIO　GPIO ポートのデータ入出力と、各 GPIO ポートの入出力切り替えや "Altモード"、プルアップ設定ができます。GPIO ポート値の変化を検出してイベントハンドラを実行することもできます。
- SPI　Raspberry Pi を SPI マスターデバイスとして動作させて、SPIチャンネルからシリアルデータの送信と受信、コンフィギュレーション操作ができます。
- I2C　Raspberry Pi をI2C マスターデバイスとして動作させて、バスに接続したスレーブデバイスのデータ書き込みと読み込み操作ができます。

Raspberry Pi 用 abs_agent の ハードウエアタイプ

abs_agent の Raspberry Piハードウエア API 関数 は Raspberry Pi の CPU(Soc) のハードウエア・レジスタに直接アクセスしています。この時、インストール時に設定したハードウエアタイプ毎に API 内部の動作を変えています。詳しくはインストール章中の "ハードウエアタイプ設定"の項を参照してください。

また、abs_agent は OS が提供する GPIO,SPI,I2C用のデバイスファイルを使用しませんので、raspi-config 等で有効にする必要はありません。有効にした場合には、この章で説明する API 以外にOS のデバイスファイル経由で Lua スクリプト等からハードウエアを操作することも可能です。排他制御に問題が発生する可能性がありますので、両方の方式(OSのデバイスファイル経由と abs_agentの raspi用API)を同時に使用することはお勧めしません。

デーモンタイプ(常駐タスク)のスクリプト内で Raspberry Pi H/W を操作している場合のシャットダウンについて

この章で説明しているライブラリ関数を使用してRaspberry Pi ハードウエアを操作している場合は、シャットダウン時には必ずハードウエア操作を行うスクリプトを終了して下さい。

ハードウエア操作中に強制的に OS をシャットダウンさせると、CPUレジスタ操作が途中で強制終了させられるため、OS再起動後の abs_agent 起動時にエラーが発生する可能性があります。この状態になった場合には、

OS のシャットダウン後に電源の入れなおしをすることで確実に復帰させることができます。また、このような状況になるのを防ぐため、OS シャットダウン前には必ずハードウエア操作を行っているバックグランドスクリプト等を全て終了させてください。具体的には下記のコマンドをコンソールから実行します。

```
# cd abs_agent        (abs_agent インストールディレクトリに移動)
# ./agent_task -K     (現在実行中の全スクリプトを強制終了)
# ./agent_shutdown    (abs_agent をシャットダウン)
# shutdown -h now     (OS のshutdown コマンド等を実行)
```

29.1 I2C,SPIで使用するGPIOのモード設定

abs_agent では Raspberry Pi の I2C や SPI ポートで使用する GPIOピンは予め適切なモードに自動設定されるため、ユーザー側で明示的に GPIO の "alt" モードを設定しなくても構いません。abs_agent のサーバー設定ファイル (abs_agent.xml) 中の下記の設定項目が、GPIO "alt" モードの自動設定を行うかどうかを決めています。

Raspberry Pi のI2C や SPI機能で使用する GPIO番号を、後述するデフォルト設定値以外の GPIOピンに割り当てたい場合には、設定ファイル中の下記設定項目を False にしてユーザーが GPIOモードを自分で設定できます。I2C, SPI 関連のライブラリ関数をコールする前に、使用する GPIO ポートに対して raspi_gpio_config() をコールして適切なモードをセットして下さい。GPIO モードは１度設定するだけで構いません。GPIO モードは次に同じ GPIO ポートに対して raspi_gpio_config()ライブラリ関数を実行するか、OS プロセスや他のプログラムから変更されるまでは最後に設定した値のままになります。

SPIAutoGPIOAltMode 設定項目を True にしている時でも、raspi_spi_config() 関数コール時に自動アサインされた各GPIO ポートの SPI用の設定値を、後から raspi_gpio_config() を実行することで別の設定に変更することもできます。

設定項目(/Document/Class/RASPI) 内のタグ名	説明
BSCAutoGPIOAltMode	True を設定すると raspi_i2c_write(),raspi_i2c_read(),raspi_i2c_clock() 関数コール時に GPIO ピンの "alt"モードが自動設定されます。使用する I2Cバスと GPIO 番号については後述。 False を設定すると自動設定されないので、raspi_gpio_config() ライブラリ関数を事前にコールして、I2Cバスで使用する GPIOピンの "alt" モードを設定します。 デフォルト値: True
SPIAutoGPIOAltMode	True を設定すると raspi_spi_config() 関数コール時に GPIO ピンの "alt"モードが自動設定されます。使用する SPIポートと GPIO 番号については後述。 False を設定すると自動設定されないので、raspi_gpio_config()

	ライブラリ関数を事前にコールして、SPIポートで使用する GPIOピンの "alt" モードを設定します。 デフォルト値: True

BSCAutoGPIOAltMode = True(デフォルト) 時に自動設定される GPIOピン

I2C(BSC)バス番号	自動設定時の GPIOモードと機能
0	GPIO#28 -> "alt0" SDA0 GPIO#29 -> "alt0" SCL0 GPIO#0 -> "input" I2C と切り離すため強制的に変更する GPIO#1 -> "input" I2C と切り離すため強制的に変更する
1	GPIO#2 -> "alt0" SDA1 GPIO#3 -> "alt0" SCL1
2 (指定不可)	
3 (ハードウエアタイプが"BCM2711"のみ)	GPIO#4 -> "alt5" SDA3 GPIO#5 -> "alt5" SCL3
4 (ハードウエアタイプが"BCM2711"のみ)	GPIO#6 -> "alt5" SDA4 GPIO#7 -> "alt5" SCL4
5 (ハードウエアタイプが"BCM2711"のみ)	GPIO#12 -> "alt5" SDA5 GPIO#13 -> "alt5" SCL5
6 (ハードウエアタイプが"BCM2711"のみ)	GPIO#22 -> "alt5" SDA6 GPIO#23 -> "alt5" SCL6

⚠ I2C バス#0 使用時の注意

Raspberry Pi model B+ やそれ以降の新しいバージョンの Raspberry Pi 2、Raspberry Pi 3, Raspberry Pi 4(40ピンヘッダを装備しているタイプ) では I2C bus#0 (BSC#0) は拡張ボードの ID 識別にリザーブされています。このため通常は bus#0 を使用しないことをお勧めします。

SPIAutoGPIOAltMode = True(デフォルト) 時に自動設定される GPIOピン

SPIポート番号	raspi_spi_config() 関数で enable,cs 指定時に自動設定されるGPIOモードと機能
0	enable: true 設定時 GPIO#7 -> "alt0" SPI0_CE1_N cs: "cs1" 指定時のみアサイン GPIO#8 -> "alt0" SPI0_CE0_N cs: "cs0" 指定時のみアサイン GPIO#9 -> "alt0" SPI0_MISO GPIO#10 -> "alt0" SPI0_MOSI GPIO#11 -> "alt0" SPI0_SCLK

	enable: false 設定時 GPIO#7 -> "input" cs: "cs1" 指定時のみアサイン GPIO#8 -> "input" cs: "cs0" 指定時のみアサイン GPIO#9 -> "input" GPIO#10 -> "input" GPIO#11 -> "input"
1（指定不可）	
2（指定不可）	
3 （ハードウエアタイプが"BCM2711"のみ）	enable: true 設定時 GPIO#0 -> "alt3" SPI3_CE0_N cs: "cs0" 指定時のみアサイン GPIO#1 -> "alt3" SPI3_MISO GPIO#2 -> "alt3" SPI3_MOSI GPIO#3 -> "alt3" SPI3_SCLK GPIO#24 -> "alt5" SPI3_CE1_N cs: "cs1" 指定時のみアサイン
	enable: false 設定時 GPIO#0 -> "input" cs: "cs0" 指定時のみアサイン GPIO#1 -> "input" GPIO#2 -> "input" GPIO#3 -> "input" GPIO#24 -> "input" cs: "cs1" 指定時のみアサイン
4 （ハードウエアタイプが"BCM2711"のみ）	enable: true 設定時 GPIO#4 -> "alt3" SPI4_CE0_N cs: "cs0" 指定時のみアサイン GPIO#5 -> "alt3" SPI4_MISO GPIO#6 -> "alt3" SPI4_MOSI GPIO#7 -> "alt3" SPI4_SCLK GPIO#25 -> "alt5" SPI4_CE1_N cs: "cs1" 指定時のみアサイン
	enable: false 設定時 GPIO#4 -> "input" cs: "cs0" 指定時のみアサイン GPIO#5 -> "input" GPIO#6 -> "input" GPIO#7 -> "input" GPIO#25 -> "input" cs: "cs1" 指定時のみアサイン
5 （ハードウエアタイプが"BCM2711"のみ）	enable: true 設定時 GPIO#12 -> "alt3" SPI5_CE0_N cs: "cs0" 指定時のみアサイン GPIO#13 -> "alt3" SPI5_MISO GPIO#14 -> "alt3" SPI5_MOSI GPIO#15 -> "alt3" SPI5_SCLK GPIO#26 -> "alt5" SPI5_CE1_N cs: "cs1" 指定時のみアサイン

	enable: false 設定時 GPIO#12 -> "input" cs: "cs0" 指定時のみアサイン GPIO#13 -> "input" GPIO#14 -> "input" GPIO#15 -> "input" GPIO#26 -> "input" cs: "cs1" 指定時のみアサイン
6 (ハードウエアタイプが"BCM2711"のみ)	enable: true 設定時 GPIO#18 -> "alt3" SPI6_CE0_N cs: "cs0" 指定時のみアサイン GPIO#19 -> "alt3" SPI6_MISO GPIO#20 -> "alt3" SPI6_MOSI GPIO#21 -> "alt3" SPI6_SCLK GPIO#27 -> "alt5" SPI6_CE1_N cs: "cs1" 指定時のみアサイン
	enable: false 設定時 GPIO#18 -> "input" cs: "cs0" 指定時のみアサイン GPIO#19 -> "input" GPIO#20 -> "input" GPIO#21 -> "input" GPIO#27 -> "input" cs: "cs1" 指定時のみアサイン

29.2 raspi_gpio_config()

- **機能概要**

  Raspberry Pi GPIO ポートの入出力モード、プルアップ・ダウン指定、"alt" モードの設定または取得

- **関数定義**

  stat [,mode] = raspi_gpio_config(gpio_num *[,mode [,pud]]*)

- **パラメータとリターン値**

  stat:Boolean　成功した場合は true, 失敗した場合は false が返る。

  gpio_num:Number　BCM2835/BCM2836/BCM2837/BCM2711 プロセッサの GPIO 番号。0 から 53 までの整数値を指定する。(CPU ボード上のピン番号とは違いますので注意してください)

  mode:String　GPIO のモードを示す文字列で、下記のモードが使用可能

  "input" 入力モード

  "output" 出力モード

  "alt0" gpio_num で指定したGPIO固有の "ALT0"モード

  "alt1" gpio_num で指定したGPIO固有の "ALT1"モード

  "alt2" gpio_num で指定したGPIO固有の "ALT2"モード

  "alt3" gpio_num で指定したGPIO固有の "ALT3"モード

  "alt4" gpio_num で指定したGPIO固有の "ALT4"モード

  "alt5" gpio_num で指定したGPIO固有の "ALT5"モード

  引数の "mode" と "pud" パラメータを省略すると、現在の設定値がリターン値 "mode" に返ります。

pud:String　　GPIO のプルアップ・ダウン指定を示す文字列で、下記の設定値が使用可能
このパラメータを指定する場合は、必ず "mode" パラメータも同時に指定してください。 GPIO のプルアップ・ダウン指定のみを変更をしたい場合には、この関数の代わりに、raspi_gpio_pud() を使用してください。pud パラメータを省略した場合には現在の設定値のまま変更しません。
"pullup"　プルアップ設定
"pulldown" プルダウン設定
"off"　　オフ設定

- **備考**

現在のプルアップ設定を取得することはできません。

- **使用例**

```
if not raspi_gpio_config(18,"output") then error() end
```

29.3　raspi_gpio_pud()

- **機能概要**

Raspberry Pi GPIO ポートのプルアップ・ダウン指定

- **関数定義**

stat = raspi_gpio_pud(gpio_num ,pud)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
gpio_num:Number　　BCM2835/BCM2836/BCM2837/BCM2711 プロセッサの GPIO 番号。0 から 53 までの整数値を指定する。(CPU ボード上のピン番号とは違いますので注意してください)
pud:String　　GPIO のプルアップ・ダウン指定を示す文字列で、下記の設定値が使用可能
"pullup"　プルアップ設定
"pulldown" プルダウン設定
"off"　　オフ設定

- **備考**

このライブラリ関数は raspi_gpio_config() とは違って、GPIO モードを現在の値から変更しないでプルアップ・ダウンのみを設定します。I2C や SPI 機能で自動アサインされた GPIO ポートに対してプルアップ・ダウン設定する場合に使用します。

現在のプルアップ設定値を取得することはできません。

- **使用例**

```
if not raspi_gpio_pud(18,"pullup") then error() end
```

29.4 raspi_gpio_write()

- **機能概要**

  Raspberry Pi GPIO ポート値の設定

- **関数定義**

  stat = raspi_gpio_write(gpio_num,value)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
gpio_num:Number	BCM2835/BCM2836/BCM2837/BCM2711 プロセッサの GPIO 番号。0 から 53 までの整数値を指定する。(CPU ボード上のピン番号とは違いますので注意してください)
value:Boolean	指定した GPIO の出力を High にする場合には true, Low に設定する場合には false を指定する

- **備考**

  このライブラリ関数を使用する前に対象となる GPIO ピンの入出力モードやプルアップを、raspi_gpio_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
if not raspi_gpio_write(18,true) then error() end
```

29.5 raspi_gpio_read()

- **機能概要**

  現在の Raspberry Pi GPIO ポート値を取得

- **関数定義**

  stat , value = raspi_gpio_read(gpio_num)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
gpio_num:Number	BCM2835/BCM2836/BCM2837/BCM2711 プロセッサの GPIO 番号。0 から 53 までの整数値を指定する。(CPU ボード上のピン番号とは違いますので注意してください)
value:Boolean	指定した GPIO の出力が High の場合には true, Low の場合には false が返る

- **備考**

このライブラリ関数を使用する前に対象となる GPIO ピンの入出力モードやプルアップを、raspi_gpio_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
local stat, val = raspi_gpio_read(18)
if not stat then error() end
```

29.6 raspi_change_detect()

- **機能概要**

指定した Raspberry Pi GPIO ポート値が変化した場合に、RASPI_CHANGE_DETECT イベントハンドラを実行する

- **関数定義**

stat [,flag] = raspi_change_detect(gpio_num *[,flag]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
gpio_num:Number	BCM2835/BCM2836/BCM2837/BCM2711 プロセッサの GPIO 番号。0 から 53 までの整数値を指定する。(CPU ボード上のピン番号とは違いますので注意してください)
flag:Boolean	GPIO 値が変化(High->Low または Low->High) した場合に、RASPI_CHANGE_DETECT イベントハンドラを実行する場合には true を指定する。検出しない場合には false を指定する。 パラメータを省略すると、現在の設定値がリターン値 "flag" に返ります。

- **備考**

指定した GPIOピンの入出力モードに関係なく値の変化を検出します。

GPIO ピンの値は 約 10ms 間隔で値を常に取得して、データ値が連続して同一の値を示した時に新しいポート値として内部に保存しています。この保存するポート値が前回保存していたポート値と比べて、変化していた場合に RASPI_CHANGE_DETECT イベントが発生します。

> **GPIO変化検出タイミングを調整する**
> raspi_change_detect() では予め決められた時間間隔で GPIO 値を取得して変化を検出しています。短いタイミングの変化を検出したり、チャタリングの除去を多めに取りたいときには、abs_agent のコンフィギュレーション RASPI/HWCheckInterval 設定値(デフォルト10ms)を変更することでユーザー側で調整可能です。
> また、非常に短い間隔で常に GPIO 値を監視し続けたい場合(PLC制御等)では、raspi_gpio_read() 関数を繰り返しコールする形のユーザースクリプトを作成して、別タスクで起動させてください。

- **使用例**

```
if not raspi_change_detect(18,true) then error() end
```

29.7 raspi_spi_config()

- **機能概要**

  Raspberry Pi SPIポートの動作モード設定

- **関数定義**

  stat *[,*enable, mode, clock_div, cs, polarity*]* =

  raspi_spi_config(*[*port*][,*enable *[,*mode, clock_div, cs, polarity*]])*

- **パラメータとリターン値**

  stat:Boolean　成功した場合は true, 失敗した場合は false が返る。

  port:Number　SPIポート番号を指定する。省略時は 0 (SPI#0) になる。

  0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能。

  enable:Boolean　SPIポートを使用するかどうかを指定する。

  パラメータを省略すると、現在の設定値がリターン値 "enable"に返ります。

  mode:String　SPI 動作モードを指定する。下記の設定値が使用可能

  "mode0"　rest state of clock = Low , First SCLK transition at middle of data bit

  "mode1"　rest state of clock = Low , First SCLK transition at beginning of data bit

  "mode2"　rest state of clock = High, First SCLK transition at middle of data bit

  "mode3"　rest state of clock = High, First SCLK transition at beginning of data bit

  パラメータを省略すると、現在の設定値がリターン値 "mode"に返ります。

  clock_div:Number　SPI クロック分周比を指定する。SPI clock frequency = (250MHz / clock_div)

  0 から 65534 の間の偶数値のみが指定できます。 0 を指定した場合には 65536 を分周比とします。

  パラメータを省略すると、現在の設定値がリターン値 "clock_div" に返ります。

  cs:String　ChipSelect 信号に使用するピンを指定する

  "cs0"　SPI<Port>_CE0_N にアサインされた GPIOを使用

  "cs1"　SPI<Port>_CE1_N にアサインされた GPIOを使用

  "cs2"　Chip select 2

  "reserved" Reserved

  enable パラメータ指定時に cs パラメータを省略すると、"cs0" が選択されます。

  パラメータ省略時には現在の設定値がリターン値 "cs"に返ります。

  polarity:String　ChipSelect 信号の極性を指定する

  "low"　chip select lines are active low

  "high"　chip select lines are active high

  パラメータを省略すると、現在の設定値がリターン値 "polarity" に返ります。

- **備考**

SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

cs パラメータの "cs2", "reserved" は Raspberry Pi OS 上で作成される SPIデバイスドライバの設定値に対応するための選択肢です。SoCのドキュメントには詳細が記載されていないため使用時は注意してください。

- **使用例**

```
if not raspi_spi_config(true) then error() end
```

```
if not raspi_spi_config(true, "mode0", 250, "cs0", "low") then error() end
...
... (SPI#0 の操作)
...
if not raspi_spi_config(false) then error() end
```

29.8 raspi_spi_write()

- **機能概要**

Raspberry Pi SPIポートのスレーブデバイスに、16進数表記の複数バイトのデータを書き込む。同時に書き込みバイト数と同じだけスレーブデバイスからデータを取得する。

- **関数定義**

stat ,recv_hexstr = raspi_spi_write([port,] send_hexstr)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
port:Number	SPIポート番号を指定する。省略時は 0 (SPI#0) になる。 0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能。
send_hexstr:String	スレーブデバイスに送信するバイナリデータ。16進数の文字列形式で指定する。
recv_hexstr:String	スレーブデバイスから受信したバイナリデータ。16進数の文字列形式。

- **備考**

send_hexstr で指定した 16進数形式の文字列データをバイナリ値に変換した後、SPIポートから送信します。16進数形式の文字列データは連続した複数のバイナリデータを指定することができます。

この API は別スレッド間でデバイスを競合することなく実行することができます。複数バイトのデータ送受信は必ず１つのコール内のトランザクションで連続的・排他的に行われます。

スレーブデバイスにデータを１バイト単位で送信する場合には、ライブラリコールを繰り返したときにより高

速で動作する raspi_spi_write_byte() を使用してください。

SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

このライブラリ関数を使用する前に SPI の動作モードを、raspi_spi_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
local stat,reply = raspi_spi_write("0A0B0C")
if not stat then error() end;
log_msg("reply = " .. reply)
```

29.9 raspi_spi_write_byte()

- **機能概要**

Raspberry Pi SPIポートのスレーブデバイスに1バイトのデータを書き込む。同時にスレーブデバイスから１バイトのデータを取得する。

- **関数定義**

stat ,recv_byte = raspi_spi_write_byte(*[*port,*]* send_byte)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
port:Number	SPIポート番号を指定する。省略時は 0 (SPI#0) になる。 0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能
snd_byte:Number	スレーブデバイスに送信するバイナリデータ。 1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。
recv_byte:Number	スレーブデバイスから受信したバイナリデータ。

- **備考**

send_byte で指定した1 バイトデータを SPIポートから送信します。同時にスレーブデバイスから１バイトのデータを受信します。

この API は別スレッド間でデバイスを競合することなく実行することができます。

SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

このライブラリ関数を使用する前に SPI の動作モードを、raspi_spi_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
local stat,reply = raspi_spi_write_byte(0x0A)
if not stat then error() end;
log_msg("reply = " .. bit_tohex(reply,2))
```

29.10 raspi_spi_write_shmem()

- **機能概要**

Raspberry Pi SPIポートのスレーブデバイスに、グローバル共有メモリエリアに格納されたバイナリデータを書き込む。サイズが大きなデータ書き込みを高速で行いたい場合に適しています。

- **関数定義**

stat = raspi_spi_write_shmem(port, channel *[,offset [,len [,<fill_arr|fill_offset>]]]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

port:Number　　SPIポート番号を指定する。他の raspi_spi_xxxx() ライブラリ関数とは違って必須パラメータです。
0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能。

channel:String　　グローバル共有メモリエリアのチャンネル名

offset:Number　　グローバル共有メモリエリア先頭から offset バイト分ずらした位置のデータから送信する。このパラメータを省略した場合は 0 を指定したのと同じ。

len:Number　　offset 位置から連続して送信するデータ数(bytes) を整数値で指定する。0 を指定するとメモリエリア終端まで取り出す。このパラメータを省略した場合は 0 を指定したのと同じ。

fill_arr:Table [1..#max_item] of Number
SPI ポート出力後に、送信対象となったグローバル共有メモリエリアに書き込むデータ配列を指定する。各データ項目は1バイトに収まる 0から255 (0x0..0xff)の整数を指定する。
送信対象となったメモリエリアに fill_arr[] 配列のデータ項目値を順に書き込む。配列の最後に達した場合には fill_arr[1] に戻って繰り返す。
パラメータ省略時はメモリエリアの内容は変化させない。

fill_offset:Number　　SPI ポート出力後に、送信対象となったグローバル共有メモリエリアを fill_offset で指定したデータで上書きする。このとき、fill_offset から lenパラメータで指定したサイズ分のメモリ領域が存在しない場合にはエラーになります。
また、offset からの送信対象となるメモリ領域と、fill_offset からのメモリ領域が重ならないように注意してください。

- **備考**

この API は別スレッド間でデバイスを競合することなく実行することができます。複数バイトのデータ送受信は必ず１つのコール内のトランザクションで連続的・排他的に行われます。また、このライブラリ関数実行中のグローバルメモリエリアは排他的に使用され、他のスレッドからのメモリエリア・アクセスは禁止されます

ので注意して下さい。

fill_arr もしくは fill_offset パラメータを指定すると、SPI ポートに出力したメモリエリア範囲のデータ値を変更することが出来ます。SPI ポート経由でディスプレイデバイス出力後に、グローバル共有メモリエリアに保存しているディスプレイバッファをクリアしたい時に使用できます。

SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

このライブラリ関数を使用する前に SPI の動作モードを、raspi_spi_config() ライブラリ関数で適切に設定してください。

29.11 raspi_spi_read()

- **機能概要**

  Raspberry Pi SPIポートのスレーブデバイスから指定したバイト数のデータを読み込む。結果は16進数表記の複数のバイトデータが得られる。

- **関数定義**

  stat ,recv_hexstr = raspi_spi_read([port,] read_len)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
port:Number	SPIポート番号を指定する。省略時は 0 (SPI#0) になる。 0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能
read_len:Number	スレーブデバイスから読み込むデータバイト数を指定する。
recv_hexstr:String	スレーブデバイスから受信したバイナリデータ。16進数の文字列形式。

- **備考**

  SPI スレーブデバイスから ReadLen バイト数のデータを取得する。この時、スレーブデバイスには同一バイト数分の 0x00が送信される。もし送信データに任意の値を指定したい場合には raspi_spi_write(), raspi_spi_write_byte() ライブラリ関数を使用してください。

  スレーブデバイスからデータを１バイト単位で受信したい場合には、ライブラリコールを繰り返したときにより高速で動作する raspi_spi_read_byte() を使用してください。

  SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

  このライブラリ関数を使用する前に SPI の動作モードを、raspi_spi_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
local stat,reply = raspi_spi_read(4)
```

```
if not stat then error() end;
log_msg("reply = " .. reply)
```

29.12 raspi_spi_read_byte()

- **機能概要**

  Raspberry Pi SPI#0 チャンネルのスレーブデバイスから1バイトのデータを読み込む。

- **関数定義**

  stat ,recv_byte = raspi_spi_read_byte(*[port]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
port:Number	SPIポート番号を指定する。省略時は 0 (SPI#0) になる。 0 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能
recv_byte:Number	スレーブデバイスから受信したバイナリデータ。

- **備考**

  SPIスレーブデバイスから 1バイトのデータを取得する。この時、スレーブデバイスには1バイト数分の 0x00 が送信される。もし送信データに任意の値を指定したい場合には raspi_spi_write_byte() ライブラリ関数を使用してください。

  SPI のデータ転送時のビット送信順序は常に MSB が最初になります。

  このライブラリ関数を使用する前に SPI の動作モードを、raspi_spi_config() ライブラリ関数で適切に設定してください。

- **使用例**

```
local stat,reply = raspi_spi_read()
if not stat then error() end;
log_msg("reply = " .. bit_tohex(reply,2))
```

29.13 raspi_i2c_write()

- **機能概要**

  Raspberry Pi I2Cバス(BSC) に接続されたスレーブデバイスにバイナリデータを書き込む。

  データ書き込み後、指定したバイト数分のデータを同一スレーブデバイスから読み込むことができる

- **関数定義**

  stat *[,recv_hexstr]* = raspi_i2c_write(bus, slave_addr, send_hexstr *[,read_len]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。

bus:Number	I2Cバス番号を指定する。 0,1 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能
slave_addr:String	I2C スレーブデバイスアドレスを16進数表記で指定する。7 bit アドレスが指定可能で R/W ビットフィールドは含めないでアドレス部分だけを指定する。
send_hexstr:String	スレーブデバイスに送信するバイナリデータ。16進数の文字列形式で指定する。
read_len:Number	スレーブデバイスから読み込むデータバイト数を指定する。
recv_hexstr:String	スレーブデバイスから受信したバイナリデータ。16進数の文字列形式。

- **備考**

スレーブデバイスアドレスで指定したI2C デバイスに指定したデータを書き込みます。I2C の “MASTER TRANSMITTER MODE” でデータを送信します。

read_len パラメータを指定した場合には、データ書き込み後に同一スレーブデバイスからデータ取得を行います。このとき I2C の“MASTER RECEIVER MODE”でデータ受信を行います。

Raspberry Pi ハードウエアの制限から、書き込み後に続く読み込み操作のトランザクションに "REPEATED START MODE"は使用していません。

この API は別スレッド間でデバイスを競合することなく実行することができます。複数バイトのデータ送信とそれに続く複数バイトのデータ受信は必ず１つのコール内のトランザクションで連続的・排他的に行われます。

- **使用例**

```
-- Atmel24LC256 の EEPROMアドレス 0x0000 から 4 バイトデータ 0xAA,0xBB,0xCC,0xDD書き込み
-- 24LC256 デバイスの A0,A1,A2 ピンは “low” で SlaveAddress は 0x50 に設定されている場合
local stat,reply = raspi_i2c_write(1,"50","0000AABBCCDD")
if not stat then error() end;
```

```
-- Atmel24LC256 の EEPROMアドレス 0x0000 から 4 バイトデータ読み込み
-- 24LC256 デバイスの A0,A1,A2 ピンは “low” で SlaveAddress は 0x50 に設定されている場合
local stat,reply = raspi_i2c_write(1,"50","0000",4)
if not stat then error() end
log_msg("reply = " .. reply)
```

29.14 raspi_i2c_write_shmem()

- **機能概要**

Raspberry Pi I2Cバス(BSC) に接続されたスレーブデバイスにグローバルメモリエリアに格納されたバイナリデータを書き込む。サイズが大きなデータ書き込みを高速で行いたい場合に適しています。

- **関数定義**

stat = raspi_i2c_write_shmem(bus, slave_addr, channel *[,offset [,len [,ins_hexstr]]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
bus:Number	I2Cバス番号を指定する。 0,1 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能
slave_addr:String	I2C スレーブデバイスアドレスを16進数表記で指定する。7 bit アドレスが指定可能で R/W ビットフィールドは含めないでアドレス部分だけを指定する。
channel:String	グローバル共有メモリエリアのチャンネル名
offset:Number	グローバル共有メモリエリア先頭から offset バイト分ずらした位置のデータから書き込む。このパラメータを省略した場合は 0 を指定したのと同じ。
len:Number	offset 位置から連続して書き込むデータ数(bytes) を整数値で指定する。0 を指定するとメモリエリア終端まで書き込む。このパラメータを省略した場合は 0 を指定したのと同じ。
ins_hexstr:String	channel に指定したメモリエリアのデータを**書き込む前に**、追加でスレーブデバイスに書き込むバイナリデータを指定する。16進数の文字列形式で指定する。パラメータ省略時は追加のデータ書き込みは行わない。

- **備考**

スレーブデバイスアドレスで指定したI2C デバイスに指定したデータを書き込みます。I2C の “MASTER TRANSMITTER MODE” でデータを送信します。

ins_hexstr パラメータを指定した場合には 16進数形式の文字列データをバイナリ値に変換した後、データをスレーブデバイスに書き込みます。16進数形式の文字列データは連続した複数のバイナリデータを指定することができます。その後、channel に指定したグローバル共有メモリエリアのバイナリデータをスレーブデバイスに書き込みます。

この API は別スレッド間でデバイスを競合することなく実行することができます。複数バイトのデータ送受信は必ず１つのコール内のトランザクションで連続的・排他的に行われます。

29.15 raspi_i2c_read()

- **機能概要**

Raspberry Pi I2Cバス(BSC) に接続されたスレーブデバイスから指定したバイト数分のデータを読み込む

- **関数定義**

stat ,recv_hexstr = raspi_i2c_read(bus, slave_addr, read_len)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
bus:Number	I2Cバス番号を指定する。 0,1 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能

slave_addr:String　　I2C スレーブデバイスアドレスを16進数表記で指定する。7 bit アドレスが指定可能で R/W ビットフィールドは含めないでアドレス部分だけを指定する。

read_len:Number　　スレーブデバイスから読み込むデータバイト数を指定する。

recv_hexstr:String　　スレーブデバイスから受信したバイナリデータ。16進数の文字列形式。

- **備考**

スレーブデバイスアドレスで指定したI2C デバイスからデータを読み込みます。I2C の “MASTER RECEIVER MODE” でデータを転送します。

この API は別スレッド間でデバイスを競合することなく実行することができます。複数バイトのデータ受信は必ず１つのコール内のトランザクションで連続的・排他的に行われます。

- **使用例**

```
-- Atmel24LC256 から 4 バイトデータ読み込み。EEPROM アドレスはデバイス内部のレジスタ値を使用
-- 24LC256 デバイスの A0,A1,A2 ピンは “low” で SlaveAddress は 0x50 に設定されている場合
local stat,reply = raspi_i2c_read(1,"50",4)
if not stat then error() end
log_msg("reply = " .. reply)
```

29.16　raspi_i2c_clock()

- **機能概要**

Raspberry Pi I2Cバス(BSC) のクロック周波数(分周比)の設定・取得

- **関数定義**

stat *[,clock_div]* = raspi_i2c_clock(bus *[,clock_div]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

bus:Number　　I2Cバス番号を指定する。
0,1 または 3,4,5,6(ハードウエアタイプ "BCM2711"のみ)を指定可能

clock_div:Number　　I2Cクロック分周比を指定する。I2C clock frequency = (150MHz / clock_div)
0 から 32768 の間の偶数値のみが指定できます。 0 を指定した場合には 32768 を分周比とします。
ライブラリ関数のパラメータを省略すると、現在の設定値がリターン値 "clock_div" に返ります。

- **備考**

このライブラリ関数をコールしない場合には、Raspberry Pi ハードウエアのデフォルト値(1500 => 100KHz)、または OS によって設定されたクロック分周比が使用されます。
バス毎に設定したクロック分周比は、変更するまで最後に設定した値のまま変化しません。

- **使用例**

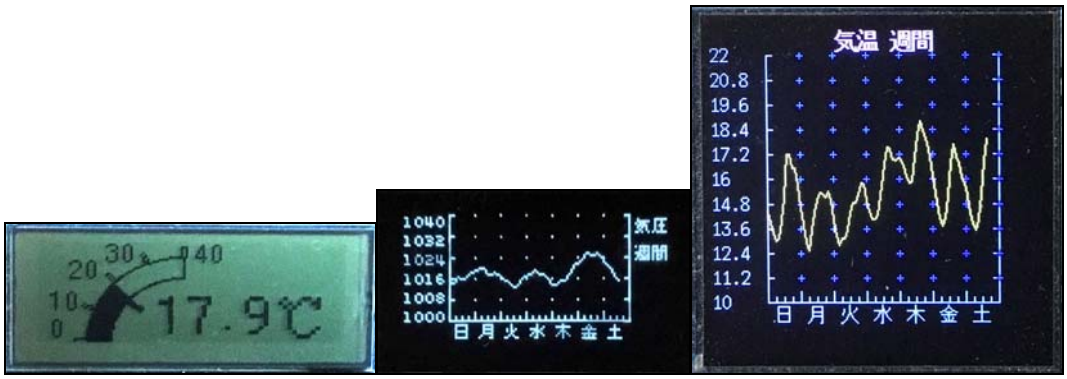
```
if not raspi_i2c_clock(1,0x5dc) then error() end -- I2C バス#1を 100KHz に設定
```

30　カラー(RGB)・モノクロ(Bitmap)グラフィックAPI

RGBディスプレイ(TFT LCD/IPS LCD等)やモノクロビットマップLCDへの描画に使用できる、abs_agent 内に作成するディスプレイ・バッファ用ライブラリ関数です。ディスプレイバッファには文字や図形を描画可能で、abs_agent のグローバル共有メモリエリア機能を利用して高速に描画可能です。

ライブラリ関数では、abs_agent のグローバル共有メモリエリアに確保しているディスプレイバッファに対して文字や図形を書き込みます。書き込んだディスプレイバッファの内容は、グローバル共有メモリエリア用のライブラリ関数 shmem_copy() 等を使用して描画済みのピクセルデータ(バイナリデータ)として取り出します。

取り出したディスプレイバッファのピクセルデータはその後、SPI や I2C インターフェイスを経由してLCDデバイスに書き込むことで図形を表示できます。Raspberry Pi 用の abs_agent では raspi_spi_write_shmem(), raspi_i2c_write_shmem() ライブラリ関数を利用してディスプレイバッファの内容を LCD デバイスに直接転送することもできます。



上記は、左からビットマップLCD(AQM1248A),モノクロOLED(SSD1306),RGB LCD(ST7789)の各デバイスに graphic2ライブラリで描画した例です。これら以外のデバイスにもSPI や I2C インターフェイスへ出力するスクリプト部分に簡単な修正を加えるだけで対応できます。下記は ST7789 ドライバの画面サイズと初期化パラメータ、SPI転送部分を一部修正して、Pimoroni社製 Display HAT Mini 用のドライバを作成したものです。

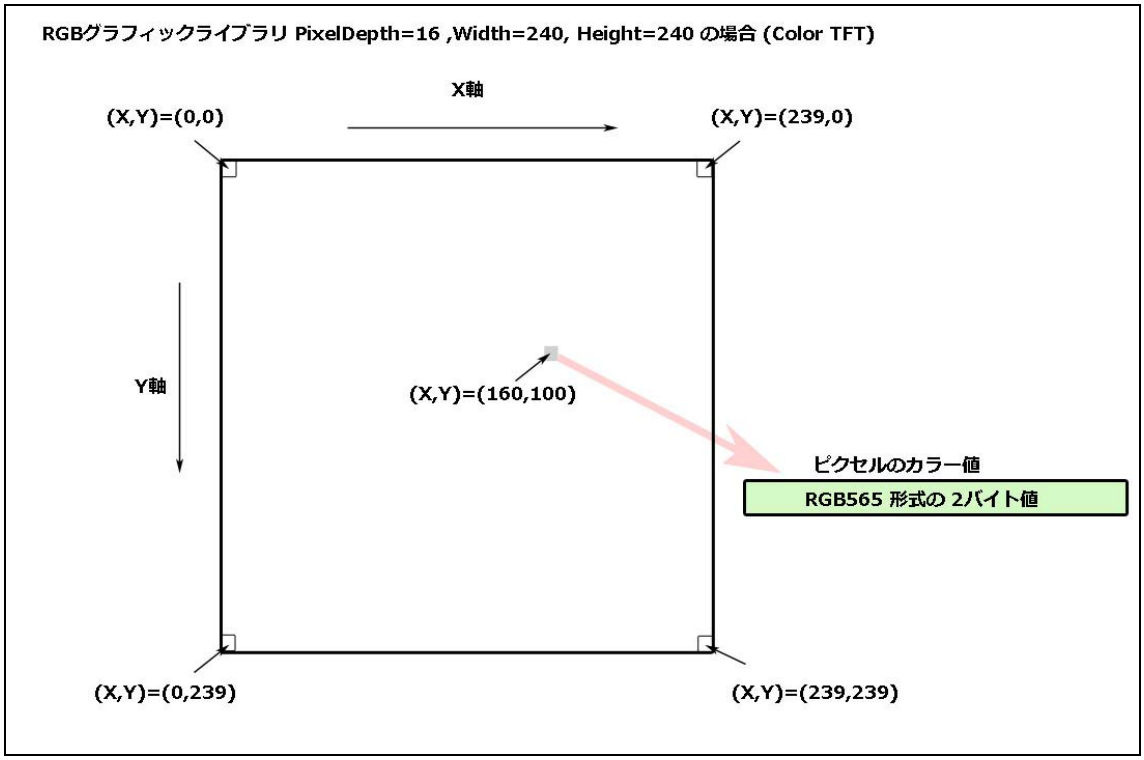
ライブラリ内部のディスプレイバッファは、graphic2_init() ライブラリ関数をコールしてピクセル幅(width)と高さ

(height)、最大ページ数(max_pages)、ピクセル毎のビット深度(depth)を指定して作成します。

ライブラリで指定可能なビット深度(depth)と、ディスプレイバッファへの格納データの関係は以下になります。

ビット深度(depth) の値	ディスプレイバッファへの格納データ
16	RGB565 形式 Red(5bit),Green(6bit),Blue(5bit)の 2バイト長のカラー値を格納する。 カラー値の MSB側が、ディスプレイバッファ中の低位バイト側になるように格納する。 ディスプレイバッファのサイズ(bytes)は width * height * 2 * max_pages です。
-1	ビットマップ形式(詳しくは後述) 1 または 0 のカラー値を、ディスプレイバッファ中の各データのビット値で格納する。 ディスプレイバッファのサイズ(bytes)は width * (height / 8) * max_pages です。

ディスプレイバッファの横方向が X軸で、縦方向が Y軸になります。ディスプレイバッファの原点座標はディスプレイの左上になり、座標値は (X,Y)=(0,0)です。ディスプレイバッファ右下の座標値は (X,Y)=(<width>-1,<height>-1)です。<width> と <height> は graphic2_init() 関数でディスプレイバッファを作成したときのパラメータ値です。
下記はディスプレイバッファの概要を図で表したものです。<depth> = 16を指定して RGB565形式で画像バッファを作成した例です。

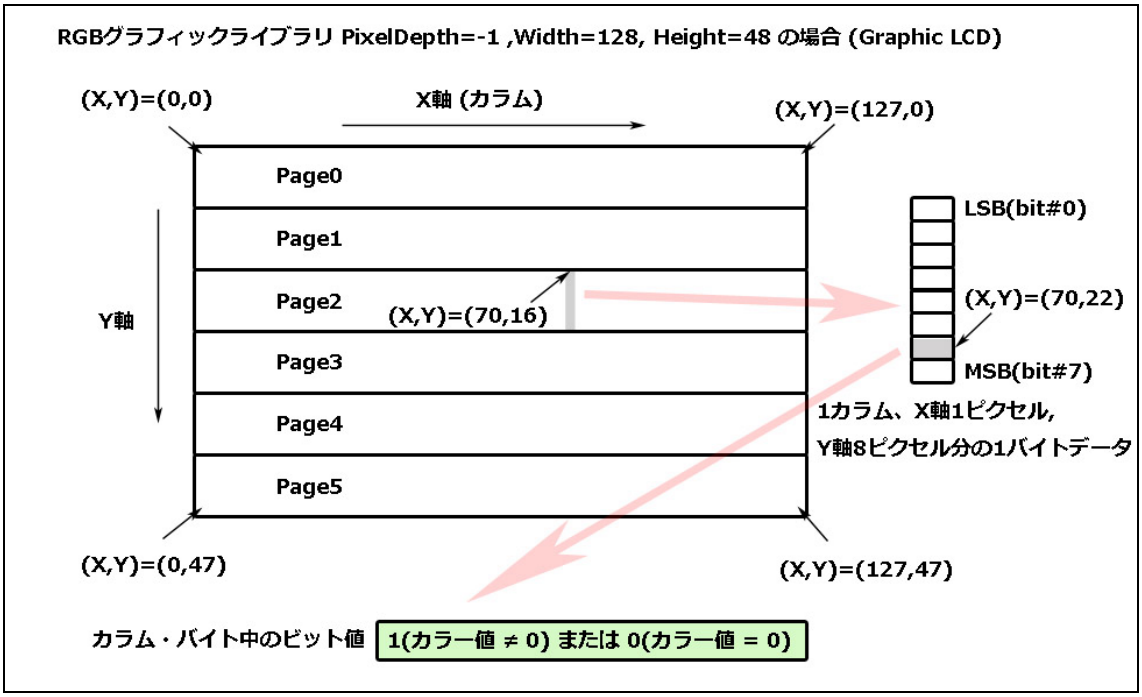


上図は１ページ分のディスプレイバッファなので、複数ページを作成した場合には同様の構造が複数存在します。このとき、複数ページのメモリ領域はグローバル共有メモリエリア内に連続して確保されます。

各ピクセル位置のカラー値は integer(最大32bit長)タイプをライブラリのパラメータに指定します。ただし実際に格納する値は、<depth> で指定したカラー値フォーマットにします。例えば depth = 16 を指定した場合には 0x0000 から

0xffff までの RGB565 フォーマット(16ビット長)の値をライブラリのカラー・パラメータに指定します。

<depth> = -1 を指定した場合には、ディスプレイバッファ中の各バイトデータ内のビット値が 1ピクセルにマッピングされます。以下が概要図です。



Y 軸方向にディスプレイバッファ中の各カラム・バイトが配置されますので、<height> は必ず 8 の倍数を指定する必要があります。このビットマップ形式でディスプレイバッファを構成した場合には、カラー値は 1 または 0 のどちらかになります。ライブラリ関数のカラー・パラメータに 0 を指定した場合にはピクセル値は 0 になり、0以外のカラー値(1など)を指定した場合のピクセル値は 1 になります。図中の "Page0" .. "Page5" は Y軸を 8 ビット毎に区切った領域の名称で、ディスプレイバッファ中のデータバイトが縦方向に並んで X軸方向に格納されています。(グラフィックライブラリの page パラメータとは関係ありません)

ディスプレイ出力、初期化用ライブラリ関数の使用について

graphic2 ライブラリのデバイス初期化とデバイス描画ライブラリ関数はスクリプトファイルとして提供していますので、別プラットフォーム(PC/AT)や、別インターフェイス(I2C, SPI)、異なった画面サイズ、別レイアウト(上下左右等) にも簡単に対応できます。Raspberry Pi 用の abs_agent インストールキットでは、デバイス初期化と描画部分のスクリプトライブラリは下記のファイル名で preload ライブラリに格納されています。

* preload/100_RASPI/_ST7789_240x240/ST7789_LIB.lua
* preload/100_RASPI/_AQM1248A_GRAPHIC2/AQM1248A_LIB.lua
* preload/100_RASPI/_OLED_SSD1306_GRAPHIC2/OLED_SSD1306_128x64.lua
* preload/100_RASPI/_displayhatmini/DISPHATMINI_LIB.lua

上記のライブラリ関数を使用する場合には、使用したいデバイスライブラリが格納されたディレクトリ名先頭の

アンダースコア文字 "_" を消して、下記の様にリネームしてから abs_agent を再起動してください。

* preload/100_RASPI/ST7789_240x240/ST7789_LIB.lua
* preload/100_RASPI/AQM1248A_GRAPHIC2/AQM1248A_LIB.lua
* preload/100_RASPI/OLED_SSD1306_GRAPHIC2/OLED_SSD1306_128x64.lua
* preload/100_RASPI/displayhatmini/DISPHATMINI_LIB.lua

上記のスクリプト中に記載されている、インターフェイス定義や画面サイズ、ページ数、画面スキャン等のレジスタ値を、新しいデバイスに合わせた関数に定義することで、簡単に新しいデバイスに対応することが可能です。インストールキットに同梱された上記のスクリプトを、別の preload ディレクトリにコピーすることで新しいデバイス用のライブラリを簡単に作成できます。同一デバイスを複数同時使用したい場合には、スクリプト内の定数名とその値、関数名と関数内部で使用している定数部分に修正を加えるだけで実現できます。

30.1 graphic2_init()

- **機能概要**

  グラフィックライブラリで使用するディスプレイ・バッファを新規に作成する。

  パラメータ省略時は現在設定されているディスプレイ・バッファ情報を取得する。

- **関数定義**

  stat *[*,width, height, pages, depth*]* = graphic2_init(channel, *[*width, height *[*,max_pages *[*,depth*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
width:Number	RGB デバイスの横方向(X軸) のピクセル数。
heightNumber	RGB デバイスの縦方向(Y軸) のピクセル数。
max_pages:Number	1 以上の整数を指定する。RGBデバイスの描画対象ページを切り替えて複数利用したい場合には 2 以上を指定する。描画や表示用のライブラリ関数コール時に指定する page パラメータには 0 から (max_pages - 1) までの値を指定する。 デフォルト値:1
depth:Number	1ピクセルあたりのビット数(ピクセル深度)を指定する。 デフォルト値:16

- **備考**

  このライブラリ関数をコールすると、指定したチャンネル名のディスプレイバッファ領域が作成される。

  グラフィックライブラリでは、ここで作成したディスプレイバッファに対して図形や文字を描画する。

  ディスプレイバッファの実体は、abs_agent 内のグローバル共有メモリ領域のチャンネルに作成される。

  他のグラフィックライブラリ API をコールする前に、必ずこの関数を最低１度コールしてディスプレイバッファを作成しておく必要があります。

ディスプレイバッファを明示的に削除したい場合には、channel パラメータで指定したグローバル共有メモリ領域を shmem_remove() ライブラリ関数で削除できます。

- **使用例１(RGB565カラー値、240x240サイズ)**

```
ST7789_WIDTH = 240
ST7789_HEIGHT = 240
ST7789_DISP_BUFF_CH = "ST7789DispBuff"
ST7789_PAGE_MAX = 2
graphic2_init(ST7789_DISP_BUFF_CH,ST7789_WIDTH,ST7789_HEIGHT,ST7789_PAGE_MAX,16)
```

- **使用例２(Bitmap値、形式 128x48サイズ)**

```
AQM1248A_WIDTH = 128
AQM1248A_HEIGHT = 48
AQM1248A_DISP_BUFF_CH = "AQM1248ADispBuff"
AQM1248A_PAGE_MAX = 2
graphic2_init(AQM1248A_DISP_BUFF_CH,AQM1248A_WIDTH,AQM1248A_HEIGHT,AQM1248A_PAGE_MAX,-1)
```

- **使用例３(Bitmap値、形式 128x64サイズ)**

```
OLED1306_WIDTH = 128
OLED1306_HEIGHT = 64
OLED1306_DISP_BUFF_CH = "OLD1306DispBuff"
OLED1306_PAGE_MAX = 2
graphic2_init(OLED1306_DISP_BUFF_CH,OLED1306_WIDTH,OLED1306_HEIGHT,OLED1306_PAGE_MAX,-1)
```

30.2 rgb565()

- **機能概要**

Red(8bit) Green(8bit) Blue(8bit) を1つのカラー値(RGB565形式16bit幅)に変換する。

- **関数定義**

color = rgb565(r, g, b)

- **パラメータとリターン値**

r:Number	Red 成分の値を指定する。(0 から 255 までの整数)
g:Number	Green 成分の値を指定する。(0 から 255 までの整数)
b:Number	Blue 成分の値を指定する。(0 から 255 までの整数)
color:Number	RGB565(16ビット幅)のカラー値。(0x0000 から 0xffff までの整数) ディスプレイバッファのピクセル深度を16に設定した場合に利用可能な、カラー値を返します。

- **備考**

パラメータ r, g, b に 255(0xff) よりも大きな数値を指定した場合には、そのパラメータ値は 255 に設定されます。

- **使用例１**

```
local c_blue = rgb565(0x1e,0x2c,0x90)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,10,20,23,40,c_blue,true)
```

- **使用例２**

```
graphic2_print(ST7789_DISP_BUFF_CH,0,0,"ABCD123これは試験文字列です。",rgb565(0,252,219))
```

30.3 graphic2_clear()

- **機能概要**

ディスプレイバッファの内容を 0x00 にする。

- **関数定義**

stat = graphic2_clear(channel *[,page]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
page:Number	クリア対象とするページ番号を指定する。 パラメータ省略時は 全てのページを対象とする。

- **使用例**

```
graphic2_clear(ST7789_DISP_BUFF_CH)
```

30.4 graphic2_print()

- **機能概要**

文字列を指定したフォントでディスプレイバッファに描画する。

- **関数定義**

stat, max_x, max_y = graphic2_print(channel, x, y, str, color *[,font [,size [,sp_h [,page]]]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	文字列の表示座標(X軸) を指定する。 nil を指定すると画面左端 (0) になる。
y:Number	文字列の表示座標(Y軸) を指定する。 nil を指定すると画面上端から文字列を表示するように自動設定される。
str:String	描画する文字列。文字列はマルチバイト文字や ASCII 文字を指定できます。

color:Number　　描画するカラー値。

font:String　　グローバル共有メモリエリア領域にロードされている FONTX データを格納したチャンネル名(フォント名)

パラメータ省略時は "SHINONOME_FONTX" が指定されます。

size:Number　　文字の拡大倍率を指定します。省略時は 1 が設定されます。

sp_h:Number　　自動で折り返して文字列を表示する場合の、縦方向の文字間隔を指定します。

パラメータ省略時は 0 が設定されます。

page:Number　　ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

max_x:Number　　文字列描画範囲の右端座標値が返る。

max_y:Number　　文字列描画範囲の下端座標値が返る。

- **備考**

abs_agent インストールキットで提供されているデフォルトフォントは、scripts/preload/013_FONTX に格納されています。フォントデータの詳細については格納されているファイルを参照してください。この関数をコールしたときにフォントライブラリが初期化されていなかった場合には自動的にフォントデータが読み込まれます。最初に fontx_try_init() をコールすることで、フォントデータを事前にロードしておくことも出来ます。

(abs_agent の FONTX ライブラリでロードされるフォント一覧)

Font名(グローバル共有メモリエリア・チャンネル名)	説明
MISAKI_FONTX	美咲フォント (全角 8x8 Shift-JIS)
AYU_FONTX	AYUフォント (全角 20x20 Shift-JIS)
E_AYU_FONTX	AYUフォント (半角 10x20 ASCII)
SHINONOME12_FONTX	東雲フォント (全角 12x12 Shift-JIS)
SHINONOME14_FONTX	東雲フォント (全角 14x14 Shift-JIS)
SHINONOME_FONTX	東雲フォント (全角 16x16 Shift-JIS)
E_SHINONOME12_FONTX	東雲フォント (半角 6x12 ASCII)
E_SHINONOME14_FONTX	東雲フォント (半角 7x14 ASCII)
E_SHINONOME_FONTX	東雲フォント (半角 8x16 ASCII)
FREEPIXEL_FONTX	FreePixelフォント (8x16 ASCII)
ASCII58_FONTX	ASCIIミニフォント (5x8 ASCII 大文字)

font パラメータ省略時は "SHINONOME_FONTX"(16x16全角)が指定されます。

font パラメータで指定したフォントが ASCII 文字用の場合には str で指定された文字列は全てバイト単位の文字コードで描画します。

font パラメータで指定したフォントデータが Shift-JIS文字用の場合には全角フォントで１文字毎に表示し

ます。このとき、font に指定したチャンネル名の先頭に "E_" を付加した名前の ASCII文字用フォントが別途存在する場合には、ASCII 文字(0 .. 0xFE の文字コード)表示にはこの ASCII用フォントを使用して描画します。ASCII文字用フォントが見つからなかった場合には、(同じ字形の)全角フォントで ASCII文字を表示します。

str に指定した文字列を描画するときに画面右端を超える場合には、自動的に折り返して表示します。画面下端を超える文字は描画されません。

- **使用例１(ST7789 RGB LCD)**

```
fontx_try_init()
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
graphic2_print(ST7789_DISP_BUFF_CH,0,0,"ABCDEF abcdef",0xFFE0,"FREEPIXEL_FONTX") -- 8x16 ASCII font
graphic2_print(ST7789_DISP_BUFF_CH,0,16,"漢字とASCII",0x07FF,"MISAKI_FONTX") -- 8x8 SJIS font
graphic2_print(ST7789_DISP_BUFF_CH,0,24,"漢字とASCII",0x7BE0,"SHINONOME_FONTX") -- 16x16 SJIS + 8x16 ASCII font
graphic2_print(ST7789_DISP_BUFF_CH,0,40,"漢字とASCII",0xFE19,"AYU_FONTX") -- 20x20 SJIS + 10x20 ASCII font
-- 2倍拡大
graphic2_print(ST7789_DISP_BUFF_CH,0,60,"ABCDEF abcdef",0xFFFF,"FREEPIXEL_FONTX",2) -- 8x16 ASCII font
graphic2_print(ST7789_DISP_BUFF_CH,0,92,"漢字とASCII",0xF800,"MISAKI_FONTX",2) -- 8x8 SJIS font
graphic2_print(ST7789_DISP_BUFF_CH,0,108,"漢字とASCII",0xF81F,"SHINONOME_FONTX",2) -- 16x16 SJIS + 8x16 ASCII font
graphic2_print(ST7789_DISP_BUFF_CH,0,140,"漢字とASCII",0x867D,"AYU_FONTX",2) -- 20x20 SJIS + 10x20 ASCII font
ST7789_display()
```

上記スクリプトの実行例

- **使用例２(AQM1248A GraphicLCD)**

```
fontx_try_init()
AQM1248A_try_init()
local c_white = 1
graphic2_clear(AQM1248A_DISP_BUFF_CH)
graphic2_print(AQM1248A_DISP_BUFF_CH,0,0,"ABCDEF abcdef",c_white,"FREEPIXEL_FONTX") -- 8x16 ASCII font
graphic2_print(AQM1248A_DISP_BUFF_CH,0,18,"漢字とASCII",c_white,"MISAKI_FONTX") -- 8x8 SJIS font
```

```
graphic2_print(AQM1248A_DISP_BUFF_CH,0,30,"漢字とASCII",c_white,"SHINONOME_FONTX") -- 16x16 SJIS + 8x16 ASCII font
AQM1248A_display()
```

上記スクリプトの実行例

30.5 graphic2_draw_pixel()

- **機能概要**

  1ピクセルをディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_pixel(channel, x, y ,color *[,page]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	ピクセルの表示座標(X軸) を指定する。
y:Number	ピクセルの表示座標(Y軸) を指定する。
color:Number	ピクセル位置に描画するカラー値。
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **使用例**

```
-- HSV => RGB変換ルーチン(H:0～360,SとVは最大値固定)
function hsv_rgb(h)
  h = math.fmod(h,360)
  if (h >= 0) and (h < 60) then
    return 255,math.floor(255*(h/60)),0
  elseif (h >= 60) and (h < 120) then
    return math.floor(255*((120-h)/60)),255,0
  elseif (h >= 120) and (h < 180) then
    return 0,255,math.floor(255*((h-120)/60))
  elseif (h >= 180) and (h < 240) then
    return 0,math.floor(255*((240-h)/60)),255
  elseif (h >= 240) and (h < 300) then
    return math.floor(255*((h-240)/60)),0,255
  elseif (h >= 300) and (h < 360) then
    return 255,0,math.floor(255*((360-h)/60))
```

```
  end
end
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local x0 = 120
local y0 = 120
local r = 100
for t = 0,1000 do
  r = r - 0.1
  local x = x0+r*math.cos(math.rad(t))
  local y = y0+r*math.sin(math.rad(t))
  graphic2_draw_pixel(ST7789_DISP_BUFF_CH,x,y,rgb565(hsv_rgb(t)))
end
ST7789_display()
```

上記スクリプトの実行例

30.6 graphic2_draw_line()

- **機能概要**

  線分をディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_line(channel, x0, y0, x1, y1 ,color *[,page]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x0:Number	線分の開始座標(X軸) を指定する。
y0:Number	線分の開始座標(Y軸) を指定する。
x1:Number	線分の終了座標(X軸) を指定する。
y1:Number	線分の終了座標(Y軸) を指定する。
color:Number	描画するカラー値。
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

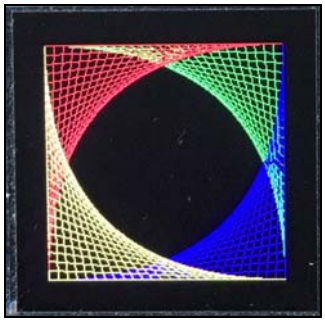
- **備考**

  線分の開始座標と終了座標の区別はありません。

- **使用例**

```
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local x0 = 20
local y0 = 20
local size = 200
for t = 0,size,8 do
  graphic2_draw_line(ST7789_DISP_BUFF_CH,x0+t,y0,x0,y0+size-t,rgb565(255,0,0))
  graphic2_draw_line(ST7789_DISP_BUFF_CH,x0+t,y0,x0+size,y0+t,rgb565(0,255,0))
  graphic2_draw_line(ST7789_DISP_BUFF_CH,x0+t,y0+size,x0+size,y0+size-t,rgb565(0,0,255))
  graphic2_draw_line(ST7789_DISP_BUFF_CH,x0,y0+t,x0+t,y0+size,rgb565(255,255,0))
end
ST7789_display()
```

上記スクリプトの実行例



30.7 graphic2_draw_rect()

- **機能概要**

  四角形をディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_rect(channel, x, y, width, height, color *[,fill [,round_r [,page]]]*)

- **パラメータとリターン値**

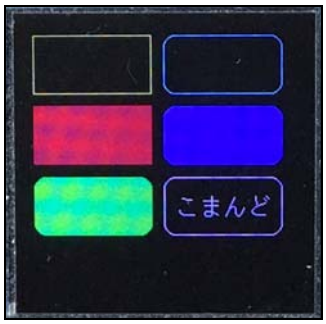
stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	四角形の左下座標(X軸) を指定する。
y:Number	四角形の左下座標(Y軸) を指定する。
width:Number	四角形の横方向のピクセル数を指定する。
height:Number	四角形の縦方向のピクセル数を指定する。

color:Number	描画するカラー値。
fill:Boolean	四角形内部を塗りつぶす場合に true を指定する。 パラメータ省略時または false を指定した場合には輪郭のみを描画する
round_r:Number	四角形の角を円弧にする場合(角丸四角形)の半径のピクセル数を指定する。 パラメータ省略時または 0 指定時は角を円弧にしない。
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **使用例**

```
fontx_try_init()
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local w = 100
local h = 50
local r = 10
graphic2_draw_rect(ST7789_DISP_BUFF_CH,10,10,w,h,rgb565(255,255,0))
graphic2_draw_rect(ST7789_DISP_BUFF_CH,10,70,w,h,rgb565(255,0,0),true)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,10,130,w,h,rgb565(0,255,0),true,r)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,120,10,w,h,rgb565(100,255,255),false,r)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,120,70,w,h,rgb565(50,155,255),true,r)

graphic2_draw_rect(ST7789_DISP_BUFF_CH,120,130,w,h,rgb565(255,255,255),false,r)
graphic2_print(ST7789_DISP_BUFF_CH,130,145,'こまんど',rgb565(255,255,255),'AYU_FONTX')
ST7789_display()
```

上記スクリプトの実行例



30.8 graphic2_draw_circle()

- **機能概要**

円をディスプレイバッファに描画する。

- **関数定義**

stat = graphic2_draw_circle(channel, x, y, r ,color *[,fill [,page]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	円の中心座標(X軸) を指定する。
y:Number	円の中心座標(Y軸) を指定する。
r:Number	半径のピクセル数を指定する。
color:Number	描画するカラー値。
fill:Boolean	円内部を塗りつぶす場合に true を指定する。 パラメータ省略時または false を指定した場合には輪郭のみを描画する
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **使用例**

```
fontx_try_init()
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local r = 30
graphic2_draw_circle(ST7789_DISP_BUFF_CH,50,40,r,rgb565(255,255,0))
graphic2_draw_circle(ST7789_DISP_BUFF_CH,50,120,r,rgb565(255,0,0),true)
graphic2_draw_circle(ST7789_DISP_BUFF_CH,50,200,r,rgb565(0,255,0),true)
graphic2_draw_circle(ST7789_DISP_BUFF_CH,120,40,r,rgb565(100,255,255),false)
graphic2_draw_circle(ST7789_DISP_BUFF_CH,120,120,r,rgb565(50,155,255),true)

graphic2_draw_circle(ST7789_DISP_BUFF_CH,120,200,r,rgb565(255,255,255),false)
graphic2_print(ST7789_DISP_BUFF_CH,102,175,'丸',rgb565(255,255,255),'AYU_FONTX',2)
ST7789_display()
```

上記スクリプトの実行例

30.9 graphic2_draw_triangle()

- **機能概要**

三角形をディスプレイバッファに描画する。

- **関数定義**

stat = graphic2_draw_triangle(channel, x0, y0, x1, y1, x2, y2 ,color *[,fill [,page]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x0:Number	三角形の1つめの頂点座標(X軸) を指定する。
y0:Number	三角形の1つめの頂点座標(Y軸) を指定する。
x1:Number	三角形の2つめの頂点座標(X軸) を指定する。
y1:Number	三角形の2つめの頂点座標(Y軸) を指定する。
x2:Number	三角形の3つめの頂点座標(X軸) を指定する。
y2:Number	三角形の3つめの頂点座標(Y軸) を指定する。
color:Number	描画するカラー値。
fill:Boolean	三角形内部を塗りつぶす場合に true を指定する。 パラメータ省略時または false を指定した場合には輪郭のみを描画する
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **備考**

3つの頂点座標の区別はありません。

- **使用例**

```
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local r = 150
local h = r*math.sin(math.rad(60))
local x0 = 50
local y0 = 200
local x1 = x0+r/2
local y1 = y0-h
local x2 = x0+r
local y2 = y0
graphic2_draw_triangle(ST7789_DISP_BUFF_CH,x0,y0,x1,y1,x2,y2,rgb565(255,255,0),true)
local sh = (r/2)*math.sin(math.rad(60))
local sw = (r/2)*math.cos(math.rad(60))
local sx0 = x0+sw
local sy0 = y0-sh
local sx1 = x0+r-sw
local sy1 = y0-sh
local sx2 = x0+r/2
```

```
local sy2 = y0
graphic2_draw_triangle(ST7789_DISP_BUFF_CH,sx0,sy0,sx1,sy1,sx2,sy2,0,true)
ST7789_display()
```

上記スクリプトの実行例

30.10 graphic2_draw_ellipse()

- **機能概要**

  楕円をディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_ellipse(channel, x, y, rx, ry, color *[,fill [,page]]*)

- **パラメータとリターン値**

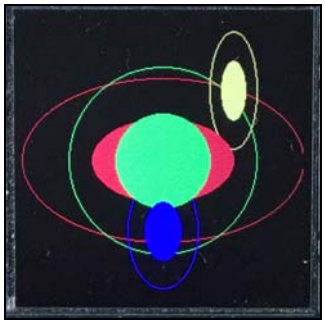
stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	楕円の中心座標(X軸) を指定する。
y:Number	楕円の中心座標(Y軸) を指定する。
rx:Number	楕円 X軸方向の半径のピクセル数を指定する。
ry:Number	楕円 Y軸方向の半径のピクセル数を指定する。
color:Number	描画するカラー値。
fill:Boolean	楕円内部を塗りつぶす場合に true を指定する。 パラメータ省略時または false を指定した場合には輪郭のみを描画する
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **使用例**

```
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,120,120,70,rgb565(255,0,0),false)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,120,80,80,rgb565(0,255,0),false)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,180,30,50,rgb565(0,0,255),false)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,180,60,20,50,rgb565(255,255,0),false)
```

```
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,120,120/2,70/2,rgb565(255,0,0),true)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,120,80/2,80/2,rgb565(0,255,0),true)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,120,180,30/2,50/2,rgb565(0,0,255),true)
graphic2_draw_ellipse(ST7789_DISP_BUFF_CH,180,60,20/2,50/2,rgb565(255,255,0),true)
ST7789_display()
```

上記スクリプトの実行例



30.11 graphic2_draw_ellipse_arc()

- **機能概要**

  楕円・円弧をディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_ellipse_arc(channel, x, y, r0x, r1x, r0y, r1y, arc_start, arc_end ,color *[,fill [,page]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	楕円の中心座標(X軸) を指定する。
y:Number	楕円の中心座標(Y軸) を指定する。
r0x:Number	円弧外側 X軸方向の半径のピクセル数を指定する。
r1x:Number	円弧内側 X軸方向の半径のピクセル数を指定する。
r0y:Number	円弧外側 Y軸方向の半径のピクセル数を指定する。
r1y:Number	円弧内側 Y軸方向の半径のピクセル数を指定する。
arc_start:Number	円弧の開始位置角度を 0 から 360 までの度数で指定する。 X 軸 プラス側方向が 0 度で、時計回りに角度が増加する。
arc_end:Number	円弧の終了位置角度を 0 から 360 までの度数で指定する。 X 軸 プラス側方向が 0 度で、時計回りに角度が増加する。
color:Number	描画するカラー値。
fill:Boolean	楕円・円弧内部を塗りつぶす場合に true を指定する。 パラメータ省略時または false を指定した場合には輪郭のみを描画する
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **使用例１**

```
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,100,70,100,70,0,90,rgb565(255,0,0),true)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,100,70,100,70,90,180,rgb565(0,255,0),true)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,100,70,100,70,180,270,rgb565(0,0,255),true)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,100,70,100,70,270,360,rgb565(255,255,0),true)

graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,50,25,50,70,0,90,rgb565(255,0,0),false)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,50,70,50,25,90,270,rgb565(0,255,0),false)
graphic2_draw_ellipse_arc(ST7789_DISP_BUFF_CH,120,120,50,0,50,0,270,360,rgb565(0,0,255),true)
ST7789_display()
```

上記スクリプトの実行例

- **使用例２**

```
--------------------------------------------------------------------------------
-- LCD(AQM1248A) に BME280 で取得した温度ゲージをアニメーション表示
--------------------------------------------------------------------------------
local center_x = 60
local center_y = 47
local outer_s = 40
local inner_s = 30
local label_font = "SHINONOME12_FONTX"
local font_size = 2
local lcd_ch = AQM1248A_DISP_BUFF_CH
function draw_tick(deg)
    graphic2_draw_ellipse_arc(lcd_ch,center_x,center_y,outer_s+5,outer_s,outer_s+5,outer_s,deg-1,deg+1,1,false,1)
end
function draw_needle(deg)
    graphic2_draw_ellipse_arc(lcd_ch,center_x,center_y,outer_s,inner_s,outer_s,inner_s,180,deg,1,true,0)
    graphic2_draw_ellipse_arc(lcd_ch,center_x,center_y,outer_s,inner_s-7,outer_s,inner_s-7,deg-2,deg+2,1,true,0)
```

```
end
fontx_try_init()
AQM1248A_try_init()
graphic2_clear(lcd_ch)
-- ゲージをバックグランドページに描画
graphic2_draw_ellipse_arc(lcd_ch,center_x,center_y,outer_s,inner_s,outer_s,inner_s,180,270,1,false,1)
-- 目盛をバックグランドページに描画
draw_tick(180)
draw_tick(180+90/4)
draw_tick(180+2*90/4)
draw_tick(180+3*90/4)
draw_tick(270)
-- ラベルをバックグランドページに描画
graphic2_print(lcd_ch,5,36,"0",1,label_font,1,0,1)
graphic2_print(lcd_ch,5,23,"10",1,label_font,1,0,1)
graphic2_print(lcd_ch,13,8,"20",1,label_font,1,0,1)
graphic2_print(lcd_ch,28,0,"30",1,label_font,1,0,1)
graphic2_print(lcd_ch,65,0,"40",1,label_font,1,0,1)
-- フォアグランドページをLCDに表示してバックグランドの内容で置き換える
AQM1248A_display(0,1)
-- 現在の温度取得
local BME280 = stat_check(script_exec2("RASPI/DEVICE/BME280_READ","",""))
local temperature = tonumber(BME280["temperature"])
temperature = temperature - 7.1 -- ボードの温度を考慮して実際の気温に修正(だいたい)
if temperature > 40 then temperature = 40 end
if temperature < 0 then temperature = 0 end
-- バックグランドの描画内容に重ねてゲージとニードルをアニメーション表示させる
local step = 20
for i = 1,step do
  local tv = temperature*i/step
  draw_needle(180+90*tv/40)
  graphic2_print(lcd_ch,50,24,string.format('%.1f℃',temperature),1,label_font,font_size,0,0)
  AQM1248A_display(0,1)
  wait_time(10)
end
```

上記スクリプトの実行例

30.12 graphic2_draw_fontx()

- **機能概要**

  ビットマップデータをディスプレイバッファに描画する。ビットマップフォント(FONTX2)形式と同じフォーマットのデータを使用します。

- **関数定義**

  stat = graphic2_draw_fontx(channel, x, y, width, height, bitmap_hexstr, color, *[,size [,page]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
x:Number	ビットマップデータ(文字)を描画する左下座標(X軸) を指定する。
y:Number	ビットマップデータ(文字)を描画する左下座標(Y軸) を指定する。
width:Number	ビットマップデータ(文字)の横方向(X軸) のピクセル数を指定する。
heightNumberg	ビットマップデータ(文字)の縦方向(Y軸) のピクセル数を指定する。
bitmap_hexstr:String	ビットマップフォントデータ(1文字分のフォントデータ)を16進数文字列で指定する。ここで指定するデータフォーマットは shmen_get_fontx() ライブラリ関数で取得できるデータ並びと同じです。ただし、数値配列から16進数文字列への変換が必要です。
color:Number	描画するカラー値。
size:Number	描画するときの拡大倍率で 1 以上の整数を指定する。 パラメータ省略時は 1 が指定される。
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画対象とするページ番号を指定する。デフォルト値:0

- **備考**

  ビットマップフォントデータのフォーマットは FONTX2 形式で指定します。

  ビットマップで表現した文字の左上が表示開始座標(x,y)になります。size パラメータを指定して拡大表示するときには、表示開始座標から右下(X,Y軸が増加する方向)に向かって拡大表示します。

  abs_agent インストールキットで提供されているデフォルトフォントを使用する場合には、フォントライブラリで提供されている初期化関数 fontx_try_init() をコールすることで共有メモリにロードできます。フォントライブラリ詳細とフォント一覧は scripts/preload/013_FONTX に格納されているファイルを参照してください。

30.13 graphic2_get_image()

- **機能概要**

  ディスプレイバッファ上の矩形範囲のカラー値配列を取得する。

- **関数定義**

  stat, image_arr = graphic2_get_image(channel, clip_x, clip_y,clip_w, clip_h *[,page]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  image_arr:Table [1..#max_item] of Number

  　　ディスプレイバッファに描画されていたカラー値配列。

  　　画像の横(x)縦(y)位置のピクセルに設定されたカラー値を (x,y) で表した時に、image_arr 配列への格納順序は (clip_x+0,clip_y+0)(clip_x+1,clip_y+0)..(clip_x+image_w-1,clip_y+0)(clip_x+0,clip_y+1),(clip_x+1,clip_y+1)..(clip_x+image_w-1,clip_y+image_h-1) になる。

  　　このとき、image_arr の配列のサイズは clip_w * clip_h に等しい。

  channel:String　　ディスプレイバッファのチャンネル名。

  clip_x:Number　　画像取得範囲の左上 X座標を指定する。

  clip_y:Number　　画像取得範囲の左上 Y座標を指定する。

  clip_w:Number　　画像取得範囲の横幅ピクセル数を指定する。

  clip_h:Number　　画像取得範囲の高さピクセル数を指定する。

  page:Number　　ディスプレイバッファが複数の描画ページをもっている場合に、取得対象とするページ番号を指定する。デフォルト値:0

- **備考**

  この関数を実行してもディスプレイバッファの内容は変化しません。

- **使用例**

```
fontx_try_init()
ST7789_try_init()
graphic2_clear(ST7789_DISP_BUFF_CH)
local c_blue = rgb565(0x1e,0x2c,0x90)
local c_white = rgb565(0xff,0xff,0xff)
local c_black = rgb565(0,0,0)
local logo_w = 230
local logo_h = 72
graphic2_draw_rect(ST7789_DISP_BUFF_CH,5,15,logo_w,logo_h,c_white,true)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,10,20,23,40,c_blue,true)
```

```
graphic2_draw_rect(ST7789_DISP_BUFF_CH,35,20,23,40,c_blue,true)
graphic2_draw_rect(ST7789_DISP_BUFF_CH,60,20,23,40,c_blue,true)
graphic2_print(ST7789_DISP_BUFF_CH,12,24,"A",c_white,"SHINONOME_FONTX",2)
graphic2_print(ST7789_DISP_BUFF_CH,37,24,"B",c_white,"SHINONOME_FONTX",2)
graphic2_print(ST7789_DISP_BUFF_CH,62,24,"S",c_white,"SHINONOME_FONTX",2)
graphic2_print(ST7789_DISP_BUFF_CH,90,23,"abs_agent",c_black,"FREEPIXEL_FONTX")
graphic2_print(ST7789_DISP_BUFF_CH,90,43,"All Blue System",c_black,"FREEPIXEL_FONTX")
graphic2_print(ST7789_DISP_BUFF_CH,15,62,"オールブルーシステム",c_black,"AYU_FONTX")
local stat,img = graphic2_get_image(ST7789_DISP_BUFF_CH,5,15,logo_w,logo_h)
graphic2_draw_image(ST7789_DISP_BUFF_CH,img,logo_w,logo_h,140,150,logo_w/2,logo_h/2,0.7,0.7,20.0)
graphic2_draw_image(ST7789_DISP_BUFF_CH,img,logo_w,logo_h,100,200,logo_w/2,logo_h/2,0.4,0.4,40.0)
ST7789_display()
```

上記スクリプトの実行例

30.14 graphic2_draw_image()

- **機能概要**

  画像のカラー値配列をディスプレイバッファに描画する。

- **関数定義**

  stat = graphic2_draw_image(channel, image_arr, image_w, image_h, dst_x, dst_y, src_x, src_y,
  zoom_x, zoom_y, angle [,a_color [,page]])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel:String　　ディスプレイバッファのチャンネル名。

  image_arr:Table [1..#max_item] of Number

  ディスプレイバッファに描画する画像のカラー値を格納した配列。

  横(x)縦(y)位置のピクセルのカラー値を (x,y) で表した時に、image_arr 配列への格納順序は(0,0)(1,0)..(image_w-1,0)(0,1),(1,1)..(image_w-1,image_h-1) にする。このとき、image_arr の配列のサイズは image_w * image_h に等しい。

image_w:Number	image_arr に指定した画像幅のピクセル数。
image_h:Number	image_arr に指定した画像高さのピクセル数。
dst_x:Number	描画するディスプレイバッファ座標(X軸) を整数または浮動少数点値で指定する。
dst_y:Number	描画するディスプレイバッファ座標(Y軸) を整数または浮動少数点値で指定する。
src_x:Number	回転・平行移動・拡大・縮小時の中心座標(X軸) を整数または浮動少数点値で指定する(0 <= src_x < image_w の値)。0 は image_arr に指定した画像の左端で、画像の中心を指定する場合は image_w/2.0 を指定する。
src_y:Number	回転・平行移動・拡大・縮小時の中心座標(Y軸) を整数または浮動少数点値で指定する(0 <= src_y < image_h の値)。0 は image_arr に指定した画像の上端で、画像の中心を指定する場合は image_h/2.0 を指定する。
zoom_x:Number	X軸方向の拡大・縮小率を整数または浮動少数点値で指定する。1.0 で等倍になる。
zoom_y:Number	Y軸方向の拡大・縮小率を整数または浮動少数点値で指定する。1.0 で等倍になる。
angle:Number	回転角度(degree)を整数または浮動少数点値で指定する。時計回りがプラス値で、反時計回りはマイナス値を指定する。0 は回転なしになる。 回転の開始位置は中心座標(src_x,src_y)から X軸プラス側(右向き方向) が 0 度で、90度で時計回りにY 軸プラス側(下向き方向)になる。
a_color:Number	画像中の透過色にするカラー値を指定する。image_arr 配列中の透過色に一致するピクセルは描画しない。省略または -1 指定時は全て描画する。
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、描画するページ番号を指定する。デフォルト値:0

- **備考**

  image_arr に指定する画像データは graphic2_get_image()ライブラリ関数で取得できます。

  このライブラリ関数内部では座標値や拡大・縮小倍率・回転角度などは全て浮動小数点型として扱われ、最終的な描画位置(整数)を計算しています。

- **使用例**

  下記のスクリプトは RASPI/DEVICE/ST7789_ANALOG_CLOCK/CLOCK.lua に格納されています。

  (無限ループに入るため別スレッドで起動してください)

```
-- 画像ファイル(RGB)をディスプレイバッファにロードした後、画像データ部分のカラー値配列を返す関数
function load_image(name,w,h,page)
  stat_check(graphic2_clear(ST7789_DISP_BUFF_CH,page))
  stat_check(graphic2_load_rgb_file(ST7789_DISP_BUFF_CH,g_dir .. '/' .. name,w,h,0,0,page))
  return stat_check(graphic2_get_image(ST7789_DISP_BUFF_CH,0,0,w,h,page))
end
local bck_w = 240              -- 文字盤画像の幅
local bck_h = 240              -- 文字盤画像の高さ
local center_x = 122.2         -- 文字盤上の針軸 x座標
```

```
local center_y = 119.6          -- 文字盤上の針軸 y座標
local short_w = 24              -- 短針画像の幅
local short_h = 71              -- 短針画像の高さ
local short_src_x = 12.06       -- 短針の回転位置 x座標（短針画像左端からの相対座標）
local short_src_y = 59.31       -- 短針の回転位置 y座標（短針画像上端からの相対座標）
local long_w = 19               -- 長針画像の幅
local long_h = 99               -- 長針画像の高さ
local long_src_x = 9.63         -- 長針の回転位置 x座標（長針画像左端からの相対座標）
local long_src_y = 90.69        -- 長針の回転位置 y座標（長針画像上端からの相対座標）
local sec_w = 19                -- 秒針画像の幅
local sec_h = 116               -- 秒針画像の高さ
local sec_src_x = 9.25          -- 秒針の回転位置 x座標（秒針画像左端からの相対座標）
local sec_src_y = 89.13         -- 秒針の回転位置 y座標（秒針画像上端からの相対座標）
local a_color = rgb565(0xff,0xff,0xff) -- 針画像中の透過色にするカラー(WHITE)
ST7789_try_init()
local short_img = load_image('短針_24x71.raw',short_w,short_h,1)-- 短針画像ファイルをロードして画像のカラー値配列を得る
local long_img = load_image('長針_19x99.raw',long_w,long_h,1) -- 長針画像ファイルをロードして画像のカラー値配列を得る
local sec_img = load_image('秒針_19x116.raw',sec_w,sec_h,1) -- 秒針画像ファイルをロードして画像のカラー値配列を得る
load_image('背景_240x240.raw',bck_w,bck_h,1)-- 文字盤画像ファイルをディスプレイバッファ中の page#1 にロードしたままにする。
local prev_sec_deg   = -1 -- 画面の描画頻度を減らすために、秒針の位置が変化しているかを調べる
while true do
  local year,mon,day,hour,minute,sec,millisec = stat_check(now_datetime())
  local hour_deg = 360 * (((hour % 12) + minute / 60) / 12)    -- 短針の角度を(時+分)の解像度で計算する。
  local min_deg = 360 * ((minute + sec / 60) / 60)             -- 長針の角度を(分+秒)の解像度で計算する。
  local sec_deg = 360 * (sec / 60)                             -- 秒針の角度を計算
  if prev_sec_deg ~= sec_deg then -- 針の位置は変化した?
    prev_sec_deg = sec_deg
    -- 文字盤が描画済みのディスプレイバッファpage#0 上に全ての針画像を描画する。
    stat_check(graphic2_draw_image(ST7789_DISP_BUFF_CH,short_img,short_w,short_h,
                                   center_x,center_y,short_src_x,short_src_y,1,1,hour_deg,a_color,0))
    stat_check(graphic2_draw_image(ST7789_DISP_BUFF_CH,long_img,long_w,long_h,
                                   center_x,center_y,long_src_x,long_src_y,1,1,min_deg,a_color,0))
    stat_check(graphic2_draw_image(ST7789_DISP_BUFF_CH,sec_img,sec_w,sec_h,
                                   center_x,center_y,sec_src_x,sec_src_y,1,1,sec_deg,a_color,0))
    -- ディスプレイバッファ page#0 の内容を LCD に転送後、
    -- 文字盤画像のみが描画されている page#1 の内容を page#0 にコピーする。
    ST7789_display(0,1)
  end
  wait_time(100) -- 更新チェック間隔
```

```
end
```

上記スクリプトの実行例

30.15 graphic2_load_rgb_file()

- **機能概要**

  RGBファイルをディスプレイバッファにロードする。

- **関数定義**

  stat = graphic2_load_rgb_file(channel, filename, width, height *[,x ,y [,page]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	ディスプレイバッファのチャンネル名。
filename:String	RGBファイル名。絶対パス名で指定します。
width:Number	画像高さのピクセル数を指定する。
height:Number	画像幅のピクセル数を指定する。
x:Number	描画するディスプレイバッファ座標(X軸) を指定する。デフォルト値:0
y:Number	描画するディスプレイバッファ座標(Y軸) を指定する。デフォルト値:0
page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、取得対象とするページ番号を指定する。デフォルト値:0

- **備考**

  filename パラメータで指定可能なファイルは、 RGB888バイナリ形式(RGB 3チャンネル・インターリーブ,チャンネル毎のデータ幅 8bit,ヘッダ無し)で格納された画像ファイルのみです。ファイルに格納されている画像の幅と高さは各々、width, height パラメータに指定します。

  ファイルからロードしたピクセル毎のRGB値は、ディスプレイバッファに設定されているピクセル深度(depth)に応じた変換を行った後、ディスプレイバッファに格納されます。

  画像はディスプレイバッファ座標(x,y) の位置にロードします。画像を回転・ズームして描画したい場合は、一旦画像をこの関数でロードした後に、graphic2_get_image(),graphic2_clear(), graphic2_draw_image() ライブラリ関数等を利用して画像のカラー配列として取り出した後、任意の位置に再描画してください。

- **使用例**

下記のスクリプトは RASPI/DEVICE/LCD_ST7789_LOGO.lua に格納されています。

```
local img_w = 240
local img_h = 50
ST7789_try_init()
stat_check(graphic2_load_rgb_file(ST7789_DISP_BUFF_CH,g_dir .. '/ABS_LOGO_240x50.raw',img_w,img_h,0,0,0))
local img_data = stat_check(graphic2_get_image(ST7789_DISP_BUFF_CH,0,0,img_w,img_h,0))
stat_check(graphic2_clear(ST7789_DISP_BUFF_CH))
stat_check(graphic2_draw_image(ST7789_DISP_BUFF_CH,img_data,img_w,img_h,120,120,img_w/2,img_h/2,2.0,2.0,45,-1,0))
stat_check(graphic2_draw_image(ST7789_DISP_BUFF_CH,img_data,img_w,img_h,120,120,img_w/2,img_h/2,1.0,1.0,-45,-1,0))
ST7789_display()
```

上記スクリプトの実行例

30.16 ST7789_display() (Raspberry Pi 専用)

- **機能概要**

ディスプレイバッファの内容を ST7789 LCD デバイスに表示する

- **関数定義**

ST7789_display([page [,reload_page]])

ST7789_display_reset([page [,ptn]])

ST7789_display_reset([page [,color]])

- **パラメータとリターン値**

page:Number　　ディスプレイバッファが複数の描画ページをもっている場合に、表示対象とするページ番号を指定する。 デフォルト値:0

reload_page　　表示後に reload_page に指定したページの内容を、page で指定したページにコピーする。

ptn:Table [1..#max_item] of Number

表示後にディスプレイバッファを指定した配列値で埋める。

color:Table [1..#max_item] of Number

表示後にディスプレイバッファを指定したカラー値で埋める。

- **備考**

このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、ST7789 コントローラを使用した LCD ディスプレイ(240x240)デバイスを接続した場合に使用することができます。他の GPIO ピンを使用する場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。下記の関数定義スクリプトの内容を参照してください。

このライブラリ関数は内部で raspi_spi_write_shmem() コマンドをコールして、ディスプレイバッファ内に描画されたピクセルデータを取得して LCD デバイスへに送信しています。

- **preload/100_RASPI/ST7789_240x240/ST7789_LIB.lua ファイル中で定義された内容**

```
ST7789_CLOCK_DIV = 8 -- SPI clock =  250MHz/<ST7789_CLOCK_DIV>
ST7789_SPI_PORT = 0
ST7789_DC_GPIO = 24
ST7789_RESET_GPIO = 25
ST7789_WIDTH = 240
ST7789_HEIGHT = 240
ST7789_DISP_BUFF_CH = "ST7789DispBuff"
ST7789_PAGE_SIZE = 2 * ST7789_WIDTH * ST7789_HEIGHT
ST7789_PAGE_MAX = 2  -- ライブラリ内で使用可能なページ(フレームバンク)数
function ST7789_data(byte)
  stat_check(raspi_gpio_write(ST7789_DC_GPIO,true)) -- set DC high for data
  stat_check(raspi_spi_write_byte(byte))
end
function ST7789_cmd(byte)
  stat_check(raspi_gpio_write(ST7789_DC_GPIO,false)) -- set DC low for command
  stat_check(raspi_spi_write_byte(byte))
end
function ST7789_setup_window()
  local x0 = 0
  local y0 = 0
  local x1 = ST7789_WIDTH - 1
  local y1 = ST7789_HEIGHT - 1
  ST7789_cmd(0x2A)   -- column addr set
  ST7789_data(bit_rshift(x0,8))
  ST7789_data(bit_and(x0,0xff))
```

```
  ST7789_data(bit_rshift(x1,8))
  ST7789_data(bit_and(x1,0xff))
  ST7789_cmd(0x2B)   -- row addr set
  ST7789_data(bit_rshift(y0,8))
  ST7789_data(bit_and(y0,0xff))
  ST7789_data(bit_rshift(y1,8))
  ST7789_data(bit_and(y1,0xff))
  ST7789_cmd(0x2C)   -- write to RAM
  stat_check(raspi_gpio_write(ST7789_DC_GPIO,true)) -- set DC high for data
end
function ST7789_display_reset(page,ptn)
  local p = 0
  if page then p = page end
  if p >= ST7789_PAGE_MAX then error() end
  -- setup drawing area
  ST7789_setup_window()
  -- draw LCD
  if ptn then
    if type(ptn) == "table" then -- reload background image
      stat_check(raspi_spi_write_shmem(ST7789_SPI_PORT,ST7789_DISP_BUFF_CH,p *
                                       ST7789_PAGE_SIZE,ST7789_PAGE_SIZE,ptn))
    else -- reload color pattern
      local fill_arr = {}
      table.insert(fill_arr,bit_rshift(bit_and(ptn,0xff00),8))
      table.insert(fill_arr,bit_and(ptn,0xff))
      stat_check(raspi_spi_write_shmem(ST7789_SPI_PORT,ST7789_DISP_BUFF_CH,p *
                                       ST7789_PAGE_SIZE,ST7789_PAGE_SIZE,fill_arr))
    end
  else
    stat_check(raspi_spi_write_shmem(ST7789_SPI_PORT,ST7789_DISP_BUFF_CH,p *
                                     ST7789_PAGE_SIZE,ST7789_PAGE_SIZE))
  end
end
function ST7789_display(page,reload_page)
  local p = 0
  if page then p = page end
  if p >= ST7789_PAGE_MAX then error() end
  -- setup drawing area
  ST7789_setup_window()
```

```
  -- draw LCD
  if reload_page then
    if reload_page >= ST7789_PAGE_MAX then error() end
    -- draw LCD and reload page
    stat_check(raspi_spi_write_shmem(ST7789_SPI_PORT,ST7789_DISP_BUFF_CH,p +
                                     ST7789_PAGE_SIZE,ST7789_PAGE_SIZE,reload_page * ST7789_PAGE_SIZE))
  else
    -- draw LCD
    stat_check(raspi_spi_write_shmem(ST7789_SPI_PORT,ST7789_DISP_BUFF_CH,p +
                                       ST7789_PAGE_SIZE,ST7789_PAGE_SIZE))
  end
end
function ST7789_init()
  stat_check(raspi_gpio_config(ST7789_DC_GPIO,"output","off"))
  stat_check(raspi_gpio_config(ST7789_RESET_GPIO,"output","off"))
  stat_check(raspi_spi_config(ST7789_SPI_PORT,true,"mode3",ST7789_CLOCK_DIV,"cs0","low"))
  stat_check(raspi_gpio_write(ST7789_RESET_GPIO,true))
  wait_time(120)
  stat_check(raspi_gpio_write(ST7789_RESET_GPIO,false))
  wait_time(120)
  stat_check(raspi_gpio_write(ST7789_RESET_GPIO,true))
  wait_time(120)
  ST7789_cmd(0x01)                          -- Software reset
  wait_time(120)
  ST7789_cmd(0x36)                          -- Memory Data Access Control
  ST7789_data(0xB0)
  --ST7789_data(0x70)
  wait_time(10)
  ST7789_cmd(0x37)                          -- Vertical Scroll Start Address of RAM
  ST7789_data(0x0)
  -- ST7789_data(320 - ST7789_WIDTH)
  ST7789_data(ST7789_WIDTH)
  ST7789_cmd(0xB2)                          -- Porch Setting
  ST7789_data(0x0C)
  ST7789_data(0x0C)
  ST7789_data(0x00)
          ST7789_data(0x33)
  ST7789_data(0x33)
  ST7789_cmd(0x3A)                          -- Interface Pixel Format
```

```
ST7789_data(0x05)                    -- 16bit/pixel RGB-5-6-5-bit (2bytes)
ST7789_cmd(0xB7)                     -- Gate Control
ST7789_data(0x14)
ST7789_cmd(0xBB)                     -- VCOMS Setting
ST7789_data(0x37)
ST7789_cmd(0xC0)                     -- LCM Control
ST7789_data(0x2C)
ST7789_cmd(0xC2)                     -- VDV and VRH Command Enable
ST7789_data(0x01)
ST7789_cmd(0xC3)                     -- VRH Set
ST7789_data(0x12)
ST7789_cmd(0xC4)                     -- VDV Set
ST7789_data(0x20)
ST7789_cmd(0xD0)                     -- Power Control 1
ST7789_data(0xA4)
ST7789_data(0xA1)
ST7789_cmd(0xC6)                     -- Frame Rate Control in Normal Mode
ST7789_data(0x0F)
ST7789_cmd(0xE0)                     -- Positive Voltage Gamma Control
ST7789_data(0xD0)
ST7789_data(0x04)
ST7789_data(0x0D)
ST7789_data(0x11)
ST7789_data(0x13)
ST7789_data(0x2B)
ST7789_data(0x3F)
ST7789_data(0x54)
ST7789_data(0x4C)
ST7789_data(0x18)
ST7789_data(0x0D)
ST7789_data(0x0B)
ST7789_data(0x1F)
ST7789_data(0x23)
ST7789_cmd(0xE1)                     -- Negative Voltage Gamma Control
ST7789_data(0xD0)
ST7789_data(0x04)
ST7789_data(0x0C)
ST7789_data(0x11)
ST7789_data(0x13)
```

```
  ST7789_data(0x2C)
  ST7789_data(0x3F)
  ST7789_data(0x44)
  ST7789_data(0x51)
  ST7789_data(0x2F)
  ST7789_data(0x1F)
  ST7789_data(0x1F)
  ST7789_data(0x20)
  ST7789_data(0x23)
  -- ST7789_cmd(0x20)                                -- Display Inversion Off
  ST7789_cmd(0x21)                          -- Display Inversion On
  ST7789_cmd(0x11)                          -- Sleep Out
  ST7789_cmd(0x29)                          -- Display On
  wait_time(100)
  graphic2_init(ST7789_DISP_BUFF_CH,ST7789_WIDTH,ST7789_HEIGHT,ST7789_PAGE_MAX,16)
end
function ST7789_try_init() -- まだデバイスが初期化されていないときにのみ初期化を行う
  local key = "ST7789_" .. "SPI" .. tostring(ST7789_SPI_PORT) .. "_INITIALIZED"
  local stat,flag = get_shared_data(key) -- 厳密な排他は行わないが充分である
  if flag ~= "1" then
    set_shared_data(key,"1")
    ST7789_init()
    stat_check(graphic2_clear(ST7789_DISP_BUFF_CH))
  end
end
```

30.17 ST7789_try_init() (Raspberry Pi 専用)

- **機能概要**

  グラフィックライブラリを ST7789 LCD デバイス用に初期設定する

- **関数定義**

  ST7789_try_init()

- **パラメータとリターン値**

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、ST7789 コントローラを使用した LCD ディスプレイ(240x240)デバイスを接続した場合に使用することができます。他の GPIO ピンを使用する場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。

このライブラリ関数は内部で graphic2_init() コマンドをコールして、ディスプレイサイズの設定を行います。また、LCD デバイス初期化のためのインストラクションコードを raspi_spi_write() ライブラリ関数を使用して送信します。

他のグラフィックライブラリ API をコールする前に、必ずこの関数を最低１度コールしてLCDデバイスの初期化とディスプレイバッファの描画エリアを指定する必要があります。このライブラリ関数を複数回コールした場合には、最初の一回目のコール時のみデバイスの初期化を行います。もし、強制的にデバイスを初期化したい場合にはこのライブラリ関数の代わりに、ST7789_init() をコールして下さい。

30.18 DISPHATMINI_display() (Raspberry Pi 専用)

- **機能概要**

  ディスプレイバッファの内容を Pimoroni製 "Display HAT Mini" LCD デバイスに表示する

- **関数定義**

  DISPHATMINI_display(*[page [,reload_page]]*)

  DISPHATMINI_display_reset(*[page [,ptn]]*)

  DISPHATMINI_display_reset(*[page [,color]]*)

- **パラメータとリターン値**

  page:Number　　ディスプレイバッファが複数の描画ページをもっている場合に、表示対象とするページ番号を指定する。 デフォルト値:0

  reload_page　　表示後に reload_page に指定したページの内容を、page で指定したページにコピーする。

  ptn:Table [1..#max_item] of Number

  　　表示後にディスプレイバッファを指定した配列値で埋める。

  color:Table [1..#max_item] of Number

  　　表示後にディスプレイバッファを指定したカラー値で埋める。

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、ST7789 コントローラを使用した Pimoroni製 "Display HAT Mini" LCD ディスプレイ(320x240)デバイスを接続した場合に使用することができます。他の GPIO ピンを使用する場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。下記の関数定義スクリプトの内容を参照してください。

  このライブラリ関数は内部で raspi_spi_write_shmem() コマンドをコールして、ディスプレイバッファ内に描画されたピクセルデータを取得して LCD デバイスへに送信しています。

- **preload/100_RASPI/displayhatmini/DISPHATMINI_LIB.lua ファイル中で定義された内容**

```
DISPHATMINI_CLOCK_DIV = 6 -- SPI clock =  250MHz/<DISPHATMINI_CLOCK_DIV>
DISPHATMINI_SPI_PORT = 0
```

```
DISPHATMINI_DC_GPIO = 9
DISPHATMINI_CS1_GPIO = 7 -- Soc 内蔵SPIモジュールによる自動アサートを使用しないで、GPIOポートを直接ライブラリから操作する。
DISPHATMINI_WIDTH = 320
DISPHATMINI_HEIGHT = 240
DISPHATMINI_DISP_BUFF_CH = "DispHATMiniBuff"
DISPHATMINI_PAGE_SIZE = 2 * DISPHATMINI_WIDTH * DISPHATMINI_HEIGHT
DISPHATMINI_PAGE_MAX = 2  -- ライブラリ内で使用可能なページ(フレームバンク)数
DISPHATMINI_LED_R = 17
DISPHATMINI_LED_G = 27
DISPHATMINI_LED_B = 22
DISPHATMINI_BUTTON_A = 5
DISPHATMINI_BUTTON_B = 6
DISPHATMINI_BUTTON_X = 16
DISPHATMINI_BUTTON_Y = 24
DISPHATMINI_BACKLIGHT = 13
function DISPHATMINI_CS_deassert()
  stat_check(raspi_gpio_write(DISPHATMINI_CS1_GPIO,true))
end
function DISPHATMINI_CS_assert()
  stat_check(raspi_gpio_write(DISPHATMINI_CS1_GPIO,false))
end
function DISPHATMINI_data(byte)
  stat_check(raspi_gpio_write(DISPHATMINI_DC_GPIO,true)) -- set DC high for data
  stat_check(raspi_spi_write_byte(byte))
end
function DISPHATMINI_cmd(byte)
  stat_check(raspi_gpio_write(DISPHATMINI_DC_GPIO,false)) -- set DC low for command
  stat_check(raspi_spi_write_byte(byte))
end
function DISPHATMINI_setup_window()
  local x0 = 0
  local y0 = 0
  local x1 = DISPHATMINI_WIDTH - 1
  local y1 = DISPHATMINI_HEIGHT - 1
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x2A)         -- column addr set
  DISPHATMINI_data(bit_rshift(x0,8))
  DISPHATMINI_data(bit_and(x0,0xff))
  DISPHATMINI_data(bit_rshift(x1,8))
```

```
  DISPHATMINI_data(bit_and(x1,0xff))
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x2B)         -- row addr set
  DISPHATMINI_data(bit_rshift(y0,8))
  DISPHATMINI_data(bit_and(y0,0xff))
  DISPHATMINI_data(bit_rshift(y1,8))
  DISPHATMINI_data(bit_and(y1,0xff))
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x2C)         -- write to RAM
  stat_check(raspi_gpio_write(DISPHATMINI_DC_GPIO,true)) -- set DC high for data
end
function DISPHATMINI_display_reset(page,ptn)
  local p = 0
  if page then p = page end
  if p >= DISPHATMINI_PAGE_MAX then error() end
  -- setup drawing area
  DISPHATMINI_setup_window()
  -- draw LCD
  if ptn then
    if type(ptn) == "table" then -- reload background image
      stat_check(raspi_spi_write_shmem(DISPHATMINI_SPI_PORT,DISPHATMINI_DISP_BUFF_CH,p *
                                       DISPHATMINI_PAGE_SIZE,DISPHATMINI_PAGE_SIZE,ptn))
    else -- reload color pattern
      local fill_arr = {}
      table.insert(fill_arr,bit_rshift(bit_and(ptn,0xff00),8))
      table.insert(fill_arr,bit_and(ptn,0xff))
      stat_check(raspi_spi_write_shmem(DISPHATMINI_SPI_PORT,DISPHATMINI_DISP_BUFF_CH,p *
                                       DISPHATMINI_PAGE_SIZE,DISPHATMINI_PAGE_SIZE,fill_arr))
    end
  else
    stat_check(raspi_spi_write_shmem(DISPHATMINI_SPI_PORT,DISPHATMINI_DISP_BUFF_CH,p *
                                     DISPHATMINI_PAGE_SIZE,DISPHATMINI_PAGE_SIZE))
  end
  DISPHATMINI_CS_deassert()
end
function DISPHATMINI_display(page,reload_page)
  local p = 0
```

```
  if page then p = page end
  if p >= DISPHATMINI_PAGE_MAX then error() end
  -- setup drawing area
  DISPHATMINI_setup_window()
  -- draw LCD
  if reload_page then
    if reload_page >= DISPHATMINI_PAGE_MAX then error() end
    -- draw LCD and reload page
    stat_check(raspi_spi_write_shmem(DISPHATMINI_SPI_PORT,DISPHATMINI_DISP_BUFF_CH,p *
                                     DISPHATMINI_PAGE_SIZE,DISPHATMINI_PAGE_SIZE,reload_page * DISPHATMINI_PAGE_SIZE))
  else
    -- draw LCD
    stat_check(raspi_spi_write_shmem(DISPHATMINI_SPI_PORT,DISPHATMINI_DISP_BUFF_CH,p *
                                     DISPHATMINI_PAGE_SIZE,DISPHATMINI_PAGE_SIZE))
  end
  DISPHATMINI_CS_deassert()
end
function DISPHATMINI_init()
  -- RGB LED
  stat_check(raspi_gpio_config(DISPHATMINI_LED_R,"output","off"))
  stat_check(raspi_gpio_config(DISPHATMINI_LED_G,"output","off"))
  stat_check(raspi_gpio_config(DISPHATMINI_LED_B,"output","off"))
  stat_check(raspi_gpio_write(DISPHATMINI_LED_R,true)) -- OFF
  stat_check(raspi_gpio_write(DISPHATMINI_LED_G,true)) -- OFF
  stat_check(raspi_gpio_write(DISPHATMINI_LED_B,true)) -- OFF
  -- SW
  stat_check(raspi_gpio_config(DISPHATMINI_BUTTON_A,"input","pullup"))
  stat_check(raspi_gpio_config(DISPHATMINI_BUTTON_B,"input","pullup"))
  stat_check(raspi_gpio_config(DISPHATMINI_BUTTON_X,"input","pullup"))
  stat_check(raspi_gpio_config(DISPHATMINI_BUTTON_Y,"input","pullup"))
  stat_check(raspi_change_detect(DISPHATMINI_BUTTON_A,true))
  stat_check(raspi_change_detect(DISPHATMINI_BUTTON_B,true))
  stat_check(raspi_change_detect(DISPHATMINI_BUTTON_X,true))
  stat_check(raspi_change_detect(DISPHATMINI_BUTTON_Y,true))
  -- LCD Backlight
  stat_check(raspi_gpio_config(DISPHATMINI_BACKLIGHT,"output","off"))
  stat_check(raspi_gpio_write(DISPHATMINI_BACKLIGHT,true)) -- ON
  stat_check(raspi_spi_config(DISPHATMINI_SPI_PORT,true,"mode0",DISPHATMINI_CLOCK_DIV,"reserved","low")) -- SoC 内蔵の CS 機
能を無効にする
```

```
  stat_check(raspi_gpio_config(DISPHATMINI_CS1_GPIO,"output","off"))      -- raspi_spi_config() の SPI#0-CE をオーバーライドする
  stat_check(raspi_gpio_write(DISPHATMINI_CS1_GPIO,true))                 -- Deassert Chip Select
  stat_check(raspi_gpio_config(DISPHATMINI_DC_GPIO,"output","off"))       -- raspi_spi_config() の SPI#0-MISO をオーバーライドする
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x01)                                   -- Software reset
  DISPHATMINI_CS_deassert()
  wait_time(150)
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x36)                                   -- Memory Data Access Control
  DISPHATMINI_data(0xB0)
  DISPHATMINI_CS_deassert()
  wait_time(10)
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0xB2)                                   -- Frame rate ctrl - idle mode
  DISPHATMINI_data(0x0C)
  DISPHATMINI_data(0x0C)
  DISPHATMINI_data(0x00)
  DISPHATMINI_data(0x33)
  DISPHATMINI_data(0x33)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x3A)                                   -- Interface Pixel Format
  DISPHATMINI_data(0x05)                                  -- 16bit/pixel RGB-5-6-5-bit (2bytes)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0xB7)                                   -- Gate Control
  DISPHATMINI_data(0x14)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0xBB)                                   -- VCOMS Setting
  DISPHATMINI_data(0x37)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0xC0)                                   -- LCM Control
  DISPHATMINI_data(0x2C)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
```

```
DISPHATMINI_cmd(0xC2)                   -- VDV and VRH Command Enable
DISPHATMINI_data(0x01)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xC3)                   -- VRH Set
DISPHATMINI_data(0x12)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xC4)                   -- VDV Set
DISPHATMINI_data(0x20)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xD0)                   -- Power Control 1
DISPHATMINI_data(0xA4)
DISPHATMINI_data(0xA1)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xC6)                   -- Frame Rate Control in Normal Mode
DISPHATMINI_data(0x0F)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xE0)                   -- Positive Voltage Gamma Control
DISPHATMINI_data(0xD0)
DISPHATMINI_data(0x04)
DISPHATMINI_data(0x0D)
DISPHATMINI_data(0x11)
DISPHATMINI_data(0x13)
DISPHATMINI_data(0x2B)
DISPHATMINI_data(0x3F)
DISPHATMINI_data(0x54)
DISPHATMINI_data(0x4C)
DISPHATMINI_data(0x18)
DISPHATMINI_data(0x0D)
DISPHATMINI_data(0x0B)
DISPHATMINI_data(0x1F)
DISPHATMINI_data(0x23)
DISPHATMINI_CS_deassert()
DISPHATMINI_CS_assert()
DISPHATMINI_cmd(0xE1)                   -- Negative Voltage Gamma Control
```

```
  DISPHATMINI_data(0xD0)
  DISPHATMINI_data(0x04)
  DISPHATMINI_data(0x0C)
  DISPHATMINI_data(0x11)
  DISPHATMINI_data(0x13)
  DISPHATMINI_data(0x2C)
  DISPHATMINI_data(0x3F)
  DISPHATMINI_data(0x44)
  DISPHATMINI_data(0x51)
  DISPHATMINI_data(0x2F)
  DISPHATMINI_data(0x1F)
  DISPHATMINI_data(0x1F)
  DISPHATMINI_data(0x20)
  DISPHATMINI_data(0x23)
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x21)                          -- Display Inversion On
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x11)                          -- Sleep Out
  DISPHATMINI_CS_deassert()
  DISPHATMINI_CS_assert()
  DISPHATMINI_cmd(0x29)                          -- Display On
  DISPHATMINI_CS_deassert()
  wait_time(100)
  graphic2_init(DISPHATMINI_DISP_BUFF_CH,DISPHATMINI_WIDTH,DISPHATMINI_HEIGHT,DISPHATMINI_PAGE_MAX,16)
end
function DISPHATMINI_try_init() -- まだデバイスが初期化されていないときにのみ初期化を行う
  local key = "DISPHATMINI_" .. "SPI" .. tostring(DISPHATMINI_SPI_PORT) .. "_INITIALIZED"
  local stat,flag = get_shared_data(key) -- 厳密な排他は行わないが充分である
  if flag ~= "1" then
    set_shared_data(key,"1")
    DISPHATMINI_init()
    stat_check(graphic2_clear(DISPHATMINI_DISP_BUFF_CH))
  end
end
```

30.19 DISPHATMINI_try_init() (Raspberry Pi 専用)

- **機能概要**

  グラフィックライブラリを Pimoroni製 "Display HAT Mini" LCD デバイス用に初期設定する

- **関数定義**

  DISPHATMINI_try_init()

- **パラメータとリターン値**

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、ST7789 コントローラを使用した Pimoroni製 "Display HAT Mini" LCD ディスプレイ(320x240)デバイスを接続した場合に使用することができます。他の GPIO ピンを使用する場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。

  このライブラリ関数は内部で graphic2_init() コマンドをコールして、ディスプレイサイズの設定を行います。また、LCD デバイス初期化のためのインストラクションコードを raspi_spi_write() ライブラリ関数を使用して送信します。

  他のグラフィックライブラリ API をコールする前に、必ずこの関数を最低１度コールしてLCDデバイスの初期化とディスプレイバッファの描画エリアを指定する必要があります。このライブラリ関数を複数回コールした場合には、最初の一回目のコール時のみデバイスの初期化を行います。もし、強制的にデバイスを初期化したい場合にはこのライブラリ関数の代わりに、DISPHATMINI_init() をコールして下さい。

  Pimoroni製 "Display HAT Mini" 用 LCDドライバでは、デフォルトの Chip Select 1 信号を使用しないでソフト側からGPIO制御しています。また、SPI#0 に自動アサインされる MISO ピンを Data/Command 出力として使用するために、raspi_spi_config() で設定されたコンフィギュレーション値を上書きしています。この時、SoC SPI#0 コントロールレジスタの CS(ChipSelect)フィールドに "reserved" (0x3) を指定して SoCからの干渉を防止しています。詳細は前項のドライバソースを参照して下さい。

30.20 AQM1248A_display() (Raspberry Pi 専用)

- **機能概要**

  ディスプレイバッファの内容を AQM1248A LCD デバイスに表示する

- **関数定義**

  AQM1248A_display([page [,reload_page]])

- **パラメータとリターン値**

page:Number	ディスプレイバッファが複数の描画ページをもっている場合に、表示対象とするページ番号を指定する。 デフォルト値:0
reload_page	表示後に reload_page に指定したページの内容を、page で指定したページにコピ

ーする。

ptn:Table [1..#max_item] of Number

表示後にディスプレイバッファを指定した配列値で埋める。

color:Table [1..#max_item] of Number

表示後にディスプレイバッファを指定したカラー値で埋める。

- **備考**

このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、秋月電子製 AQM1248A LCD デバイスを接続した場合に使用することができます。LCD デバイスのコントロールコマンドとピクセルデータを区別するためのピンは、Raspberry Pi GPIO#25 に接続していることを想定しています。通信条件や使用する GPIO ピンを変更したい場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。

このライブラリ関数は内部で raspi_spi_write_shmem() コマンドをコールして、ディスプレイバッファ内に描画されたピクセルデータを取得して LCD デバイスへに送信しています。

- **preload/100_RASPI/AQM1248A_GRAPHIC2/AQM1248A_LIB.lua ファイル中で定義された内容**

```
-------------------------------------------------------------------------------------------------
-- AQM1248A (128x48 pixel) 秋月電子製 LCD ディスプレイ出力用ライブラリ関数(graphic2 ライブラリ版)
-------------------------------------------------------------------------------------------------
AQM1248A_SPI_PORT = 0
AQM1248A_RS_PIN = 25
AQM1248A_CLOCK_DIV = 25 -- spi clock: 250MHz/25 = 10MHz clock
AQM1248A_WIDTH = 128
AQM1248A_HEIGHT = 48
AQM1248A_DISP_BUFF_CH = "AQM1248ADispBuff"
AQM1248A_PAGE_SIZE = AQM1248A_WIDTH * AQM1248A_HEIGHT / 8
AQM1248A_PAGE_MAX = 2           -- ライブラリ内で使用可能なページ(フレームバンク)数
function AQM1248A_cmd(cmd)
  stat_check(raspi_gpio_write(AQM1248A_RS_PIN,false))
  stat_check(raspi_spi_write_byte(AQM1248A_SPI_PORT,cmd))
end
-------------------------------------------------------------------------------------------------
-- ディスプレイバッファ中の指定ページ page の内容を LCD RAM に書き込む
-- reload_page 指定時は、表示後に reload_page に指定した内容で page パラメータで指定したメモリ値を上書きする
-------------------------------------------------------------------------------------------------
function AQM1248A_display(page,reload_page)
  local p = 0
  if page then p = page end
  if p >= AQM1248A_PAGE_MAX then error() end
```

```
  local y_max_idx = math.floor(AQM1248A_HEIGHT/8) - 1
  local offset = 0
  for y_idx = 0,y_max_idx,1 do
    stat_check(raspi_gpio_write(AQM1248A_RS_PIN,false)) -- Instructions taransfer
    stat_check(raspi_spi_write(AQM1248A_SPI_PORT,bit_tohex(0xb0 + y_idx,2) .. "1000")) -- set page/column address
    stat_check(raspi_gpio_write(AQM1248A_RS_PIN,true))  -- Display RAM transfer
    stat_check(raspi_spi_write_shmem(AQM1248A_SPI_PORT,AQM1248A_DISP_BUFF_CH,p * AQM1248A_PAGE_SIZE +
                                     offset,AQM1248A_WIDTH)) -- transfer bitmap data
    offset = offset + AQM1248A_WIDTH
  end
  if reload_page then
    if reload_page >= AQM1248A_PAGE_MAX then error() end
    stat_check(shmem_move(AQM1248A_DISP_BUFF_CH,reload_page * AQM1248A_PAGE_SIZE,p *
                          AQM1248A_PAGE_SIZE,AQM1248A_PAGE_SIZE))
  end
end

function AQM1248A_init()
  log_msg("initializing AQM1248A device, port = ".. tostring(AQM1248A_SPI_PORT),g_script)
  -- Raspberry Pi SPI#0 configulation
  stat_check(raspi_spi_config(true,"mode0",AQM1248A_CLOCK_DIV,"cs0","low"))
  -- AQM1248A initialize
  AQM1248A_cmd(0xAE) -- LCD Display off
  AQM1248A_cmd(0xA0) -- ADC select normal
  AQM1248A_cmd(0xC8) -- common output mode reverse direction
  AQM1248A_cmd(0xA3) -- LCD bias set 1/7
  AQM1248A_cmd(0x2C) -- power control 1
  wait_time(2)
  AQM1248A_cmd(0x2E) -- power control 2
  wait_time(2)
  AQM1248A_cmd(0x2F) -- power control 3
  AQM1248A_cmd(0x23) -- Vo voltage regulator internal resistor ratio set
  AQM1248A_cmd(0x81) -- Electronic volume mode set
  AQM1248A_cmd(0x1C) -- Electronic volume value set
  AQM1248A_cmd(0xA4) -- display all point normal
  AQM1248A_cmd(0x40) -- display start line = 0
  AQM1248A_cmd(0xA6) -- display normal
  AQM1248A_cmd(0xAF) -- LCD Display on
  ------------------------------
```

```
  -- ディスプレイバッファ初期化
  ------------------------------
  graphic2_init(AQM1248A_DISP_BUFF_CH,AQM1248A_WIDTH,AQM1248A_HEIGHT,AQM1248A_PAGE_MAX,-1)
end
function AQM1248A_try_init() -- まだデバイスが初期化されていないときにのみ初期化を行う
  local name = "AQM1248A_" .. tostring(AQM1248A_SPI_PORT) .. "_INITIALIZED"
  local flag = stat_check(get_shared_data(name)) -- 厳密な排他は行わないが充分である
  if flag ~= "1" then
    stat_check(set_shared_data(name,"1"))
    AQM1248A_init()
  end
end
```

30.21 AQM1248A_try_init() (Raspberry Pi 専用)

- **機能概要**

  グラフィックライブラリを AQM1248A LCD デバイス用に初期設定する

- **関数定義**

  AQM1248A_try_init()

- **パラメータとリターン値**

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの SPI#0 インターフェイスに、秋月電子製 AQM1248A LCD デバイスを接続した場合に使用することができます。LCD デバイスのコントロールコマンドとピクセルデータを区別するためのピンは、Raspberry Pi GPIO#25 に接続していることを想定しています。他の GPIO ピンを使用する場合には、preload フォルダ中に格納されたライブラリ関数の内容を修正してください。関数定義の詳細は、AQM1248A_display() 関数の項を参照してください。

  このライブラリ関数は内部で graphic2_init() コマンドをコールして、ディスプレイサイズの設定を行います。また、LCD デバイス初期化のためのインストラクションコードを raspi_i2c_write() ライブラリ関数を使用して送信します。

  他のグラフィックライブラリ API をコールする前に、必ずこの関数を最低１度コールしてLCDデバイスの初期化とディスプレイバッファの描画エリアを指定する必要があります。このライブラリ関数を複数回コールした場合には、最初の一回目のコール時のみデバイスの初期化を行います。もし、強制的にデバイスを初期化したい場合にはこのライブラリ関数の代わりに、AQM1284A_init() をコールして下さい。

30.22 OLED1306_display() (Raspberry Pi 専用)

- **機能概要**

  ディスプレイバッファの内容を SSD1306 コントローラ使用 OLED デバイスに表示する

- **関数定義**

  OLED1306_display(*[page [,reload_page]]*)

- **パラメータとリターン値**

  page:Number　　ディスプレイバッファが複数の描画ページをもっている場合に、表示対象とするページ番号を指定する。 デフォルト値:0

  reload_page　　表示後に reload_page に指定したページの内容を、page で指定したページにコピーする。

  ptn:Table [1..#max_item] of Number

  　　表示後にディスプレイバッファを指定した配列値で埋める。

  color:Table [1..#max_item] of Number

  　　表示後にディスプレイバッファを指定したカラー値で埋める。

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの I2Cバスに OLED(128x64 pixel) SSD1306 コントローラデバイスを接続した場合に使用することができます。I2C バス番号やクロックスピード等の設定値の確認や変更を行いたい場合には、preload フォルダ中に格納された関数定義スクリプトの内容を参照してください。

  このライブラリ関数は内部で raspi_i2c_write_shmem() コマンドをコールして、ディスプレイバッファ内に描画されたピクセルデータを取得して LCD デバイスへに送信しています。

- **preload/100_RASPI/OLED_SSD1306_GRAPHIC2/OLED_SSD1306_128x64.lua ファイル中で定義された内容**

```
-------------------------------------------------------------------------------
-- OLED(128x64 pixel) SSD1306 コントローラ用ライブラリ関数(graphic2 ライブラリ版)
-------------------------------------------------------------------------------
OLED1306_ADDR = "3C"
OLED1306_CLOCK_DIV = 375 -- i2c clock: 150MHz/375 = 400KHz clock
OLED1306_I2C_BUS = 1
OLED1306_WIDTH = 128
OLED1306_HEIGHT = 64
OLED1306_DISP_BUFF_CH = "OLD1306DispBuff"
OLED1306_PAGE_SIZE = OLED1306_WIDTH * OLED1306_HEIGHT / 8
OLED1306_PAGE_MAX = 2           -- ライブラリ内で使用可能なページ(フレームバンク)数
-------------------------------------------------------------------------------
-- ディスプレイバッファ中の指定ページ page の内容を LCD RAM に書き込む
-- reload_page 指定時は、表示後に reload_page に指定した内容で page パラメータで指定したメモリ値を上書きする
```

```
--------------------------------------------------------------------------------------------------------------

function OLED1306_display(page,reload_page)
  local p = 0
  if page then p = page end
  if p >= OLED1306_PAGE_MAX then error() end
  local y_max_idx = math.floor(OLED1306_HEIGHT/8) - 1
  local offset = 0
  for y_idx = 0,y_max_idx,1 do
    stat_check(raspi_i2c_write(OLED1306_I2C_BUS,OLED1306_ADDR,"00" .. bit_tohex(0xb0 + y_idx,2) .. "1000")) -- set
page/column address
    stat_check(raspi_i2c_write_shmem(OLED1306_I2C_BUS,OLED1306_ADDR,OLED1306_DISP_BUFF_CH,p *
                                     OLED1306_PAGE_SIZE + offset,OLED1306_WIDTH,"40")) -- transfer bitmap data
    offset = offset + OLED1306_WIDTH
  end
  if reload_page then
    if reload_page >= OLED1306_PAGE_MAX then error() end
      stat_check(shmem_move(OLED1306_DISP_BUFF_CH,reload_page * OLED1306_PAGE_SIZE,p *
                            OLED1306_PAGE_SIZE,OLED1306_PAGE_SIZE))
    end
end
function OLED1306_init()
  log_msg("initializing OLED1306 device, bus = " .. tostring(OLED1306_I2C_BUS),g_script)
  -- Raspberry Pi I2C clock configulation
  stat_check(raspi_i2c_clock(OLED1306_I2C_BUS,OLED1306_CLOCK_DIV)) -- 150MHz / 375 = 400KHz clock
  -- SSD1306 initialize
  local data = ""
  data = data .. "00"                           -- start command stream
  data = data .. "AE"                           -- display off
  data = data .. "D580"                 -- set display clock divide ratio/oscillator frequency to 0x80
  data = data .. "A83F"                 -- set multiplex ratio  to 0x3F (OLED1306_HEIGHT(64) - 1)
  data = data .. "D300"                 -- set display offset to 0
  data = data .. "40"                           -- set display start line 0
  data = data .. "8D14"                 -- set enable charge pump regulator to 0x14
  data = data .. "2002"                 -- set page addressing mode
  --data = data .. "A0"                 -- set segment column 0 to SEG0
  data = data .. "A1"                           -- set segment column 127 to SEG0
  --data = data .. "C0"                 -- COM output scan from COM0
  data = data .. "C8"                           -- COM output scan to COM0
  data = data .. "817F"                 -- contrast to 0x7F
```

```
  data = data .. "A4"                              -- disable entire display
  data = data .. "A6"                              -- set normal display
  data = data .. "2E"                              -- deactivate scroll
  data = data .. "21007F"                 -- column range 0..127
  data = data .. "220007"                 -- page range 0..7
  data = data .. "AF"                              -- display on
  stat_check(raspi_i2c_write(OLED1306_I2C_BUS,OLED1306_ADDR,data))
  ------------------------------
  -- ディスプレイバッファ初期化
  ------------------------------
  graphic2_init(OLED1306_DISP_BUFF_CH,OLED1306_WIDTH,OLED1306_HEIGHT,OLED1306_PAGE_MAX,-1)
end
function OLED1306_try_init() -- まだデバイスが初期化されていないときにのみ初期化を行う
  local name = "OLED1306_" .. tostring(OLED1306_I2C_BUS) .. "_INITIALIZED"
  local flag = stat_check(get_shared_data(name)) -- 厳密な排他は行わないが充分である
  if flag ~= "1" then
    stat_check(set_shared_data(name,"1"))
    OLED1306_init() -- OLED(128x64 pixel) SSD1306初期化
  end
end
```

30.23 OLED1306_try_init() (Raspberry Pi 専用)

- **機能概要**

  グラフィックライブラリを SSD1306 コントローラ使用 OLED デバイス用に初期設定する

- **関数定義**

  OLED1306_try_init()

- **パラメータとリターン値**

- **備考**

  このライブラリ関数は Raspberry Pi ハードウエアの I2Cバスに、OLED(128x64 pixel) SSD1306 コントローラデバイスを接続した場合に使用することができます。関数定義の詳細は、raspi_OLED1306_display() 関数の項を参照してください。

  このライブラリ関数は内部で graphic2_init() コマンドをコールして、ディスプレイサイズの設定を行います。また、LCD デバイス初期化のためのインストラクションコードを raspi_i2c_write() ライブラリ関数を使用して送信します。

  他のグラフィックライブラリ API をコールする前に、必ずこの関数を最低１度コールしてLCDデバイスの初期

化とディスプレイバッファの描画エリアを指定する必要があります。このライブラリ関数を複数回コールした場合には、最初の一回目のコール時のみデバイスの初期化を行います。もし、強制的にデバイスを初期化したい場合にはこのライブラリ関数の代わりに、OLED1306_init() をコールして下さい。

31 時系列集計用インメモリデータベース API

abs_agent はセンサーから取得した計測値や A/D変換値等の数値データを集計用のインメモリデータベース(以降、このマニュアルでは "FASTDB" と呼びます)に登録して、一定期間毎の集計値を簡単に計算することができます。登録データにはデータ毎に任意のキー値を指定することができます。

共通のキー名で登録されたデータは、時系列で並べられたデータ列として扱われ集計計算で利用できます。abs_agent のライブラリ関数を使用して、複数のキー名を使用したデータ列を同時に幾つも作成することができます。センサーデバイスや計測器ごとに任意のキー名を付けて計測データデータを登録することができます。その後、センサーデバイス毎の時系列データ表やグラフ等を簡単に作成することができます。

FASTDB のデータ保存やデータアクセス時には、メモリ領域のみを使用して高速に動作します。FASTDBを使用する場合には ファイルシステムへの読み書きが発生しないので、フラッシュメモリ等のファイルシステム上で OS や abs_agent を動作させている場合でも、ストレージに負荷をかけることなく長期間安定して運用することができます。

FASTDB では任意のキー文字列と計測値を示す数値(整数や実数) を fastdb_add()ライブラリ関数を使用して登録します。登録したデータは集計用の fastdb_summary() ライブラリ関数を使用して指定した期間ごとの平均値などを簡単に取り出すことができます。データ登録時には abs_agent が動作しているコンピュータのシステム時刻がデフォルトのタイムスタンプ値として使用されます。このとき、タイムスタンプ値に任意の時刻を指定することも可能です。

FASTDB はメモリ中にデータを保持していますので、abs_agent プログラムを停止した場合には全てのデータは破棄されます。クライアントコマンド agent_fastdb を使用するとデータベースの内容をファイルに書き出したり、ファイルからデータを登録することができます。

FASTDB で保存しているデータはオンラインでグローバル共有メモリエリアにアーカイブしたり、反対にアーカイブされたアーカイブデータを元に FASTDB データレコードとしてリストアすることができます。agent_fastdb コマンドを使用すると、これらの機能とグローバル共有メモリエリアのリモート送・受信機能を併用することで、FASTDB レコードデータをオンラインでリモートコンピュータ上にバックアップ・リストアすることができます。詳しくはクライアントコマンド agent_fastdb の項を参照して下さい。

ここで紹介しているライブラリ関数は全て、マルチスレッドの環境で確実に操作できるように内部でリソースの管理が行われています。このため、イベントハンドラやユーザースクリプトからスレッド間の競合を意識することなく使用できます。

データベースロックについて

abs_agent では FASTDB のデータを更新したり集計する場合に、内部でキー毎にデータの更新を自動でロックしてデータの一貫性を保っています。あるキーに対してデータベースがロック中に、同一のキーによるデータ登録を可能にする為に、FASTDB 内部に２つのデータベース(PrimaryDBと TemporaryDB) を持っています。これらのデータベースはライブラリ関数内で自動的に使用されます。通常はデフォルトのイベントハンドラスクリプトによってこれらのデータベースは自動的にデータ同期されますので、ユーザーは特にデータベースロックを意識しなくてもデータの更新や集計、削除をマルチスレッド下で操作できます。

31.1 fastdb_add(), fastdb_add_net()

- **機能概要**

  FASTDB データベースにデータを登録する。

- **関数定義**

  stat = fastdb_add(key, value *[,timestamp]*)

  stat = fastdb_add_net(key, value *[,remote_host [,timestamp]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
key:String	データベースに登録するときの Key 文字列
value:Number	データベースに登録するデータ値。(Double 値で保存します)
timestamp:String	タイムスタンプ値を "YYYY/MM/DD HH:MM:SS"または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き) の形式で指定する。パラメータを省略した場合には、登録時のサーバー時刻が設定される。
remote_host:String	リモートコンピュータのホスト名または IP アドレスを指定する。パラメータを省略した場合や "" 空文字列を指定した場合には、サーバー設定ファイル中の設定項目 "/Document/ServiceMain/DefaultRemoteHost" に指定した値が使用される。

- **備考**

  key に指定する文字列を変えることで、複数の時系列データを同時に扱うことができます。

  タイムスタンプ値をミリ秒まで指定する場合には、各項目の桁数をゼロ埋めで合わせてください。ミリ秒を指定しない場合には、タイムスタンプ値のミリ秒部分は ".000" になります。

  リモートで動作している abs_agent の FASTDB にデータを追加する場合には fastdb_add_net() を使用できます。リモート側の abs_agent では予め、API をコールするコンピュータへのアクセス許可を与えておく必要があります。リモートコンピュータに対する API の詳しい説明はグローバル共有変数 API の項の "リモート共有ライブラリ関数 xxxxx_net_data() について"を参照してください。

- **使用例**

```
if not fastdb_add("KeyName1",1.234) then error() end
```

31.2 fastdb_purge()

- **機能概要**

  指定した日付よりも古いデータを削除する。

- **関数定義**

  stat = fastdb_purge(key [,timestamp_until])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  key:String　　削除対象のキー名

  timestamp_until:String　削除対象のタイムスタンプ値

  指定した日付時刻よりも古いデータレコードを削除する。

  タイムスタンプ値を "YYYY/MM/DD HH:MM:SS"または "YYYY/MM/DD HH:MM:SS.mmm"（ミリ秒付き）の形式で指定する。

  パラメータ省略時は key に一致する全レコードが削除される

- **備考**

  指定した条件に一致する FASTDB データを全て削除します。

- **使用例**

```
if not fastdb_purge("KEY1", "2002/1/1 0:0:0") then error() end
```

  上記の例ではタイムスタンプ値が 2001 年を含んだそれよりも古いデータを全て削除します。

31.3 fastdb_summary()

- **機能概要**

  FASTDB データベースに登録されているデータの集計計算を行う。

- **関数定義**

  stat, datetime, sample, total, mean, max, min, first =

  fastdb_summary(key, datetime_start, interval, count)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  datetime:Table [1..#count] of String　　集計期間の最初の日付時刻

  'YYYY/MM/DD HH:MM:SS' の形式

  sample:Table [1..#count] of Number　　集計期間中に見つかったデータ個数(整数値)

  total:Table [1..#count] of Number　　集計期間中に見つかったデータの合計値(Double値)

  mean:Table [1..#count] of Number　　集計期間中に見つかったデータの平均値(Double値)

  max:Table [1..#count] of Number　　集計期間中に見つかったデータの最大値(Double値)

  min:Table [1..#count] of Number　　集計期間中に見つかったデータの最小値(Double値)

  first:Table [1..#count] of Number　　集計期間中に見つかった最初の(最も古い)値(Double値)

key:String　　集計対象のキー名

datetime_start:String　　集計対象となるデータのタイムスタンプの開始日付時刻を指定する
'YYYY/MM/DD HH:MM:SS' の形式で指定する

interval:Number　　集計対象データの時間間隔(秒) を指定する (1 以上の整数値)

count:Number　　集計データの個数を指定する (1 以上の整数値)

- **備考**

例 : 2001/1/1 の 1 時間毎の集計値を計算する場合には、パラメータを以下の値に設定する。

key	<fastdb_add()コール時に指定した任意のキー文字列>
datetime_start	"2001/01/01 0:0:0"
interval	3600
count	24

集計結果の配列を参照するときに sample[n] が 0 の値を持つ集計期間は、集計対象となるデータが存在しないことを表します。このとき、その集計期間の total[n], mean[n], max[n], min[n], first[n] の値は強制的に 0 になっていますが、これらの 0 はデータ値や統計値としての意味を持たないので注意してください。

- **使用例**

```
stat,datetime,sample,total,mean,max,min = fastdb_summary("KEY","2001/01/01 0:0:0",3600,24)
if not stat then error() end
for key,val in ipairs(datetime) do
  log_msg(string.format("summary[%d] datefrom = %s total= %f
              ",key,val,tostring(total[key])),file_id)
end
```

31.4 fastdb_key_list(), fastdb_temp_key_list()

- **機能概要**

FASTDB データベース中で使用しているキー名と、そのキーを使用して登録されているデータ個数を取得する。

- **関数定義**

stat, key_list [,record_count_list] = fastdb_key_list([locked_only]) -- PrimaryDB

stat, key_list [,record_count_list] = fastdb_temp_key_list([locked_only]) -- TemporaryDB

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

key_list:Table [1..#count] of String
FASTDBデータベースに登録されているキー名リスト

record_count_list:Table [1..#count] of Number
そのキーを使用したデータレコード数リスト

locked_only:Boolean	内部で排他使用中とマークされたキーのみを取得する。 デフォルト値 false true を指定した場合には record_count_list はリターン値に返りません。

- **備考**

FASTDB データベースで現在使用中のキー名リストを取得する。

PrimaryDB と TemporaryDB のデータベースに対して各々のロック状態を取得できます。

- **使用例**

```
stat,key_list,rec_count_list = fastdb_key_list()
if not stat then error() end
for key,val in ipairs(key_list) do
  log_msg(string.format("key = %s record count = %d",val,rec_count_list[key]),g_script)
end
```

31.5 fastdb_unlock(), fastdb_temp_unlock()

- **機能概要**

FASTDBで排他ロックとマークされているキーをアンロックする。

- **関数定義**

```
stat = fastdb_unlock(key) -- PrimaryDB
stat = fastdb_temp_unlock(key) -- TemporaryDB
```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
key:String	アンロック対象のキー名

- **備考**

何らかの原因で FASTDB の特定のキーが排他使用中のままロックされて、データの追加や検索などができなくなった場合に、強制的にロック状態を解除するときに使用します。このライブラリ関数は abs_agent 側で自動的に使用されますので、通常はユーザがコールする必要はありません。

key で指定したキーが排他ロック中でなかった場合には、この関数はなにも行いません。

PrimaryDB と TemporaryDB のデータベースに対して各々のロック状態を解除できます。

- **使用例**

インストール直後は定期的に下記の（FASTDB_FREE_LOCK.lua）スクリプトがコールされていて、長期間ロック

中のキーを強制的にアンロックしています。必要に応じてこのスクリプトをカストマイズしてください。

(FASTDB_FREE_LOCK スクリプトの内容)

```
-- ２重起動防止用チェック
if not exclusive_check(g_script) then
  log_msg("*ERROR* exclusive_check() failed. script = " .. g_script,g_script)
  return
end
local lock_timestamp_prefix = '$FASTDB_LOCK_'
local lock_timeout = 60
if g_params["lock_timeout"] then
  lock_timeout = tonumber(g_params["lock_timeout"])
end
-- ロック中のキー名を使用したフラグ(グローバル共有変数)に保存するタイムスタンプ文字列を作成
local now = os.date "*t"
local timestamp =
string.format("%4.4d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d",now["year"],now["month"],now["day"],now["hour"],now["min"],now["sec"])
-- 全キー名のテーブルのインデックスに使用したテーブルを作成。テーブルの値にはロック中かどうかを示すフラグを格納する
local key_stat = {}
local key_list
key_list = stat_check(fastdb_key_list()) -- 全キー取得
for key,val in ipairs(key_list) do
  key_stat[val] = "free"
end
key_list = stat_check(fastdb_key_list(true)) -- ロック中のキーのみ取得
for key,val in ipairs(key_list) do
  key_stat[val] = "lock"
end
-- FASTDB で使用中のキーに対して下記の処理を実行
for key,val in pairs(key_stat) do
  if val == "free" then
    -- 今回コールするよりも前にロック済みにマークされて、タイムスタンプを保存していた場合には削除する
    if not set_shared_data(lock_timestamp_prefix .. key,"") then error() end
  else
    local old_timestamp = stat_check(get_shared_data(lock_timestamp_prefix .. key))
    if old_timestamp ~= "" then
      -- 今回コールするよりも前にロック済みだった場合にはタイムスタンプが lock_timeout 以上経過していないかをチェックする
      local o_year,o_month,o_day,o_hour,o_min,o_sec = stat_check(str_to_datetime(old_timestamp))
      local elapsed_sec = stat_check(seconds_between(o_year,o_month,o_day,o_hour,o_min,o_sec,
```

```
                        now["year"],now["month"],now["day"],now["hour"],now["min"],now["sec"]))
      if elapsed_sec > lock_timeout then -- lock_timeout 以上経過したロック中のキーを強制的にアンロックする
        log_msg("*WARNING* force unlock: " .. key,g_script)
        stat_check(fastdb_unlock(key))
        stat_check(set_shared_data(lock_timestamp_prefix .. key,""))
      end
    else
      -- 今回コールするよりも前にはロックされていなかったので新規にタイムスタンプを保存する
      stat_check(set_shared_data(lock_timestamp_prefix .. key,timestamp))
    end
  end
end
```

上記のスクリプトを定期的にコールする部分は下記のようになっています。

(PERIODIC_TIMER イベントハンドラスクリプトの抜粋)

```
-----------------------------------------------------------------
-- FASTDB で排他ロック中のキーが一定期間以上存在しつづけている場合に、
-- 強制的にロック状態を解除する
-----------------------------------------------------------------
if not script_fork_exec("FASTDB_FREE_LOCK","lock_timeout","60") then error() end
```

31.6 fastdb_search()

- **機能概要**

  FASTDB データベースに登録されているデータを検索する。

- **関数定義**

  stat, value_list, timestamp_list, break =

  fastdb_search(key, datetime_from, datetime_until *[,descend [,maxlist]]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  value_list:Table [1..#count] of Number

  検索で見つかったデータの値

  timestamp_list:Table [1..#count] of String

  検索で見つかったデータのタイムスタンプ値

  'YYYY/MM/DD HH:MM:SS' の文字列形式

  break:Boolean　　検索結果が maxlist に指定した値(デフォルトは 5000) を超えた場合には、true がセットされます。この場合にはリターン値には maxlist 分のデータが格納されます。それ以外の場合には false が入ります。

key:String	検索対象のキー名
datetime_from:String	検索集計対象となるデータのタイムスタンプの開始日付時刻を指定する。 指定した日時を含むそれ以降のタイムスタンプを持つデータを検索対象にします。 パラメータを使用しない場合には空文字列 “” を指定する。空文字列を指定した場合には FASTDB に登録されている最も古いデータから検索します。 "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。
datetime_until:String	検索集計対象となるデータのタイムスタンプの終了日付時刻を指定する。指定した日付時刻を含まない直前までのデータが検索対象となる パラメータを使用しない場合には空文字列 “” を指定する。空文字列を指定した場合には FASTDB に登録されている最も新しいデータまでを検索します。 "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm"(ミリ秒付き)形式で指定する。ミリ秒は秒の後ろに小数点 "." に続けて常に3桁ゼロ埋めで指定する。
descend:Boolean	データのタイムスタンプを降順(タイムスタンプ値が新しいものから)に検索する場合には true を指定する。(パラメータ省略時デフォルト) false を指定した場合には、昇順(タイムスタンプ値が古いものから)に検索して結果を value_list, timestamp_listに格納する。このとき、テーブルに格納される順序もここで指定したソート順になります。
maxlist:Number	検索結果の最大レコード数を指定する。このパラメータを省略した場合には 5000 がデフォルトで指定されます。検索条件に合致するデータレコード数が maxlist に指定した数よりも多かった場合には break には true が返ります。

- **備考**

例1：キー名が “TEST” の直近のデータを 100 レコード分取得する。

key	“TEST”
datetime_from	“”
datetime_until	“”
descend	true
maxlist	100

(検索結果で得られるレコードデータは降順にテーブルに格納されます)

例2：キー名が “TEST” の 2010/1/1 以降のデータを 100 レコード分取得する。

key	“TEST”
datetime_from	“2010/1/1 0:0:0”
datetime_until	“”
descend	false
maxlist	100

例3：キー名が “TEST” の 2010/1/1 から 2010/1/2 までの二日分のデータを全て取得する。

key	“TEST”
datetime_from	“2010/1/1 0:0:0”
datetime_until	“2010/1/3 0:0:0” （日付に注意!）
descend	false
maxlist	省略、または予測されるデータレコード数を指定

- **使用例**

```
stat,value,timestamp = fastdb_search(“KEY”,””,””,100)
if not stat then error() end
for key,val in ipairs(value) do
  log_msg(string.format("data[%d] = %f ",key,val),file_id)
end
```

31.7 fastdb_sync()

- **機能概要**

TemporaryDB に登録済みのデータを PrimaryDB に移動する。

- **関数定義**

stat, record_count = fastdb_sync(key)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
record_count:Number	TemporaryDB から PrimaryDB に移動したレコード数
key:String	再登録対象のキー名

- **備考**

TemporaryDB 中の指定したキーのデータを全て PrimaryDB に再登録する。再登録後にデータは TemporaryDB から削除される。

指定したキーがロック中の場合や対象データが存在しない場合にはこの関数は何も行いません。
この時のリターン値は stat = true, record_count = 0 になります。

- **使用例**

インストール直後は定期的に下記の（FASTDB_SYNC.lua）スクリプトがコールされていて、自動的に TemporaryDB に登録済みのデータを PrimaryDB に移動させます。

(FASTDB_SYNC スクリプトの内容)

```
-- ２重起動防止用チェック
if not exclusive_check(g_script) then
```

```
  log_msg("*ERROR* exclusive_check() failed. script = " .. g_script,g_script)
  return
end
-----------------------------------------------------------------
-- TemporaryDB で長期間ロック中のキーを強制的にアンロックする
-----------------------------------------------------------------
local lock_timestamp_prefix = '$FASTDB_TEMP_LOCK_'
local lock_timeout = 10

-- ロック中のキー名を使用したフラグ(グローバル共有変数)に保存するタイムスタンプ文字列を作成
local now = os.date "*t"
local timestamp =
string.format("%4.4d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d",now["year"],now["month"],now["day"],now["hour"],now["min"],now["sec"])
-- 全キー名のテーブルのインデックスに使用したテーブルを作成。テーブルの値にはロック中かどうかを示すフラグを格納する
local key_stat = {}
local key_list = stat_check(fastdb_temp_key_list()) -- 全キー取得
for key,val in ipairs(key_list) do
  key_stat[val] = "free"
end
local key_list = stat_check(fastdb_temp_key_list(true)) -- ロック中のキーのみ取得
for key,val in ipairs(key_list) do
  key_stat[val] = "lock"
end
-- TemporaryDB で使用中のキーに対して下記の処理を実行
for key,val in pairs(key_stat) do
  if val == "free" then
    -- 今回コールするよりも前にロック済みにマークされて、タイムスタンプを保存していた場合には削除する
    stat_check(set_shared_data(lock_timestamp_prefix .. key,""))
  else
    local old_timestamp = stat_check(get_shared_data(lock_timestamp_prefix .. key))
    if old_timestamp ~= "" then
      -- 今回コールするよりも前にロック済みだった場合にはタイムスタンプが lock_timeout 以上経過していないかをチェックする
      local o_year,o_month,o_day,o_hour,o_min,o_sec = stat_check(str_to_datetime(old_timestamp))
      local elapsed_sec = stat_check(seconds_between(o_year,o_month,o_day,o_hour,o_min,o_sec,
                          now["year"],now["month"],now["day"],now["hour"],now["min"],now["sec"]))
      if elapsed_sec > lock_timeout then -- lock_timeout 以上経過したロック中のキーを強制的にアンロックする
       log_msg("*WARNING* force temp-unlock: " .. key,g_script)
        stat_check(fastdb_temp_unlock(key))
        stat_check(set_shared_data(lock_timestamp_prefix .. key,""))
```

```
        end
      else
        -- 今回コールするよりも前にはロックされていなかったので新規にタイムスタンプを保存する
        stat_check(set_shared_data(lock_timestamp_prefix .. key,timestamp))
      end
    end
  end
  -- TemporaryDB に登録されたデータを PrimaryDB に移動
  local key_list,record_count_list = stat_check(fastdb_temp_key_list()) -- 全キーとレコード数取得
  for i,v in ipairs(key_list) do
    local key_name = key_list[i]
    if record_count_list[i] > 0 then
      local moved_record_count = stat_check(fastdb_sync(key_name))
      log_msg("fastdb_sync key:" .. key_name .. " record:" .. tostring(moved_record_count) .. "/" .. record_count_list[i],g_script)
    end
  end
```

上記のスクリプトを定期的にコールする部分は下記のようになっています。

(TICK_TIMER イベントハンドラスクリプトの抜粋)

```
-- FASTDB TemporaryDB のデータ同期 (10秒毎に実行)
local timer = "FASTDB_SYNC_TIMER"
local val = stat_check(inc_shared_data(timer))
if (tonumber(val) >= 10) then
  stat_check(set_shared_data(timer,""))
  stat_check(script_fork_exec("FASTDB_SYNC",{}))
end
```

31.8 fastdb_backup_shmem(), fastdb_restore_shmem()

- **機能概要**

  指定したキーの FASTDBデータをグローバル共有メモリエリアにバックアップする。

  バックアップしたデータを元に、指定したキーの FASTDBデータとしてリストアする。

- **関数定義**

  stat = fastdb_backup_shmem(key, channel)

  stat = fastdb_restore_shmem(key, channel *[,adj_sec]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  key:String　　バックアップまたはリストア対象の FASTDBキー名

channel:String　　グローバル共有メモリエリアのチャンネル名

adj_sec:Number　　リストア時にタイムスタンプ値を補正する秒数を指定する。単位は秒で小数点以下３桁(ミリ秒)まで指定可。プラス値はバックアップデータ中のタイムスタンプ値に補正分を足して未来にずらし、マイナス値は時刻を過去にずらす。
パラメータ省略時または 0 を指定するとタイムスタンプ値の補正は行わない。

- **備考**

バックアップ時は、key で指定した FASTDB データレコードを全てグローバル共有メモリエリアに保存します。

リストア時はグローバル共有メモリエリアに保存したバックアップデータを元に、FASTDB データレコードとしてロードします。この時 key で指定された既存の FASTDBデータレコードは全て破棄されます。リストア時に指定する key はバックアップ時に指定した key と異なっていても構いません。

グローバル共有メモリエリアに保存するデータは、FASTDBデータレコード中のタイムスタンプ値(64bit)とDouble型データ値(64bit) のバイナリ形式です。このときFASTDB 1レコードあたり 16バイトの容量を使用します。

- **使用例1 バックアップ**

```
stat_check(fastdb_backup_shmem("KEY1", "MyBackupCh"))
```

- **使用例2 リストア**

```
stat_check(fastdb_restore_shmem("KEY1", "MyBackupCh"))
```

31.9　古いデータを削除する

FASTDB に登録された過去の古いデータは fastdb_purge() ライブラリ関数を使用して FASTDB データベースから取り除くことができます。abs_agent のインストールキットには fastdb_purge()ライブラリ関数を使用して FASTDB 中の古いデータを削除するスクリプト FASTDB_PURGE.lua が提供されています。このスクリプトを PERIODIC_TIMER イベントハンドラからコールすると簡単に古いデータを自動削除する機能を実現することができます。

(FASTDB_PURGE スクリプトの内容)

```
--[[
●機能概要
FASTDB に登録されている古いデータを削除する
●参照するグローバル共有データ
--------------------------------------------------------------------------------
キー値　　値
--------------------------------------------------------------------------------
$FASTDB_PURGE_BEFORE　　"10"
```

```
            現在日からこの日数分過去の日付よりも古いデータを削除する
            グローバル共有データが未設定の場合には "10" (10日) を使用
]]
local days_before = 10
local val = stat_check(get_shared_data("$FASTDB_PURGE_BEFORE"))
if val ~= "" then days_before = tonumber(val) end
-- 削除対象日付を取得
local now = os.date "*t"
local yyyy,mm,dd = stat_check(inc_day(-1 * days_before,now["year"],now["month"],now["day"]))
local timestamp = string.format("%4.4d/%2.2d/%2.2d 0:0:0",yyyy,mm,dd)
log_msg("data purge until " .. timestamp)
-- 全キーに対して fastdb_purge() 実行
local key_list,rec_list = stat_check(fastdb_key_list())
for key,val in ipairs(key_list) do
  stat_check(fastdb_purge(val,timestamp))
end
```

インストール直後の PERIODIC_TIMER イベントハンドラ中には下記のスクリプトが記述されています。 FASTDB 中の全てのキーのデータ中でデフォルトで 10 日以上経過したデータは 8 時間毎のタイミングで自動削除されます。実際にデータを削除するスクリプトは上記の FASTDB_PURGE スクリプトを使用しています。

(PERIODIC_TIMER イベントハンドラスクリプトの抜粋)

```
-----------------------------------------------------------------
-- FASTDB データベースから過去のデータを削除する(8時間に一回実行)
-----------------------------------------------------------------
local timer = "FASTDB_PURGE_TIMER"
local val = stat_check(inc_shared_data(timer))
if (tonumber(val) >= 480) then
  stat_check(set_shared_data(timer,""))
  if not script_fork_exec("FASTDB_PURGE","","") then error() end
end
```

abs_agent が動作するコンピュータで使用可能なメモリーサイズや FASTDB データを利用したい期間に合わせて保存期間を変更できます。例えばデータ保存期間を 30日に変更する場合には FASTDB_PURGE スクリプト中で参照するグローバル共有変数 "$FASTDB_PURGE_BEFORE" に "30" を設定します。abs_agent 起動時にこの変数を自動設定するために SERVER_START イベントハンドラに下記の記述を追加します。

(SERVER_START イベントハンドラスクリプトに追加する内容)

```
-- FASTDB の過去保存期間を 30日に設定(default 10日)
```

```
set_shared_data("$FASTDB_PURGE_BEFORE","30")
```

⚠ コンピュータのメモリー・リソースに注意

FASTDB を使用する場合には、abs_agent が動作するコンピュータのメモリー・リソースに注意してください。Linux コマンドの "free -m" コマンドで現在利用可能なメモリサイズを確認できます。もし、リソースが不足する場合にはコンピュータの物理メモリを増設するか、上記のデータ保存期間を短く設定する様にしてください。abs_agent が動作するコンピュータでメモリが逼迫すると、OS のプロセスによって "abs_agent" プロセスが強制終了される場合があります。

32 XBee 802.15.4, XBee-ZB Zigbee API

シリアルデバイスに登録した XBee 802.15.4 デバイスや XBee-ZB Zigbee デバイスと同一 PAN 内にあるリモート側 XBee, XBee-ZB デバイスを操作するためのライブラリ関数です。

abs_agent のシリアルポートに接続されたローカル側の XBee, XBee-ZB デバイスを予めシリアルデバイスタイプ "XBEE" または "ZB" で登録しておきます。
また、ローカルとリモート側の全ての XBee, XBee-ZB デバイスは全て `API Mode = 1` で動作させてください。
詳しい設定方法については、"XBee 802.15.4 デバイス接続" または "XBee-ZB Zigbee デバイス接続"の章を参照してください。

ライブラリ関数名の先頭が xbee_xxxxxx() で始まるものは XBee 802.15.4 デバイス用で、zb_xxxxxx() のものは XBee-ZB Zigbee デバイス用のライブラリ関数です。

32.1 xbee_rebuild_master(), zb_rebuild_master()

- **機能概要**

  XBee, XBee-ZB デバイスリストを保持しているマスターファイルを最新の状態に更新する。

- **関数定義**

  ```
  stat = xbee_rebuild_master(com)
  stat = zb_rebuild_master(com)
  ```

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
com:String	シリアルデバイスに "XBEE" または "ZB" タイプで登録済みの COMPort名またはタイトル名

- **備考**

  abs_agent のシリアルポートに接続されたローカルの XBee, XBee-ZBモジュールで "Node Discover"または

"Network Discovery" コマンドを実行した後、受信したフレームデータを元にマスターファイルを作成・更新する。

更新後のマスターファイルの内容は xbee_all_list() または zb_all_list() ライブラリ関数で取得できます。

abs_agent に接続しているローカルの XBee, XBee-ZB デバイスについても AT コマンドで受信した情報を元に同様にマスターファイルに登録します。

マスターファイル上に既に登録済みのデバイス情報は更新されて、このライブラリ関数をコール時に見つからなかった(無線モジュールに電源が供給されていなかった場合など)デバイスはそのままマスターファイルに残されます。

- **使用例**

```
local com = "/dev/ttyUSB0"
local stat = xbee_rebuild_master(com)
if not stat then error() end
```

32.2 xbee_all_list(), zb_all_list()

- **機能概要**

xbee_rebuild_master() ライブラリ関数によって、探索・登録された周辺の XBee デバイスリストを取得する。

- **関数定義**

stat, addr64, addr16, node_id, is_local = xbee_all_list(com)

stat, addr64, addr16, node_id, dev_type, digi_type, is_local = zb_all_list(com)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

addr64:Table [1..#max_device] of String

　　XBee, XBee-ZBデバイスのSerialNumber(64bitアドレス) 16進数表記の文字列

addr16:Table [1..#max_device] of String

　　XBee, XBee-ZBデバイスのSourceAddress, Network Address(16bitアドレス) 16進数表記の文字列

node_idname:Table [1..#max_device] of String

　　XBee, XBee-ZB デバイスの Node Identification 文字列

dev_type:Table [1..#max_device] of String

　　XBee-ZB デバイスのデバイスのデバイスタイプ文字列で次の何れかが入る。

　　"coordinator", "router", "end device"

digi_type:Table [1..#max_device] of String

XBee-ZB デバイスの DeviceTypeID(32bit)16進数表記の文字列

is_local:Table [1..#max_device] of boolean

XBee, XBee-ZB デバイスが abs_agent のCOM ポートに接続されたローカルデバイスであるかどうかを示す

com:String　シリアルデバイスに "XBEE" または "ZB" タイプで登録済みの COMPort名またはタイトル名

- **備考**

xbee_all_list(), zb_all_list() では、周辺デバイスの探索や新規登録は行いません、これらの操作が必要な場合は、xbee_rebuild_master() ライブラリ関数をコールするか、クライアントプログラム "agent_xbee" または "agent_zb" プログラムを "-m" オプション付きで実行してください。

マスターに登録された XBeeデバイスの 16bit address, 64bit address, Node Identification文字列とそのデバイスが com パラメータで指定した、abs_agent にシリアル接続されたローカルデバイスであるかを示すフラグを返します。XBee-ZB デバイスの場合には Device Type文字列と Digi Device Type ID も同時に返します。

abs_agent のシリアルポートに複数の(ローカル)XBee, XBee-ZB デバイスを登録して、それぞれ別の PAN_ID や同一の PAN_ID を設定して、それぞれの XBee, XBee-ZB 経由でリモートデバイスを操作することもできます。このとき、xbee_rebuild_master(), zb_rebuild_master() で登録されるデバイスマスターは共通のものを使用しています。このため、xbee_all_list(com), zb_all_list() をコールするときに指定した com デバイスに接続中の XBee, XBee-ZB の PAN_ID 以外のデバイスもリターン値に返ります。

- **使用例**

```
local com = "/dev/ttyUSB0"
local stat, addr64, addr16, node_id, is_local = xbee_all_list(com)
if not stat then error() end
for i,v in ipairs(addr64) do
    log_msg("addr64: " .. addr64[i] .. " addr16: " .. addr16[i] .." node_id: " .. node_id[i] .."
        is_local: " .. tostring(is_local[i]),g_script)
end
```

32.3 xbee_find_device(), zb_find_device()

- **機能概要**

指定したXBeeデバイスの Source Address(16bit)とSerialNumber(64bit), NodeIdentification 文字列を取得する。XBee-ZB の場合には Device Type文字列 と Digi Device Type ID も同時に取得する。

- **関数定義**

stat, addr64, addr16, node_id, is_local = xbee_find_device(com, dev)

stat, addr64, addr16, node_id, dev_type, digi_type, is_local = zb_find_device(com, dev)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
com:String	シリアルデバイスに "XBEE"または "ZB" タイプで登録済みの COMPort名またはタイトル名
dev:String	検索対象のXBeeデバイスを指定する。Device文字列には以下のいずれかを指定する。 XBee, XBee-ZBデバイスの Node Identification 文字列 XBee, XBee-ZBデバイスの Source Address(16bitアドレス)16進数表記の文字列 XBeeデバイスの Serial Number(64bitアドレス)16進数表記の文字列 *注意* XBee-ZB デバイスの Network Address(16bitアドレス)は dev に指定できません。
addr16:String	XBee, XBee-ZBデバイスの Source Address(16bitアドレス)16進数表記の文字列
addr64:String	XBee, XBee-ZBデバイスの SerialNumber(64bitアドレス)16進数表記の文字列
node_id:String	XBee, XBee-ZBデバイスの Node Identification 文字列
dev_type:String	XBee-ZB デバイスのデバイスのデバイスタイプ文字列で次の何れかが入る。 "coordinator", "router", "end device"
digi_type:String	XBee-ZB デバイスの DeviceTypeID(32bit)16進数表記の文字列
is_local:Boolean	XBee, XBee-ZBデバイスが abs_agent のCOM ポートに接続されたローカルデバイスであるかどうかを示す

- **備考**

パラメータに指定された XBeeデバイスの 16bit address, 64bit address, Node Identification文字列とそのデバイスが abs_agent にシリアル接続されたローカルデバイスであるかを示すフラグを返す。XBee-ZB デバイスの場合には Device Type文字列と Digi Device Type ID も同時に返します。

xbee_rebuid_master(), zb_rebuild_master() でマスターに登録済みの XBee デバイスリストを検索して該当するデバイス情報を取得します。dev に指定した文字列は、マスター中の Node Identification に一致するものがあるかどうかを先に検索します。一致するものが見つからなかった場合には、デバイスアドレス(16bit,64bit)を順に検索します。

- **使用例**

```
stat,addr16,serial,name,localdevice = xbee_find_device("Device1")
```

32.4 xbee_at_command(), zb_at_command()

- **機能概要**

指定したXBeeデバイスに任意の ATコマンドを送信する。

- **関数定義**

stat, frame [,mframe] = xbee_at_command(com, dest, atcmd *[,param [,option]]*)

stat, frame [,mframe] = zb_at_command(com, dest, atcmd *[,param [,option]]*)

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

com:String　　シリアルデバイスに "XBEE"または "ZB" タイプで登録済みの COMPort名またはタイトル名

frame:String　　コマンド実行後に受信したレスポンスフレームデータ。16進数表記の文字列

mframe:Table [1..#max_multi_frame] of String

マルチフレームを構成するレスポンスフレームを受信した場合に、マルチフレーム部分のデータが16進数文字列の配列に格納される。mframe にはマルチフレーム部分のみが格納されて、実行結果ステータスを含むレスポンスフレームは同時に frame に返る。Remote AT Command APIではマルチフレームを受信しないので常に nil になります

dest:String　　送信先のXBee, XBee-ZBデバイスを指定する。dest文字列には以下のいずれかを指定する。

XBee, XBee-ZB デバイスの Node Identification 文字列

XBee, XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列

XBee デバイスの Source Address(16bitアドレス)16進数表記の文字列

“*” ブロードキャストアドレス

“” 空文字列は、abs_agent に接続したXBee, XBee-ZB デバイスが対象

注意

XBee-ZB デバイスの Network Address(16bitアドレス)は dev に指定できません。

atcmd:String　　AT コマンド文字列(2 文字の大文字ASCII)

param:String　　AT コマンドパラメータ。16進数表記の文字列で指定する。パラメータ省略時や "" 空文字列を指定すると、コマンドパラメータは送信されない。

option:String　　リモートXBee, XBee-ZB デバイスに送信するコマンドオプションデータ。16進数表記の文字列で指定する。このパラメータを省略した場合や"" 空文字列を指定した場合には "02" (0x02: Apply Change on remote) が指定される。

- **備考**

dest の指定によって、自動的にリモートコマンドフレームまたは、コマンドフレームを使用して AT コマンドフレームを送信します。全ての ATコマンドは、abs_agent の COM ポートに接続されたローカル XBee, XBee-ZB デバイスを経由して送信され、レスポンスフレームを受信します。

option パラメータを "00" にして AT コマンドを実行した場合には、続けて ATコマンド "AC" を送信してパラメータの変更をコミットしてください。

AT コマンドパラメータを指定してリモート側デバイスの設定値を変更した場合に、設定内容を不揮発メモリに

保存するための AT コマンド "WR" を必要に応じて実行してください。

- **制限事項**

  現在、マルチフレーム受信が可能なコマンドは、ローカル XBee, XBee-ZB デバイス に対する "ND" コマンドのみです。

- **使用例**

```
local com = "/dev/ttyUSB0"
function debug_frame_dump(stat,frame,mframe)
  if not stat then error() end
  log_msg("-- frame --",g_script)
  log_hexdump(frame)
  if mframe then
    for k,v in ipairs(mframe) do
      log_msg("-- mframe#" .. k .. " --",g_script)
      log_hexdump(v)
    end
  end
end
debug_frame_dump(xbee_at_command(com,"","NI"))
debug_frame_dump(xbee_at_command(com,"Device2","NI"))
debug_frame_dump(xbee_at_command(com,"Device5","NI"))
debug_frame_dump(xbee_at_command(com,"","ND"))
```

上記のコマンド実行時ログ

```
2019/06/13 13:45:25 raspberryp TEST/XBEE_API_TEST   0 -- frame --
2019/06/13 13:45:25 raspberryp LUA Script           0  length = 16 bytes
2019/06/13 13:45:25 raspberryp LUA Script           0
2019/06/13 13:45:25 raspberryp LUA Script           0  data[000]-[007] 7E 00 0C 88 C7 4E 49 00
2019/06/13 13:45:25 raspberryp LUA Script           0                  ~             N  I
2019/06/13 13:45:25 raspberryp LUA Script           0  data[008]-[00F] 44 65 76 69 63 65 35 94
2019/06/13 13:45:25 raspberryp LUA Script           0                  D  e  v  i  c  e  5
2019/06/13 13:45:25 raspberryp TEST/XBEE_API_TEST   0 -- frame --
2019/06/13 13:45:25 raspberryp LUA Script           0  length = 26 bytes
2019/06/13 13:45:25 raspberryp LUA Script           0
2019/06/13 13:45:25 raspberryp LUA Script           0  data[000]-[007] 7E 00 16 97 CB 00 13 A2
2019/06/13 13:45:25 raspberryp LUA Script           0                  ~
2019/06/13 13:45:25 raspberryp LUA Script           0  data[008]-[00F] 00 40 4A C3 97 0B 02 4E
2019/06/13 13:45:25 raspberryp LUA Script           0                     @  J             N
2019/06/13 13:45:25 raspberryp LUA Script           0  data[010]-[017] 49 00 44 65 76 69 63 65
```

```
2019/06/13 13:45:25 raspberryp LUA Script          0                I    D  e  v  i  c  e
2019/06/13 13:45:25 raspberryp LUA Script          0  data[018]-[019] 32 DE
2019/06/13 13:45:25 raspberryp LUA Script          0                2
2019/06/13 13:45:26 raspberryp TEST/XBEE_API_TEST  0 -- frame --
2019/06/13 13:45:26 raspberryp LUA Script          0  length = 16 bytes
2019/06/13 13:45:26 raspberryp LUA Script          0
2019/06/13 13:45:26 raspberryp LUA Script          0  data[000]-[007] 7E 00 0C 88 E8 4E 49 00
2019/06/13 13:45:26 raspberryp LUA Script          0                ~            N  I
2019/06/13 13:45:26 raspberryp LUA Script          0  data[008]-[00F] 44 65 76 69 63 65 35 73
2019/06/13 13:45:26 raspberryp LUA Script          0                D  e  v  i  c  e  5  s
2019/06/13 13:45:29 raspberryp TEST/XBEE_API_TEST  0 -- frame --
2019/06/13 13:45:29 raspberryp LUA Script          0  length = 9 bytes
2019/06/13 13:45:29 raspberryp LUA Script          0
2019/06/13 13:45:29 raspberryp LUA Script          0  data[000]-[007] 7E 00 05 88 33 4E 44 00
2019/06/13 13:45:29 raspberryp LUA Script          0                ~          3  N  D
2019/06/13 13:45:29 raspberryp LUA Script          0  data[008]-[008] B2
2019/06/13 13:45:29 raspberryp LUA Script          0
2019/06/13 13:45:29 raspberryp TEST/XBEE_API_TEST  0 -- mframe#1 --
2019/06/13 13:45:29 raspberryp LUA Script          0  length = 28 bytes
2019/06/13 13:45:29 raspberryp LUA Script          0
2019/06/13 13:45:29 raspberryp LUA Script          0  data[000]-[007] 7E 00 18 88 33 4E 44 00
2019/06/13 13:45:29 raspberryp LUA Script          0                ~          3  N  D
2019/06/13 13:45:29 raspberryp LUA Script          0  data[008]-[00F] 0C 03 00 13 A2 00 40 4A
2019/06/13 13:45:29 raspberryp LUA Script          0                                   @  J
2019/06/13 13:45:29 raspberryp LUA Script          0  data[010]-[017] C3 98 3D 44 65 76 69 63
2019/06/13 13:45:29 raspberryp LUA Script          0                      =  D  e  v  i  c
2019/06/13 13:45:29 raspberryp LUA Script          0  data[018]-[01B] 65 33 00 49
2019/06/13 13:45:29 raspberryp LUA Script          0                e  3     I
2019/06/13 13:45:29 raspberryp TEST/XBEE_API_TEST  0 -- mframe#2 --
2019/06/13 13:45:29 raspberryp LUA Script          0  length = 28 bytes
2019/06/13 13:45:29 raspberryp LUA Script          0
2019/06/13 13:45:29 raspberryp LUA Script          0  data[000]-[007] 7E 00 18 88 33 4E 44 00
2019/06/13 13:45:29 raspberryp LUA Script          0                ~          3  N  D
2019/06/13 13:45:29 raspberryp LUA Script          0  data[008]-[00F] 0D 04 00 13 A2 00 40 55
2019/06/13 13:45:29 raspberryp LUA Script          0                                   @  U
2019/06/13 13:45:29 raspberryp LUA Script          0  data[010]-[017] 80 26 47 44 65 76 69 63
2019/06/13 13:45:29 raspberryp LUA Script          0                   &  G  D  e  v  i  c
2019/06/13 13:45:29 raspberryp LUA Script          0  data[018]-[01B] 65 34 00 E6
2019/06/13 13:45:29 raspberryp LUA Script          0                e  4
```

```
2019/06/13 13:45:29 raspberryp TEST/XBEE_API_TEST   0 -- mframe#3 --
2019/06/13 13:45:29 raspberryp LUA Script           0  length = 28 bytes
2019/06/13 13:45:29 raspberryp LUA Script           0
2019/06/13 13:45:29 raspberryp LUA Script           0  data[000]-[007] 7E 00 18 88 33 4E 44 00
2019/06/13 13:45:29 raspberryp LUA Script           0                  ~          3  N  D
2019/06/13 13:45:29 raspberryp LUA Script           0  data[008]-[00F] 0A 01 00 13 A2 00 40 4A
2019/06/13 13:45:29 raspberryp LUA Script           0                                  @  J
2019/06/13 13:45:29 raspberryp LUA Script           0  data[010]-[017] C3 9C 3D 44 65 76 69 63
2019/06/13 13:45:29 raspberryp LUA Script           0                        =  D  e  v  i  c
2019/06/13 13:45:29 raspberryp LUA Script           0  data[018]-[01B] 65 31 00 4B
2019/06/13 13:45:29 raspberryp LUA Script           0                  e  1     K
2019/06/13 13:45:29 raspberryp TEST/XBEE_API_TEST   0 -- mframe#4 --
2019/06/13 13:45:29 raspberryp LUA Script           0  length = 28 bytes
2019/06/13 13:45:29 raspberryp LUA Script           0
2019/06/13 13:45:29 raspberryp LUA Script           0  data[000]-[007] 7E 00 18 88 33 4E 44 00
2019/06/13 13:45:29 raspberryp LUA Script           0                  ~          3  N  D
2019/06/13 13:45:29 raspberryp LUA Script           0  data[008]-[00F] 0B 02 00 13 A2 00 40 4A
2019/06/13 13:45:29 raspberryp LUA Script           0                                  @  J
2019/06/13 13:45:29 raspberryp LUA Script           0  data[010]-[017] C3 97 41 44 65 76 69 63
2019/06/13 13:45:29 raspberryp LUA Script           0                        A  D  e  v  i  c
2019/06/13 13:45:29 raspberryp LUA Script           0  data[018]-[01B] 65 32 00 49
2019/06/13 13:45:29 raspberryp LUA Script           0                  e  2     I
```

32.5 xbee_transmit_data(), zb_transmit_data()

- **機能概要**

  指定したリモート側 XBee, XBee-ZB デバイスにデータを送信する。

- **関数定義**

  stat, frame = xbee_transmit_data(com, dest, data *[,option]*)

  stat, frame = zb_transmit_data(com, dest, data *[,option]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
com:String	シリアルデバイスに "XBEE"または "ZB" タイプで登録済みの COMPort名またはタイトル名
frame:String	コマンド実行後に受信したTransmit Statusフレームデータ。16進数表記の文字列
dest:String	送信先のリモート XBee, XBee-ZBデバイスを指定する。dest文字列には以下のいずれかを指定する。 XBee, XBee-ZB デバイスの Node Identification 文字列

	XBee, XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列 XBee デバイスの Source Address(16bitアドレス)16進数表記の文字列 “*” ブロードキャストアドレス *注意* XBee-ZB デバイスの Network Address(16bitアドレス)は dev に指定できません。
data:String	送信するデータ。16進数表記の文字列で指定する。
option:String	Transmit Request API 中に指定する Option バイト(1byte)。16進数表記の文字列で指定する。このパラメータを省略した場合や”” 空文字列を指定した場合には ″00″ が指定される。

- **備考**

data に指定可能なデータ数は、XBee, XBee-ZB デバイスの仕様によって100 バイトまでに制限されます。

dest に、abs_agentの COM ポートに接続した XBee, XBee-ZB デバイス(送信元)を指定することはできません。

- **使用例**

```
local com = ″/dev/ttyUSB0″
function debug_frame_dump(stat,frame,mframe)
    if not stat then error() end
    log_msg(″-- frame --″,g_script)
    log_hexdump(frame)
    if mframe then
        for k,v in ipairs(mframe) do
            log_msg(″-- mframe#″ .. k .. ″ --″,g_script)
            log_hexdump(v)
        end
    end
end
debug_frame_dump(xbee_transmit_data(com,″Device3″,tbl_to_hex(str_to_tbl(″Hello World!!″))))
debug_frame_dump(xbee_transmit_data(com,″*″,tbl_to_hex(str_to_tbl(″Hello World!!″))))
```

上記スクリプト実行時のログ出力例

```
2019/06/14 09:08:11 raspberryp TEST/XBEE_API_TEST   0 -- frame --
2019/06/14 09:08:11 raspberryp LUA Script           0  length = 7 bytes
2019/06/14 09:08:11 raspberryp LUA Script           0
2019/06/14 09:08:11 raspberryp LUA Script           0  data[000]-[006] 7E 00 03 89 56 00 20
2019/06/14 09:08:11 raspberryp LUA Script           0                  ~           V
2019/06/14 09:08:11 raspberryp TEST/XBEE_API_TEST   0 -- frame --
2019/06/14 09:08:11 raspberryp LUA Script           0  length = 7 bytes
2019/06/14 09:08:11 raspberryp LUA Script           0
```

```
2019/06/14 09:08:11 raspberryp LUA Script          0  data[000]-[006] 7E 00 03 89 C4 00 B2
2019/06/14 09:08:11 raspberryp LUA Script          0                  ~
```

32.6 xbee_tdcp_command(), zb_tdcp_command()

- **機能概要**

指定したリモート XBee, XBee-ZB デバイスに接続してTDCP モニタプログラムが動作しているコントローラデバイスに任意のTDCPコマンドを送信してリプライを受信する。

- **関数定義**

stat [,result] = xbee_tdcp_command(com, dest, tdcp_cmd [,no_result])

stat [,result] = zb_tdcp_command(com, dest, tdcp_cmd [,no_result])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。 コマンド送信とレスポンス受信が正常に行われた場合には true が返りエラー発生時は false が返る。コマンド実行ステータスが成功しているかどうかは resut 中の第２エントリに格納された実行結果ステータス値を確認すること。
com:String	シリアルデバイスに "XBEE"または "ZB" タイプで登録済みの COMPort名またはタイトル名
result:Table [1..#max_frame] of String	コマンドリプライデータ文字列配列 カンマ区切りの TDCP コマンドリプライデータを、配列に変換した値で取得する。
dest:String	送信先の XBee, XBee-ZBデバイスを指定する。dest文字列には以下のいずれかを指定する。 XBee, XBee-ZB デバイスの Node Identification 文字列 XBee, XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列 XBee デバイスの Source Address(16bitアドレス)16進数表記の文字列 “*” ブロードキャストアドレス *注意* XBee-ZB デバイスの Network Address(16bitアドレス)は dest に指定できません。
tdcp_cmd:String	コントローラデバイスに送信する TDCP コマンド文字列
no_result:Boolean	コマンドリプライデータ(result) の受信を省略する。パラメータ省略時には、false が指定される。

- **備考**

リモートデバイスへのデータ送信処理とリプライデータ受信に成功した場合には stat に true が設定されます。マスターに存在しないデバイスを指定した場合や、リモートへのアクセスに失敗した場合には stat に false が返ります。

stat に true が返った場合でも、コントローラデバイスで実行したリモートコマンド自身が成功したかどうか

は別途 result 配列中のコマンド実行ステータスデータを確認する必要があります。

result 配列には、コントローラデバイスで動作している TDCP モニタプログラムから返されたリプライデータが文字配列として格納されます。リプライデータはカンマ区切りのフォーマットで、各カラムの内容を取り出して result 配列に格納しています。TDCP コマンドのリプライデータは第1カラム result[1] に TDCP プリフィックスを示す "$$$xxxxx"(xxxxx はランダムな数字) の文字列が入ります。第2カラム result[2] には TDCP コマンドの実行が成功した場合には "1" が格納されて、失敗した場合には "0" が格納されています。コマンド実行が成功した場合には、コマンドによってはいくつかのリターン値が result[3] ... result[n] に格納されます。result 配列に格納されるデータの詳しい仕様は TDCP ユーザーマニュアル中の各コマンドリファレンスのリプライデータフォーマットを参照してください。

tdcp_cmd に指定可能な文字列長は、XBee デバイスの仕様によって100 – 9 バイト(TDCP コマンドプリフィックス = 9bytes)までに制限されます。(ファームウエアバージョン "10CD")

dest に、abs_agent の COM ポートに接続したXBee, XBee-ZBデバイス(送信元)は指定できません。

no_result パラメータに true を指定することで、TDCP コントローラ側でのリプライパケット送信と、abs_agent 側の受信処理を省略して処理時間を短縮することができます。ただし、TDCP コントローラ側で実行時に発生したエラーは検出できません。

- **制限事項**

dest にブロードキャストアドレスを指定すると、リターン値の result には空テーブルが返ります。
ただしこの場合でも、コマンド実行はブロードキャストパケットを受信した複数の TDCP コントローラデバイスで正常に実行されます。(no_result パラメータの指定内容に関わらず、常に result は空になります)

- **使用例**

```
local com = "/dev/ttyUSB0"
function debug_result_dump(stat,result)
  if not stat then error() end
  if result then
    for k,v in ipairs(result) do
      log_msg("result#" .. k .. " = " .. v,g_script)
    end
  end
end

debug_result_dump(xbee_tdcp_command(com,"Device1","version"))
debug_result_dump(xbee_tdcp_command(com,"Device2","version"))
debug_result_dump(xbee_tdcp_command(com,"Device1","lcd_disp,ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
1234567890 !#$%&() Hello World!!"))
debug_result_dump(xbee_tdcp_command(com,"Device2","lcd_disp,1,ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890 !#$%&() Hello World!!"))
```

上記スクリプト実行結果ログ

```
2019/06/14 09:18:29 raspberryp SERIAL_XBEE          0 XBee_802_15_4[/dev/ttyUSB0]
7E0014810A0146002424243237373738 2C312C322E35375D
2019/06/14 09:18:29 raspberryp TEST/XBEE_API_TEST   0 result#1 = $$$27778
2019/06/14 09:18:29 raspberryp TEST/XBEE_API_TEST   0 result#2 = 1
2019/06/14 09:18:29 raspberryp TEST/XBEE_API_TEST   0 result#3 = 2.57
2019/06/14 09:18:30 raspberryp SERIAL_XBEE          0 XBee_802_15_4[/dev/ttyUSB0]
7E0014810B0242002424243437373430 2C312C322E303470
2019/06/14 09:18:30 raspberryp TEST/XBEE_API_TEST   0 result#1 = $$$47740
2019/06/14 09:18:30 raspberryp TEST/XBEE_API_TEST   0 result#2 = 1
2019/06/14 09:18:30 raspberryp TEST/XBEE_API_TEST   0 result#3 = 2.04
2019/06/14 09:18:31 raspberryp SERIAL_XBEE          0 XBee_802_15_4[/dev/ttyUSB0]
7E000F810A014500242424333634373 92C3158
2019/06/14 09:18:31 raspberryp TEST/XBEE_API_TEST   0 result#1 = $$$36479
2019/06/14 09:18:31 raspberryp TEST/XBEE_API_TEST   0 result#2 = 1
2019/06/14 09:18:32 raspberryp SERIAL_XBEE          0 XBee_802_15_4[/dev/ttyUSB0]
7E000F810B0242002424243935323233 2C3161
2019/06/14 09:18:32 raspberryp TEST/XBEE_API_TEST   0 result#1 = $$$95223
2019/06/14 09:18:32 raspberryp TEST/XBEE_API_TEST   0 result#2 = 1
```

32.7 xbee_get_local_addr64(), zb_get_local_addr64()

- **機能概要**

  abs_agent の COM ポートに接続されたXBeeデバイスのシリアル番号(64bit Address)を取得する。

- **関数定義**

  stat, addr64 = xbee_get_local_addr64(com)

  stat, addr64 = zb_get_local_addr64(com)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
com:String	シリアルデバイスに "XBEE"または "ZB" タイプで登録済みの COMPort名またはタイトル名
addr64:String	abs_agent の COM ポートに接続されたXBee, XBee-ZB デバイスのシリアル番号(64bitアドレス)16進数表記の文字列

- **備考**

  このライブラリ関数は abs_agent 起動後に最初にコールされたときに、AT コマンドでシリアル番号の問い合

わせを実行します。

この時取得したシリアル番号を abs_agent のグローバル共有変数に保存しています。それ以降このライブラリ関数がコールされる毎に、グローバル共有変数に保存されたシリアル番号を返すだけの動作を行います。

- **使用例**

```
local com = "/dev/ttyUSB0"
local stat, local_addr64 = xbee_get_local_addr64(com)
if not stat then error() end
```

33 Excel VBA・WindowsアプリケーションAPI (XASDLCMD.DLL)

Windows PC で動作する Win32アプリケーションや EXCEL VBA(32ビット版のみ)等から abs_agent の機能を使用する場合に、スクリプト実行やグローバル共有変数アクセスなどを行う機能を DLL ライブラリで提供しています。

この DLL ライブラリをリモート Windows PCで利用する場合には、予めクライアント PCに DLLライブラリ (XASDLCMD.DLL) をインストールしてください。 XASDLCMD.DLL ファイルのインストール方法については、下記の項を参照してください。ログサーバー(ABS-9000 LogServer)をインストールした Windows PC にはこの DLLライブラリが自動でインストールされますので、別途手動でインストールする必要はありません。ただし、abs_agent のインストールキットに同梱されている DLLライブラリの最新版を使用したい場合には、最新の DLLファイルを手動でインストール(上書きコピー)することをお勧めします。

XASDLCMD.DLL では、"AG_" で始まる名前のライブラリ関数が定義されています。Win32 アプリケーションから利用する場合には、直接これらのライブラリ関数をコールしてください。後述の各ライブラリ関数の説明では "**関数定義 Win32**" の項にコール形式が記述されています。

EXCEL VBA 等から利用する場合には、ラッパー関数を経由してこれらのライブラリ関数をコールする方法が便利です。ラッパー関数の定義は、abs_agent インストールキットに同梱されている EXCEL デモファイル中の標準モジュールで定義しています。この標準モジュールは、自由に複製・配布して利用できます。abs_agent のインストールキット中の "contrib/Excel/Sample_Agent.xls" に格納されています。

後述の各ライブラリ関数の説明では "**関数定義 VBA**" の項にコール形式が記述されています。

33.1 DLLライブラリのインストール

abs_agentに付属するAPIライブラリ (XASDLCMD.DLL) をインストールする場合は、下記のファイルをクライアントPCにコピーしてください。

abs_agent キット中のライブラリファイル	クライアントPC上のコピー先

abs_agent/contrib/XASDLCMD.dll  	Windows のシステムディレクトリ： WindowsXP,Windows2003 の場合は ”C:¥Windows¥System32¥” Windows7(64bit) の場合は ″C:¥Windows¥SysWOW64″ になります。

33.2 "Win32" 引数タイプの説明

引数タイプ名	意味
PChar	8 ビット文字で構成される，ヌルで終わる文字列へのポインタ
Smallint	符号付き 16 ビット整数
Integer	符号付き 32 ビット整数
Double	IEEE-754 倍精度浮動小数値
WordBool	2 バイト 論理型、非ゼロの場合に True と見なされる
PSmallint	Smallint 変数へのポインタまたは、Smallint 変数配列の先頭ポインタ。
Pinteger	Integer 変数へのポインタまたは、Integer 変数配列の先頭ポインタ。
PDouble	Double 変数へのポインタまたは、Double 変数配列の先頭ポインタ。

引数が複数ある場合は、セパレータ “;”で区切ります。引数宣言に “var” が付いているものは参照渡し（ポインタ）で、付いていないものは値渡しになります。

関数定義 ″Win32″ のパラメータのスタックへ渡される順序は、右から左の順に渡されます。パラメータ削除はルーチン側が行います。これらは、Windowsオペレーティングシステム API の一般的な呼び出し形式と同一です。

33.3 エラーコード(Return値)

エラーコード[名前]	説明
0 [SUCCESS]	処理成功
1 [OUTBUFF_OVERFLOW]	バッファーエラー
2 [INPUT_ERROR]	入力エラー、パラメータエラー等
4 [COMMAND_ERROR]	コマンド処理エラー。 エラーの詳細は ログサーバーに出力されている場合がありますので、そちらも参照してください。
8 [ABORT]	コマンド処理が中止された。 エラーの詳細は ログサーバーに出力されている場合がありますので、そちらも参照してください。

33.4 abs_agent アクセス用ライブラリ関数

abs_agent にアクセスするときに使用できるライブラリ関数です。

abs_agent ではDLLライブラリ経由でアクセスする場合にはログイン認証は必要ありません。このため、Web API や WebSocketサーバーを外部の PC やコンピュータから利用するときに必要となるセッショントークン文字列を予め取得する必要はありません。

ログイン認証の代わりに、abs_agent のマスターファイル(masters.xml)中 の "HostsEquiv" エントリに、ライブラリ関数を実行するリモートコンピュータのホスト名または MAC アドレスを追加しておく必要があります。
"agent_hosts" クライアントプログラムを使用すると、アクセスするリモート側のコンピュータホスト名や MAC アドレスを簡単に登録できます。

リモート側へのソケット接続に失敗した場合や、このライブラリ関数を実行したホストからのアクセスをリモート側の abs_agent 側で許可していない場合にはエラーが発生します。リモート側 abs_agent でアクセス許可されていない場合には、リモート側 abs_agent のログ出力に "UpdateGlobalParam:*EXCEPTION* sender-host is not authorized" の様なメッセージが記録されます。

"agent_hosts" クライアントプログラムの詳細は、"クライアントプログラム" の章を参照してください。リモートアクセスを行う場合の abs_agent の動作については、"グローバル共有データAPI" の章でも説明していますので参考にして下さい。

33.4.1 AG_get_shared_data()

- **機能概要**

  abs_agent の共有データから、key に対応する値を取得する。共有データが見つからない場合は、空文字列 "" が value に設定される。

- **関数定義 VBA**

```
function AG_get_shared_data(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_get_shared_data(Host:PChar;Port:Smallint;
                        KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	共有データのキー値を指定する
ValueStr	キー値に対応する共有データの値が返る。

値が見つからない場合は空文字列が設定される。

Return値　　成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

33.4.2　AG_set_shared_data()

- **機能概要**

abs_agent の共有データに、key と対応する値valueを設定する。既存データがある場合は更新される。

- **関数定義 VBA**

```
function AG_set_shared_data(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_set_shared_data(Host:PChar;Port:Smallint;
                        KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host　　サーバーホスト名

Port　　サーバーポート番号

KeyStr　　共有データKey 文字列。(128 文字以内の文字列)

ValueStr　　共有データのKey に対応する値。(128 文字以内の文字列)

Return値　　成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

共有データを消去したい場合は value に空文字列 "" を指定します。

abs_agent を再起動した場合は、すべての共有データは破棄されます。

33.4.3　AG_inc_shared_data()

- **機能概要**

abs_agent の共有データから、key に対応するデータに対して、その現在値を数値とみなして１をインクリメントした値を文字列で新しく設定する。

- **関数定義 VBA**

```
function AG_inc_shared_data(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_inc_shared_data(Host:PChar;Port:Smallint;
                        KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	共有データのKey値を指定する。
ValueStr	共有データの Keyに対応するカウンタ値(文字列)
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

共有データが見つからないか、既存の共有データが数値に変換できない場合は、"1" が共有データに設定され、value にも "1" が設定されます。

33.4.4 AG_shift_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリストから最も過去に登録されたデータを取り出す。リスト中にデータが見つからない場合は、空文字列 "" が value に設定される。

- **関数定義 VBA**

```
function AG_shift_shared_strlist(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_shift_shared_strlist(Host:PChar;Port:Smallint;
                        ChannelStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
ValueStr	リスト中の最も過去に登録したデータ値。取り出したデータはリストから削除される。値が見つからない場合は空文字列が設定される。
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

リストに登録された最も新しいデータを取り出す場合には、AG_pop_shared_strlist() 関数を使用してください。

33.4.5 AG_pop_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリストから最も新しく登録されたデータを取り出す。リスト中にデータが見つからない場合は、空文字列 "" が value に設定される。

- **関数定義 VBA**

function AG_pop_shared_strlist(

ByVal Host As String,

ByVal Port As Integer,

ByVal ChannelStr As String,

ByVal ValueStr As String) As Integer

- **関数定義 Win32**

function AG_pop_shared_strlist(Host:PChar;Port:Smallint;

ChannelStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
ValueStr	リスト中の最も最近に登録したデータ値。取り出したデータはリストから削除される。値が見つからない場合は空文字列が設定される。
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

リストに登録された最も古いデータを取り出す場合には、AG_shift_shared_strlist() 関数を使用してください。

33.4.6 AG_get_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリスト内の指定したオフセット位置のデータを取得。

- **関数定義 VBA**

function AG_get_shared_strlist(

ByVal Host As String,

ByVal Port As Integer,

ByVal ChannelStr As String,

ByVal Index As Long,

ByVal ValueStr As String) As Integer

- **関数定義 Win32**

function AG_shift_shared_strlist(Host:PChar;Port:Smallint;

ChannelStr:PChar;Index:Integer;ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
Index	リストのオフセット位置。0 から（リストサイズ-1）間の整数を指定
ValueStr	リスト中のオフセット位置のデータ値

Return値　　成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

関数コール後もリストの内容は変化しない。

33.4.7　AG_size_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリストサイズ取得。

- **関数定義 VBA**

```
function AG_size_shared_strlist(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByRef Size As Long) As Integer
```

- **関数定義 Win32**

```
function AG_size_shared_strlist(Host:PChar;Port:Smallint;
                        ChannelStr:PChar;Size:Pinteger):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
Size	現在のリストサイズ。ChannelStr に指定した文字列リストが存在しない場合は -1 が返る。AG_get_shared_strlist()の Index パラメータで指定可能な値は 0 から Size-1 になる
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

33.4.8　AG_add_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリストにデータを登録する。

- **関数定義 VBA**

```
function AG_add_shared_strlist(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByVal Unique As Integer,
                        ByVal Limit As Integer,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_add_shared_strlist(Host:PChar;Port:Smallint;ChannelStr:PChar;Unique:Smallint;
```

Limit:Smallint;ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
Unique	1 を指定すると、登録対象のリスト中に ValueStr で指定した文字列と同じものがある場合には登録しない。0 を指定した場合にはチェック無しに登録する。
Limit	0 よりも大きな整数を指定すると、登録対象のリスト中に Limit数以上のデータが登録済みの場合には、古いデータを削除してから ValueStr で指定したデータを登録する。0 を指定した場合にはチェック無しに登録する。
ValueStr	登録する文字列データ。(128 文字以内の文字列)
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

abs_agent を再起動した場合は、すべてのグローバル共有文字列リストは破棄されます。

33.4.9 AG_clear_shared_strlist()

- **機能概要**

abs_agent のグローバル共有文字列リスト中の、Channel に指定したリストを削除する。

- **関数定義 VBA**

```
function AG_clear_shared_strlist(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_clear_shared_strlist(Host:PChar;Port:Smallint;
                                ChannelStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有文字列リストのチャンネル名を指定する
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

abs_agent を再起動した場合は、すべてのグローバル共有文字列リストは破棄されます。

33.4.10 AG_script_exec()

- **機能概要**

abs_agent のスクリプトを実行する。

- **関数定義 VBA**

```
function AG_script_exec(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ScriptName As String,
                        ByVal KeyList As String,
                        ByVal ValueList As String) As Integer
```

- **関数定義 Win32**

```
function AG_script_exec(Host:PChar;Port:Smallint;
                        ScriptName, KeyList, ValueList:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ScriptName	スクリプト名
KeyList	スクリプトパラメータ Key リスト(CSV形式)
	パラメータが無い場合は、空文字列 "" を渡すこと。
ValueList	スクリプトパラメータ Value リスト(CSV形式)
	パラメータが無い場合は、空文字列 "" を渡すこと。
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

指定したスクリプトを実行します。呼び出したスクリプト実行が終了するまで、呼び出し側は待ち状態になります。

abs_agent で同時実行可能なスクリプトの数に制限があるので使用時は注意してください。

33.4.11 AG_script_exec2()

- **機能概要**

abs_agent でスクリプトを実行する。スクリプト中から指定したリターン値を取得できる。

- **関数定義 VBA**

```
function AG_script_exec2(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ScriptName As String,
                        ByVal KeyList As String,
                        ByVal ValueList As String,
                        ByVal ResultKeyList As String,
                        ByVal ResultValueList As String ) As Integer
```

- **関数定義 Win32**

```
function AG_script_exec(Host:PChar;Port:Smallint;
```

ScriptName, KeyList, ValueList, ResultKeyList, ResultValueList:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ScriptName	スクリプト名
KeyList	スクリプトパラメータ Key リスト(CSV形式)
	パラメータが無い場合は、空文字列 "" を渡すこと。
ValueList	スクリプトパラメータ Value リスト(CSV形式)
	パラメータが無い場合は、空文字列 "" を渡すこと。
ResultKeyList	スクリプトで指定したリターン値 Key リスト(CSV形式)が返る。
ResultValueList	スクリプトで指定したリターン値 Value リスト(CSV形式)が返る。
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

AG_script_exec() の機能と同等ですが、スクリプト中で指定した複数のリターン値を取得できます。リターン値は、実行するスクリプト中で script_result() 関数を使用して設定します。リターン値を使用しない場合は、実行スピードが AG_script_exec2()関数よりも速い、AG_script_exec()を使用してください。

呼び出したスクリプト実行が終了するまで、呼び出し側は待ち状態になります。

abs_agent で同時実行可能なスクリプトの数に制限があるので使用時は注意してください。

33.4.12 AG_csv_key_value()

- **機能概要**

AG_script_exec2() ライブラリ関数で取得したリターンパラメータのキーと値の CSV 文字列から、指定したキー文字列に対応する値(文字列) を取得する。

- **関数定義 VBA**

```
function AG_csv_key_value(
                        ByVal KeyList As String,
                        ByVal ValueList As String,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) as Integer
```

- **関数定義 Win32**

function AG_csv_key_value(KeyList, ValueList, KeyStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

KeyList	スクリプトリターンパラメータ Key リスト(CSV形式)
ValueList	スクリプトリターンパラメータ Value リスト(CSV形式)
KeyStr	キー値を指定する。(128 文字以内の文字列)
ValueStr	Key値に一致するリターンパラメータの値が返る。
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

KeyList と ValueList に指定した CSV 形式のキーと値のリストのペアから、KeyList 中の任意の項目(キー値)を KeyValue に指定して対応する値(文字列)を ValueList から探してValueStr に入れる。このときリターン値には 0 が入る。

KeyValue に指定したキー値に一致する項目が KeyList 中に見つからなかった場合には “”(空文字列) が ValueStrに入り、リターン値は 1 が入る。

33.4.13 AG_get_shmem_numarr()

- **機能概要**

abs_agent のグローバル共有メモリエリア中から数値データを取り出す。

- **関数定義 VBA**

```
function AG_get_shmem_numarr(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByVal Index As Long,
                        ByRef Value As Double) As Integer
```

- **関数定義 Win32**

```
function AG_get_shmem_numarr(Host:PChar;Port:Smallint;
                        ChannelStr:PChar;Index:Integer;Value:Pdouble):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有メモリエリアのチャンネル名を指定する
Index	チャンネル先頭からのオフセット値。実際のバイトデータ位置は Index * 8 になる 現在のメモリエリアサイズを超える Index 値を指定した場合にはエラーになる。
Value	Index で指定した位置にある IEEE-754 倍精度浮動少数値(8 bytes長)
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

グローバル共有メモリエリアのデータは予め、AG_set_shmem_numarr() DLL ライブラリ関数や shmem_store_num() Lua ライブラリ関数、agent_shmem クライアントプログラムで設定できる。

33.4.14 AG_set_shmem_numarr()

- **機能概要**

abs_agent のグローバル共有メモリエリア中に数値データを設定する。

- **関数定義 VBA**

```
function AG_set_shmem_numarr(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByVal Index As Long,
                        ByVal Value As Double) As Integer
```

- **関数定義 Win32**

```
function AG_set_shmem_numarr(Host:PChar;Port:Smallint;
                        ChannelStr:PChar;Index:Integer;Value:Double):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有メモリエリアのチャンネル名を指定する 存在しないチャンネル名を指定した場合には新規に作成される。
Index	チャンネル先頭からのオフセット値。実際のバイトデータ位置は Index * 8 になる 現在のメモリエリアサイズを超える Index 値を指定した場合には自動拡張される
Value	Index で指定した位置に設定する IEEE-754 倍精度浮動少数値(8 bytes長)
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

このライブラリ関数で設定したグローバル共有メモリエリアのデータは、AG_get_shmem_numarr() DLL ライブラリ関数や shmem_copy_num() Lua ライブラリ関数、agent_shmem クライアントプログラムで取り出せる。

33.4.15 AG_size_shmem_numarr()

- **機能概要**

abs_agent のグローバル共有メモリエリアの数値データ配列サイズを取得。

- **関数定義 VBA**

```
function AG_size_shmem_numarr(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String,
                        ByRef Size As Long) As Integer
```

- **関数定義 Win32**

```
function AG_size_shmem_numarr(Host:PChar;Port:Smallint;
                        ChannelStr:PChar;Size:Pinteger):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号

ChannelStr	グローバル共有メモリエリアのチャンネル名を指定する
Size	現在のメモリエリアサイズ。AG_get_shmem_numarr()の Index パラメータで指定可能な値は 0 から Size-1 になる
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

33.4.16 AG_remove_shmem_numarr()

- **機能概要**

abs_agent のグローバル共有メモリエリア内のチャンネルとその内容を削除。

- **関数定義 VBA**

```
function AG_remove_shmem_numarr(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal ChannelStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_remove_shmem_numarr(Host:PChar;Port:Smallint;
                        ChannelStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

Host	サーバーホスト名
Port	サーバーポート番号
ChannelStr	グローバル共有メモリエリアのチャンネル名を指定する
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

33.4.17 AG_log()

- **機能概要**

abs_agent のログサーバーにメッセージ出力。

- **関数定義 VBA**

```
function AG_size_shmem_numarr(
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal MessageStr As String,
                        ByVal ModuleStr As String) As Integer
```

- **関数定義 Win32**

```
function AG_log(Host:PChar;Port:Smallint;MessageStr,ModuleStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

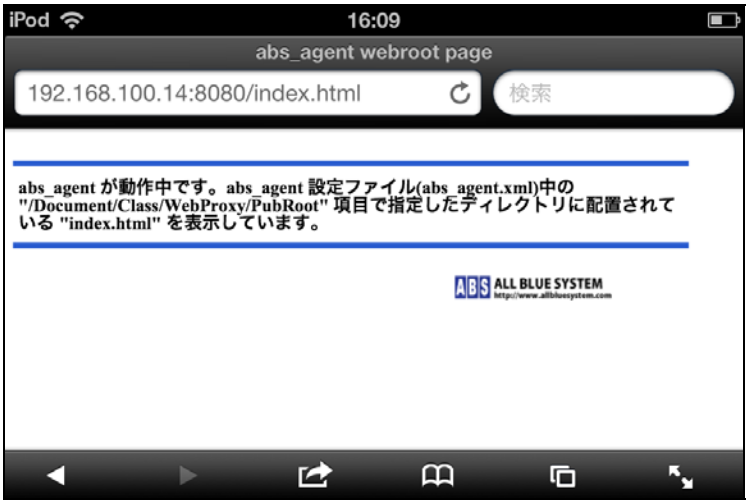
Host	サーバーホスト名
Port	サーバーポート番号
MessageStr	出力したいログメッセージを指定する

ModuleStr	出力したいログモジュール名を指定する。空文字列の場合は "BASIC" になる
Return値	成功した場合は 0 失敗した場合は 0 以外のエラーコードが返る。

- **備考**

34 HTTPサーバー & Web API

abs_agent には HTTPサーバー(Webサーバー) 機能が内蔵されています。デフォルトではインストールディレクトリ内の "webroot" 以下に格納した、HTMLやJavaScript,CSS ドキュメント等をWebブラウザからポート番号(8080)を使用して参照することができます。



(インストール直後に、タブレットデバイスから abs_agent の公開ディレクトリにアクセスした様子)

Webで公開するディレクトリやポート番号は abs_agent.xml サーバー設定ファイルで自由に変更するすることができます。HTTPサーバー機能を使用すると、簡単にローカルでWebサイトの作成や、Webアプリケーションを配信することが出来ます。abs_agent の Web API 機能を利用した Webアプリケーションを配置すると、abs_agent が動作しているコンピュータに接続されたセンサーや I/O 装置を簡単に操作できるようになります。

HTTP サーバーのドキュメント配信機能の他に、ログイン認証やスクリプト実行、グローバル共有変数操作などの Web API 機能を利用することができます。

Webブラウザや Webアプリケーションから HTTP-GETプロトコルや XMLHttpRequest()関数で abs_agent にアクセスすると、abs_agent 内のログイン認証やスクリプト実行、グローバル共有変数、グローバル共有文字列リストを操作して結果を取得することができます。

Web API 実行結果は JSON や JSONP 形式で取得することができます。これによって JavaScript 等から簡単に abs_agent の機能を利用したシステムを構築することができます。

abs_agent の Web API 機能は、クロスドメイン通信を行うための “XMLHttpRequest Level 2” と “XdomainRequest” の処理に必要な “Access-Control-Allow-Origin” ヘッダを両方共サポートしています。また、JSONP 形式でのクロスドメイン通信もサポートしています。これによって、様々な Web ブラウザやアプリケーションサーバー、他のWebサービスから abs_agent の機能を呼び出してアプリケーションを構築することができます。

34.1 Webサーバーの設定

abs_agent の Webサーバー機能はインストール直後にはデフォルトポート番号 8080 で有効になっています。公開するHTTP ドキュメントの格納ディレクトリは abs_agent インストールディレクトリ以下の "webroot" に設定されています。

これらのデフォルト設定を変更する場合には、abs_agentサーバー設定ファイル（abs_agent.xml）中の以下の項目を編集してください。設定変更後は、abs_agent プログラムを再起動して新しい設定内容を反映させます。設定項目の詳しいタグの名前や設定値については "サーバープログラム・設定ファイル"章の "abs_agent.xml サーバー設定ファイル" の項を参照して下さい。

abs_agent.xml 設定項目	説明
UseHTTPServer	abs_agent の Webサーバー、Web API機能を使用するかどうかを指定する
HTTPServerPort	HTTP プロトコルでアクセスするときに使用するポート番号
PubRoot	Webサーバーで公開するディレクトリ
DetailLog	Webサーバー、Web API アクセス時の詳細ログメッセージをログサーバーに送信

34.2 アクセスエラーカウンタ $HTTP_ERROR_COUNT

abs_agent の HTTPサーバーに外部からアクセスがあった時に、URL で指定されたリソースが存在しない場合や、Web API コマンドで無効なパスが指定された場合にはログサーバーにエラーメッセージが出力されます。このとき同時に、グローバル共有変数 $HTTP_ERROR_COUNT に累計エラー数が格納されます。

agent_data クライアントプログラムやユーザースクリプトから $HTTP_ERROR_COUNT グローバル共有変数を参照すると、 HTTPサーバーで発生した累計エラー数を何時でも確認できます。

累計エラー数をリセットしたい場合には、$HTTP_ERROR_COUNT グローバル共有変数を削除してください。

```
pi@raspi3:~/abs_agent$ ./agent_data

ServerName: raspi3      SharedData: 5
------------------------------------------------------------------------
<Key>=<Value>
------------------------------------------------------------------------
TEST/RASPI_HC595_COUNTER_RUNNING=1
RASPI/ALARM_TASK_RUNNING=1
```

```
abc=def
HC595_COUNTER=159
$HTTP_ERROR_COUNT=3


pi@raspi3:~/abs_agent$ ./agent_data -k '$HTTP_ERROR_COUNT' -d
pi@raspi3:~/abs_agent$
```

(agent_data クライアントプログラムを使用して、Webサーバーのアクセスエラー数を表示した後、カウンタをリセットした様子。スクリプト中から set_shared_data() ライブラリ関数を使用しても同様の操作ができます)

34.3 Web API使用例

abs_agent の Web API でスクリプト実行や共有変数にアクセスする場合には、Web API のログイン認証コマンドを使用してセッショントークンを取得します。このとき、ログイン認証用のアカウントを予め abs_agent に作成しておく必要があります。

クライアントプログラム agent_webuser コマンドを使用してユーザーアカウントの作成や削除等を行うことができます。たとえば、ユーザー名が "webuser1" 初期パスワード "webpass1" でログインできるユーザーアカウントを作成する場合には下記の様なコマンドを実行します。

```
pi@raspi3:~/abs_agent$ ./agent_webuser -a webuser1 -p webpass1
pi@raspi3:~/abs_agent$ ./agent_webuser

ServerName: raspi3    Users: 2    Master: /home/pi/my_config/masters.xml
------------------------------------------------------------------------
<login_name>
------------------------------------------------------------------------
user1
webuser1

pi@raspi3:~/abs_agent$
```

作成したユーザーアカウントを使用して、Web API のログイン認証(/command/json/session_login)を行う場合には下記の URL にアクセスします。このとき、取得できる文字列(JSON形式) にログイン認証されたセッショントークン文字列が格納されています。

```
pi@raspi3:~/abs_agent$ curl 'localhost:8080/command/json/session_login?user=webuser1&pass=webpass1'
{"Result":"Success","ErrorText":"","SessionToken":"ST04452EBADA3C6D"}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$
```

(上記の実行例では、"ST04452EBADA3C6D" がログイン認証で得られたセッショントークン文字列です)

以降、Web API 経由でスクリプト実行やグローバル共有変数アクセス、グローバル共有文字列リストを操作する場合には、ログイン認証時に得られたセッショントークン文字列を URL パラメータに指定します。例えば、Web API でスクリプト実行(/command/json/script)する場合には、下記の URLにアクセスします。name パラメータで指定した PARAM_ECHO スクリプトを実行しています。この PARAM_ECHO スクリプトは実行時に指定された全てのリクエストパラメータを単にリターンパラメータに戻すだけの動作を行います。

```
pi@raspi3:~/abs_agent$ curl 'localhost:8080/command/json/script?session=ST04452EBADA3C6D&name=PARAM_ECHO&Key1=Val1'
{"Result":"Success","ErrorText":"","TaskID":"LUA04452EC6C2A42E","ResultParams":{"Key1":"Val1"}}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$
```

既存のユーザーアカウントのパスワードを変更(/command/json/session_password)する場合は、ログイン認証済みのセッショントークンと新しいパスワードを URL パラメータに指定して Web API コマンドを実行します。例えばパスワードを "webpass2" に変更する場合には、下記の URL にアクセスします

```
pi@raspi3:~/abs_agent$ curl 'localhost:8080/command/json/session_password?session=ST04452EBADA3C6D&pass=webpass2'
{"Result":"Success","ErrorText":""}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$
```

ログアウト(/command/json/session_logout)する場合には、Web API 経由で下記の URLにアクセスします。

```
pi@raspi3:~/abs_agent$ curl 'localhost:8080/command/json/session_logout?session=ST04452EBADA3C6D'
{"Result":"Success","ErrorText":""}pi@raspi3:~/abs_agent$
pi@raspi3:~/abs_agent$
```

ここでの実行例で使用した Web API コマンドの詳しい仕様については以降で詳しく説明します。また、agent_webuser クライアントプログラムの詳しい仕様は、"クライアントプログラム" の章を参照してください。

34.4 セッショントークン作成方法

WebAPI の URL パラメータ"session" で指定するセッショントークン文字列は下記のいずれかの方法で作成したものを指定します。

1. クライアントプログラム agent_session で "-n <new_session_token>" オプションを使用して作成したセッショントークン
2. スクリプトライブラリ関数 create_session() で作成したセッショントークン
3. Web API コマンド "/command/json/session_login" でログイン認証時に取得したセッショントークン

上記 3 で作成する方法ではログイン認証を伴いますので、その都度ユーザー名とパスワードを入力する必要があります。1, 2 の方法ではログイン認証を伴わずに（ユーザーアカウント情報が無い）セッショントークンを強制的に作成します。簡易的な認証で運用を行なっても構わない場合や、他の方法でセキュリティが確保できる場合には 1, 2

の方法が手軽に利用できます。

3. の方法は、Web(HTTP)経由でユーザー名とパスワードを URL で指定してログインする方法です。セキュリティの確保されたネットワークを使用して、Webブラウザやアプリケーションサーバーなどのアプリケーションから abs_agent にアクセスする場合に利用できます。詳しくは “/command/json/session_login” API の項を参照して下さい。

> ⚠ **ログイン認証時にアクセスしたコンピュータ以外からは Web API は利用できません**
> Web API経由で abs_agent の機能を利用する場合には、セッショントークン(session パラメータ)を URL に指定する必要があります。セキュリティの為、Web API ログイン認証コマンドを使用して得られたセッショントークンを、ログイン認証時に使用した コンピュータ(HTTP クライント)以外から使用すると認証エラーになります。
> agent_session クライアントコマンドや、create_session() ライブラリ関数を使用して作成したセッショントークンにはこの制限は適用されずに、常にどのコンピュータからのアクセス時でも使用できます。

例: abs_agent 起動時に “SystemDefaultSession” の名前で自動的にセッショントークンを作成する

abs_agent 起動時に実行される SERVER_START イベントハンドラスクリプト中に下記の記述を追加する

```
file_id = "SERVER_START"

--[[
*******************************************************************************
このスクリプトは abs_agent 起動時にコールされます
このスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
*******************************************************************************
]]

------------------------------------------------------
-- 認証を省略してアクセスするためのセッションを作成する
------------------------------------------------------
if not create_session("SystemDefaultSession",true) then error() end
```

34.5 /command/json/script (スクリプト実行)

abs_agent で指定したスクリプトを実行します。
URL パラメータで任意のスクリプトパラメータを渡すことができます。また、スクリプト中で指定したリターンパラメータを JSON(または JSONP)形式で受け取ることも出来ます。

リクエスト URL

ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/script	
URL パラメータ	session (必須)	ログイン認証済みまたは create_session() ライブラリ関数で作成したセッショントークン文字列
	name (必須)	abs_agent に設定した スクリプト名
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。
	resultrecords (オプション)	スクリプト中で設定したリターンパラメータを受け取る “1” を設定するとパラメータを受け取ります(省略時デフォルト) “0” を設定するとパラメータを受け取りません。abs_agent のスクリプト中でscript_result()を使用していない場合や、スクリプト実行ステータスの確認のみで構わない場合には、”0” を指定することで実行スピードを向上させることが出来ます。
	noquote (オプション)	リターンパラメータの値を文字列としてダブルコートで囲まないで JSON オブジェクトとして扱う ″0″ を指定すると全てのスクリプトリターンパラメータの値はダブルコートで囲まれます(省略時デフォルト) ″1″ を指定するとダブルコートで囲まない。このときスクリプトリターンパラメータの値に、JSON フォーマットの文字列が格納されていることを想定しています。 ″1″ を指定した場合の例外として、リターンパラメータのキー名が ″Status″(ASCII 大文字・小文字を区別しない)の場合には、対応する値には(JSON 文字列ではない)文字列が格納されているものと仮定して常にダブルコートで囲みます。キー名が ″Status″ 以外のリターンパラメータの値はダブルコートで囲まれません。
	任意のスクリプトパラメータ (オプション)	スクリプト実行時に渡されるパラメータを複数任意に指定することができます。ASCII 文字以外を指定する場合には UTF-8 で URL エンコードして下さい。

例１ サーバー名 svc01 にセッショントークン “ST02983095510584”, スクリプト名 “SAMPLE” を実行する場合

http://svc01/command/json/script?session=ST02983095510584&name=SAMPLE

例２ サーバー名 localhost で HTTPServer ポート番号 8080, セッショントークン “ST02983095510584”, スクリプト名 “PARAM_ECHO” を実行。スクリプトパラメータとして key1=val1, キー２=値２ の2 つのパラメータを指定する場合。コールバック関数に my_handler を指定する。

http://localhost:8080/command/json/script?session=ST02983095510584&name=PARAM_ECHO&callback=my_handler&key1=val1&%E3%82%AD%E3%83%BC%EF%BC%92=%E5%80%A4%EF%BC%92

> ⚠ **URL 長さ制限**
> URL で指定する文字列は、エンコード後のサイズが 2000 バイトを超えないようにしてください。

レスポンスJSONフォーマット

(JSON形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "TaskID":"<TaskID>", "ResultParams":{"<Key>":"<Value>", ...}}

(JSON形式) noquote=1 を指定した場合 (リターンパラメータ中の <Value> の両端のダブルコートが除かれます)

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "TaskID":"<TaskID>", "ResultParams":{"<Key>":<Value>, ...}}

(JSONP形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "TaskID":"<TaskID>", "ResultParams":{"<Key>":"<Value>", ...}});

(JSONP形式) noquote=1 を指定した場合 (リターンパラメータ中の <Value> の両端のダブルコートが除かれます)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "TaskID":"<TaskID>", "ResultParams":{"<Key>":<Value>, ...}});

データ項目	内容
<Result>	“Success” 処理成功 “Fail” エラー発生
<ErrorText>	Result タグが “Fail” の場合に、詳細エラーメッセージが入る
<TaskID>	スクリプトが実行されたときにアサインされた ID 実行中のスクリプトで g_taskid 変数で取得できる文字列と同一の内容が入る

ResultParams	実行したスクリプト中から script_result() 関数で設定したリターン値が入る リターン値を指定しなかった場合には ResultParams配列の内容は空になる ResultParams 配列の要素は、<Key> と <Value> のペアが入る
<Key>	script_result() 関数で設定したリターン値のキー名
<Value>	script_result() 関数で設定したリターン値のキーに対応する値 URLオプションに noquote=0 を指定した場合や noquote を指定しない場合には、スクリプトリターンパラメータの値部分は必ず文字列になります。 noquote=1 を指定した場合には、リターンパラメータ中に格納された JSON 文字列がそのまま設定されますので、オブジェクトや配列データなどをそのままレスポンス JSON 文字列中に含めることができます。

例１ スクリプト実行成功で、コールバック関数指定とリターンパラメータが無い場合

```
{"Result":"Success","ErrorText":"","TaskID":"LUA029914EB5B5039","ResultParams":{}}
```

例２ スクリプト実行成功で、リターンパラメータが複数あり、コールバック関数をmy_handler に指定した場合

```
my_handler({"Result":"Success","ErrorText":"","TaskID":"LUA0299150A101521","ResultParams":
{"key1":"val1","¥u30AD¥u30FC¥uFF12":"¥u5024¥uFF12"}});
```

例３ スクリプト実行に失敗した場合(コールバック指定あり)

```
my_handler({"Result":"Fail","ErrorText":"CertifyUpdateSession failed","TaskID":""});
```

例４ スクリプト実行に失敗した場合(コールバック指定なし)

```
{"Result":"Fail","ErrorText":"CertifyUpdateSession failed","TaskID":""}
```

例5 スクリプト中から JSON オブジェクトをリターンパラメータに設定した場合

実行するスクリプト例

```
local json_data1 = '{ "abc":"試験","MyList":["壱","弐","参"] }'
local json_data2 = '[{"tag1":"値1","tag2":"値2","tag3":
                                    "値3"},{"tag1":"Value1","tag2":"Value2","tag3":"Value3"}]'
if not script_result(g_taskid,"結果",json_data1) then error() end
if not script_result(g_taskid,"MyResult",json_data2) then error() end
```

このスクリプトを URL パラメータに noquote=1 を指定して実行した場合のレスポンス(コールバック指定なし)

```
{"Result":"Success","ErrorText":"","TaskID":"LUA038CC9B9981953","ResultParams":
  {
    "¥u7D50¥u679C":{ "abc":"¥u8A66¥u9A13","MyList":["¥u58F1","¥u5F10","¥u53C2"] },
    "MyResult":[{"tag1":"¥u50241","tag2":"¥u50242","tag3":"¥u50243"},
                {"tag1":"Value1","tag2":"Value2","tag3":"Value3"}]
  }
}
```

** 実際の レスポンスJSON 文字列中には改行やインデント用の空白文字は入りません **

34.6 /command/json/shared_data (グローバル共有データ操作)

abs_agent のグローバル共有データの key に対応する値を取得または設定する。

グローバル共有データが見つからない場合は、空文字列 "" が リターンパラメータ中の value に設定される。

実行する他のユーザースクリプトやイベントハンドラ間で、これらのデータは共有されています。このコマンドを使用して設定したグローバル共有データは、サーバーが再起動すると消去されます。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/shared_data	
URL パラメータ	session (必須)	ログイン認証済みまたは create_session() ライブラリ関数で作成したセッショントークン文字列
	key (必須)	グローバル共有データのキー名

	value (オプション)	key パラメータで指定したキーに対応するグローバル共有データ値。 このパラメータを省略した場合には、key で指定したグローバル共有データが取得できます。value パラメータを指定するとグローバル共有データの値を更新します。 value に “” 空文字列を指定すると、グローバル共有データを削除します。 value に “_increment_” を指定すると、key に対応するデータに対して、その現在値を数値とみなして１をインクリメントした値を文字列で新しく設定します。既存のグローバル共有データが見つからないか、既存のグローバル共有データが数値に変換できない場合は、 ″1″ が設定されます value に “_decrement_” を指定すると、key に対応するデータに対して、その現在値を数値とみなして１をデクリメントした値を文字列で新しく設定します。デクリメント後の値が ″0″ またはそれ以下の値になる場合や、既存のグローバル共有データが数値に変換できない場合はグローバル共有データが削除されます。
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。
	afterdelete (オプション)	“1” を指定すると、key パラメータで指定したグローバル共有データを取得後、データを削除します。このパラメータは value パラメータを指定していないときにのみ有効です。
	noquote (オプション)	取得したグローバル共有データの値を文字列としてダブルコートで囲まないで JSON オブジェクトとして扱う。 ″0″ を指定すると共有データの値はダブルコートで囲まれます(省略時デフォルト) ″1″ を指定するとダブルコートで囲まない。このとき取得したグローバル共有データには、JSON フォーマットの文字列が格納されていることを想定しています。

レスポンスJSONフォーマット

(JSON形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":"<SharedValue>"}

(JSON形式) noquote=1 を指定した場合

リターンパラメータ中の <SharedValue> の両端のダブルコートが除かれます

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":<SharedValue>}

(JSONP形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":"<SharedValue>"});

(JSONP形式) noquote=1 を指定した場合

リターンパラメータ中の <SharedValue> の両端のダブルコートが除かれます

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":<SharedValue>});

データ項目	内容
<Result>	“Success” 処理成功 “Fail”　エラー発生
<ErrorText>	Result タグが “Fail” の場合に、詳細エラーメッセージが入る
<SharedValue>	グローバル共有データの key に対応する文字列 value パラメータを指定してグローバル共有データを更新した場合には、更新後のデータが設定される グローバル共有データが見つからない場合でかつ、URLオプションに noquote=0 を指定した場合や noquote を指定しない場合には空文字列 "" が設定される。 グローバル共有データが見つからない場合でかつ、URLオプションに noquote=1 を指定した場合には null 文字列が設定される。 noquote=1 を指定した場合には、グローバル共有データに格納された JSON 文字列がそのまま設定されますので、オブジェクトや配列データなどをそのままレスポンス JSON 文字列中に含めることができます。

34.7 /command/json/shared_strlist (グローバル共有文字列リスト操作)

abs_agent のグローバル共有文字列リストを操作する。リストから文字列を取り出したり、新規文字列エントリをリストに登録することができます。また、文字列リスト全体を削除することもできます。

実行する他のユーザースクリプトやイベントハンドラ間で、全てのグローバル共有文字列リストは共有されています。グローバル共有文字列リストは、サーバーが再起動すると消去されます。

グローバル共有文字列リストに登録されている全てのチャンネル名(文字列リスト名)を JSON 形式で取得するときには、スクリプト実行 Web API (/command/json/script) を使用して "ETC/SHARED_STRLIST_CHANNELS" スクリプトを実行することで得られます。詳しい使用方法はインストールキット中に含まれる"ETC/SHARED_STRLIST_CHANNELS" スクリプトファイルのコメントをご覧ください。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/shared_strlist	
URL パラメータ	session (必須)	abs_agent でログイン認証済みまたは create_session() ライブラリ関数で作成したセッショントークン文字列
	channel (必須)	グローバル共有文字列リストのリスト名
	value (オプション)	文字列リストに新規に登録する文字列。 このパラメータを省略した場合には、文字列リストからデータを取得します。デフォルトでは文字列リストに登録した順に(古いデータから) 取り出します。(shift動作) "pop=1" URLパラメータを指定することで、新しいデータから取り出すこともできます。(pop動作)
	unique (オプション)	"1" を指定すると、value パラメータで指定したデータをリストに登録するときに、リスト中に既に同一データが存在する場合には登録しない。
	limit (オプション)	"0" よりも大きな整数を指定すると、value パラメータで指定したデータをリストに登録するときに、リストで保持できるデータ最大数を制限できる。最大数を超えている場合には古いデータから順に自動削除される。"0" を指定すると制限無しにデータを登録する。パラメータ省略時は "0" を指定したのと同様の動作。
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

	pop (オプション)	“1” を指定すると、文字列リストからデータを取得するときに一番最近に登録したデータを取り出します。このパラメータを省略した場合には最も古いデータから取り出します。このパラメータは value パラメータを指定していないときにのみ有効です。
	clear (オプション)	“1” を指定すると、文字列リストを削除します。このパラメータを指定した場合には、他の value, pop パラメータは無視されます。
	noquote (オプション)	取得したグローバル共有文字列リストの文字列データをダブルコートで囲まないで JSON オブジェクトとして扱う。 "0" を指定すると文字列データの値はダブルコートで囲まれます(省略時デフォルト) "1" を指定するとダブルコートで囲まない。このとき取得した文字列データには、JSON フォーマットの文字列が格納されていることを想定しています。

レスポンスJSONフォーマット

(JSON形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":"<SharedStrlistData>"}

(JSON形式) noquote=1 を指定した場合

リターンパラメータ中の <SharedStrlistData> の両端のダブルコートが除かれます

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":<SharedStrlistData>}

(JSONP形式) noquote URLパラメータを省略した場合、または noquote=0 を指定した場合

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":"<SharedStrlistData>"});

(JSONP形式) noquote=1 を指定した場合

リターンパラメータ中の <SharedValue> の両端のダブルコートが除かれます

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "Value":<SharedStrlistData>});

データ項目	内容
<Result>	“Success” 処理成功 “Fail” エラー発生
<ErrorText>	Result タグが “Fail” の場合に、詳細エラーメッセージが入る

<SharedStrlistData>	グローバル共有文字列リストから取得した文字列データ。 value パラメータを指定しない場合には、リストから取得した文字列データが設定される。value パラメータを指定して新規にグローバル共有文字列リストにデータを登録した場合には、登録したデータ自身が設定される グローバル共有文字列リストが空の場合でかつ、URLオプションに noquote=0 を指定した場合や noquote を指定しない場合には空文字列 "" が設定される。 グローバル共有文字列リストが空の場合でかつ、URLオプションに noquote=1 を指定した場合には null 文字列が設定される。 noquote=1 を指定した場合には、グローバル共有文字列リストから取得した JSON 文字列がそのまま設定されますので、オブジェクトや配列データなどをそのままレスポンス JSON 文字列中に含めることができます。

34.8 /command/json/session_login (ログイン認証とセッション作成)

ユーザー名とパスワードを指定して、abs_agent でログイン認証します。ログイン認証に成功すると、セッショントークン文字列を取得します。

> ⚠ **HTTP パケット内容の盗聴に注意すること**
> この WebAPI コマンドで URL パラメータに指定されるユーザー名とパスワードの情報は、ネットワーク中に平文で送信されます。このため、他からパケットの内容を盗聴される危険性があります。セキュリティが保たれていないネットワークで使用する場合には、他の方法でログイン操作を行うことを検討してください。
> 他のログインセッションの作成方法については、"セッショントークン作成方法" の項を参照して下さい。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/session_login	
URL パラメータ	user (必須)	abs_agent に登録されたユーザーのログイン名 agent_webuser クライアントコマンドでユーザーの管理ができます。
	pass (必須)	ログインパスワード文字列

	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

例１ サーバー名 svc01 にログイン名 “yamada”, パスワード “tarou” でログインする場合

http://svc01/command/json/session_login?user=yamada&pass=tarou

例２ サーバー名 localhost で HTTPServer ポート番号 8080, ログイン名 “yamada”, パスワード “tarou” でログインする場合。コールバック関数に my_handler を指定する。

http://localhost:8080/command/json/session_login?user=yamada&pass=tarou&callback=my_handler

> ⚠ **URL 長さ制限**
> URL で指定する文字列は、エンコード後のサイズが 2000 バイトを超えないようにしてください。

レスポンスJSONフォーマット

(JSON形式)

"WebSocketServerPort"タグは abs_agent でWebSocketServer 機能が有効になっているときのみ含まれる。

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>","WebSocketServerPort":<WebSocketServerPort>}

(JSONP形式)

"WebSocketServerPort"タグは abs_agent でWebSocketServer 機能が有効になっているときのみ含まれる。

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>","WebSocketServerPort":<WebSocketServerPort>});

タグ名	内容
Result	“Success” 処理成功 “Fail” エラー発生
ErrorText	Result タグが “Fail” の場合に、詳細エラーメッセージが入る
SessionToken	ログインに成功した場合に、abs_agent で作成されたセッショントークン文字列
WebSocketServerPort	WebSocketServer が動作しているポート番号

例１ ログイン成功で、コールバック関数指定が無い場合

```
{"Result":"Success","ErrorText":"","SessionToken":" ST030F6CCC865820"}
```

例２ ログイン成功で、コールバック関数をmy_handler に指定した場合

```
my_handler({"Result":"Success","ErrorText":"","SessionToken":" ST030F6CCC865820"});
```

例３ ログインに失敗した場合(コールバック指定あり)

```
my_handler({"Result":"Fail","ErrorText":"incorrect password, user = user"});
```

例４ ログインに失敗した場合(コールバック指定なし)

```
{"Result":"Fail","ErrorText":"incorrect password, user = user"}
```

34.9 /command/json/session_logout (ログアウト・セッション削除)

abs_agentのセッションを削除してログアウトする。

Luaライブラリ関数 create_session() で作成したセッションは、この WebAPI では削除することはできません。
この場合には、delete_session() 関数を使用してください。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/session_logout	
URL パラメータ	session (必須)	ログイン認証済みのセッショントークン文字列
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

レスポンスJSONフォーマット

(JSON形式)

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>"}

(JSONP形式)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>"});

タグ名	内容
Result	“Success” 処理成功

	“Fail”　エラー発生
ErrorText	Result タグが “Fail” の場合に、詳細エラーメッセージが入る

34.10 /command/json/session_update (セッション情報手動更新)

abs_agent のセッション情報を更新する。

abs_agent ではログイン時に作成したセッション情報を最後に利用（WebAPI をコールしてタイムスタンプを更新）してから10 分経過すると、セッションが利用されていないと見なされて自動的にログアウトします。このときセッション情報も同時に削除されます。/command/json/session_update API は、この自動ログアウトを防ぐ為に定期的にセッションのタイムスタンプのみを更新する場合に使用できます

“renew” パラメータを指定することで、セッショントークン文字列を変更することができます。これは、セキュリティを向上させるために、セッショントークン文字列を定期的に更新したい場合に使用します。セッションパラメータや他の属性値は全て新規に作成したセッションに引き継がれます。新規に作成されるセッショントークン文字列を明示的に指定することはできません。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/session_update	
URL パラメータ	session （必須）	ログイン認証済みまたは create_session() ライブラリ関数で作成したセッショントークン文字列
	renew （オプション）	セッショントークン文字列を更新して、新しい文字列をアサインします。 “0” を設定するとセッショントークン文字列の更新を行わずに、既存のセッションのタイムスタンプのみ更新します(省略時デフォルト) “1” を設定するとセッショントークン文字列の更新を行います。 新しく作成したセッショントークン文字列は、リプライデータ中の ”SessionToken”タグに格納されます。
	callback （オプション）	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

レスポンスJSONフォーマット

（JSON形式）

レスポンスJSONフォーマット

{"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>"}

(JSONP形式)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>"});

タグ名	内容
Result	“Success” 処理成功 “Fail” エラー発生
ErrorText	Result タグが “Fail” の場合に、詳細エラーメッセージが入る
SessionToken	タイムスタンプを更新したセッショントークン文字列。 renew パラメータに “1” を指定した場合には、新しく作成したセッショントークン文字列が格納される。

34.11 /command/json/session_password (ログインパスワード変更)

指定したログインセッションのログインユーザーパスワードを変更する。

Web API 経由で、現在ログイン中のユーザーパスワードを変更します。URL パラメータで指定するセッショントークンは、Web API /command/json/session_login で取得したものを指定します。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/session_password	
URL パラメータ	session (必須)	ログイン認証済みのセッショントークン文字列
	pass (必須)	変更するログインユーザーパスワード文字列。
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

レスポンスJSONフォーマット

(JSON形式)

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>"}

(JSONP形式)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>"});

タグ名	内容
Result	“Success” 処理成功 “Fail” エラー発生
ErrorText	Result タグが “Fail” の場合に、詳細エラーメッセージが入る

34.12 /command/json/getmyip (クライアント側のIPアドレス取得)

この API をアクセスしたクライアント自身の IP アドレスを取得する。

この API は、Web サービスクライアントが自身のローカル/グローバル IP アドレスを取得したい場合に、abs_agent の Web API 経由で、自身の HTTP リクエストパケット送信時(この WebAPI をコールした時のパケット)に確立した Peer ソケット情報を取得します。

“session”URL パラメータで指定する abs_agent 内部のセッションやログイン時のユーザー情報と、この WebAPI で取得する IP アドレスは、abs_agent 内部では何ら関連図けられていません。”session”URL パラメータはこのAPI の実行権を判断する目的のためだけに使用されます。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/getmyip	
URL パラメータ	session (必須)	ログイン認証済みまたは create_session() ライブラリ関数で作成したセッショントークン文字列
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。このパラメータを省略した場合には JSON 形式でレスポンス文字列を受信する。コールバック関数名には 英数字とアンダースコアのみが使用できます。

レスポンスJSONフォーマット

(JSON形式)

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "IPAddress":"<IPAddress>"}

(JSONP形式)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "IPAddress":"<IPAddress>"});

タグ名	内容
Result	“Success” 処理成功 “Fail” エラー発生
ErrorText	Result タグが “Fail” の場合に、詳細エラーメッセージが入る
IPAddress	この API をコールしたクライアントの IP アドレス。 abs_agent が WebAPI(HTTP)パケットを受信した時の Peer アドレスを設定します。そのため、クライントが LAN 内でローカルIP アドレスを使用していて、グローバルIP アドレスが付与されたルータ経由で abs_agent にアクセスした場合には、<IPAddress> にはルータのグローバル IP アドレスが格納されます。

34.13 /command/log (ログメッセージ出力)

abs_agent のログサーバーにログメッセージを出力します。URL パラメータで任意のログメッセージとログモジュール名を渡すことができます。このコマンドのリプライメッセージデータは常に空になります。

リクエスト URL		
ホスト名	abs_agent が動作しているコンピュータのホスト名または IP アドレス	
ポート番号	abs_agent サーバー設定ファイルで指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/log	
URL パラメータ	message (必須)	ログメッセージ本文
	module (オプション)	ログモジュール名 省略時は “WEBPROXY” がモジュール名になります。
	callback (オプション)	レスポンス文字列を JSONP 形式で受信する場合に、コールバック関数名を指定する。ただし、コールバック関数で受信するデータは常に空になります。 “callback({});”

35 WebSocketサーバー

abs_agent には HTTPサーバー(Webサーバー) 機能とは独立して WebSocket サーバー機能が内蔵されています。デフ

ォルト設定では、Webブラウザ等からポート番号(9090)を使用してabs_agent の WebSocketサーバーに接続できます。

WebSocket サーバーのポート番号は abs_agent.xml サーバー設定ファイルで自由に変更することができます。WebSocketサーバー機能を使用すると、ブラウザと abs_agent 間で双方向のリアルタイム通信を行うことができるようになります。センサーデータの最新の値をブラウザに配信して、グラフやメータ等の GUI を自動的に更新することができます。

abs_agent の WebSocket 接続時に任意のチャンネル名を指定することができます。チャンネル名を利用して同一アプリケーションが複数同時に動作しているときに、それらのクライアント全体に対してブロードキャストメッセージを送信することもできます。

abs_agent の WebSocket サーバーはアクセス認証にセッション情報を利用します。abs_agent の ログイン認証 WebAPI やライブラリ関数で作成したセッショントークンを URL に指定することでアクセス認証を行います。

abs_agent の WebSocketサーバーは RFC6455 仕様に基づいたインプリメントを行っています。PC やスマートフォン、タブレットコンピュータの Web ブラウザでデフォルトでサポートされている WebSocket インターフェイスを使用することで、外部のライブラリを使用しなくても簡単に WebSocket を利用した双方向通信を実現できます。

35.1 WebSocketサーバーの設定

abs_agent の WebSocketサーバー機能はインストール直後にはデフォルトポート番号 9090 で有効になっています。

これらのデフォルト設定を変更する場合には、abs_agentサーバー設定ファイル（abs_agent.xml）中の以下の項目を編集してください。設定変更後は、abs_agent プログラムを再起動して新しい設定内容を反映させます。設定項目の詳しいタグの名前や設定値については "サーバープログラム・設定ファイル"章の "abs_agent.xml サーバー設定ファイル" の項を参照して下さい。

abs_agent.xml 設定項目 (WebProxy サービスクラス)	説明
UseWebSocketServer	abs_agent の WebSocketサーバー機能を使用するかどうかを指定する
WebSocketServerPort	WebSocketサーバーで使用するポート番号
DetailLog	WebSocketサーバー、Webサーバー、Web APIアクセス時に共通で使用するフラグで、詳細ログメッセージをログサーバーに出力するかどうかを指定する

35.2 WebSocketサーバー仕様と接続パラメータ

abs_agent の WebSocket サーバーの仕様は下記になります。Web ブラウザやアプリケーションサーバーから WebSocket サーバーに接続する場合には仕様に従った接続パラメータを指定してください。

HTTP プロトコル項目	指定する値
HTTPプロトコルヘッダ	"HTTP/1.1" を指定します

HTTPコマンド	"GET" を指定します
セキュアプロトコル	TCP 上のレギュラーな HTTP コネクションのみをサポート。(ws://xxx/xxx) TLS ベースの WebSocket(wss://xxx/xxx) はサポートしない。必要な場合は別途 VPN 等のセキュアトンネル機能を併用してください。 abs_agent のアクセス制限は "Origin"モデルではなく、リソース名(URL) 中に記述したセッショントークンによって行います。詳しくは下記の項目を参照してください。
URL パス名(リソース名)	"ws://<abs_agentホスト名・IPアドレス>:9090/<channel>/<SessionToken>" の形式で URLパス名を指定します。 <channel> は URLパス名に記述可能な英数文字であれば任意の文字列を指定することができます。ここで指定した <channel> 文字列を使用して、abs_agent サーバー側から特定の WebSocketクライアント接続に対してフレームデータを送信することができます。 <SessionToken> は abs_agent 側に作成されたセッショントークン文字列を指定します。セッショントークンを使用して WebSocket サーバーに接続許可を与えるかどうかを判断しています。Web API や Lua ライブラリ関数等を使用してセッショントークンを作成できます。詳しくは、"HTTPサーバー＆WebAPI"の章中の "セッショントークン作成方法" の項を参照してください。 abs_agent 上の WebSocketポート番号はデフォルトで 9090 を使用します。このポート番号は任意の番号に変更できます。詳しくは "サーバープログラム・設定ファイル"の章を参照してください。
URLパラメータ	任意のURLパラメータを指定しても構わないが、abs_agent の WebScoektサーバーでは無視されます
HTTPコンテンツ	接続開始時の HTTP パケットに任意のコンテンツを格納しても構わないが、abs_agent の WebScoektサーバーでは無視されます
"Sec-WebSocket-Key" ヘッダ	必須パラメータ
"Upgrade" ヘッダ	必ず "websocket" 項目を含めてください
"Connection" ヘッダ	必ず "Upgrade" 項目を含めてください
"Sec-WebSocket-Version" ヘッダ	必ず "13" 項目を含めてください
"Origin" ヘッダ	任意のホスト名や null、またはヘッダ自身を省略可。 abs_agent の WebSocket サーバーは、クロスドメインやWebブラウザ以外からの接続等に関して一切の制限をかけないため常に接続を許可します。
サブ・プロトコル指定	abs_agent の WebScoektサーバーはサブ・プロトコルをサポートしません。指定された場合でも受け付済みのサブ・プロトコルタグは返しません。

35.3 WebSocketサーバー機能

abs_agent の WebSocket サーバーは、RFC6455 で定義されたプロトコルでクライアントからの接続を受け付けます。WebSocket 接続が完了した後、クライアントからテキストやバイナリフレームを受信すると abs_agent 側では WEBSOCKET_DATA イベントハンドラが起動します。

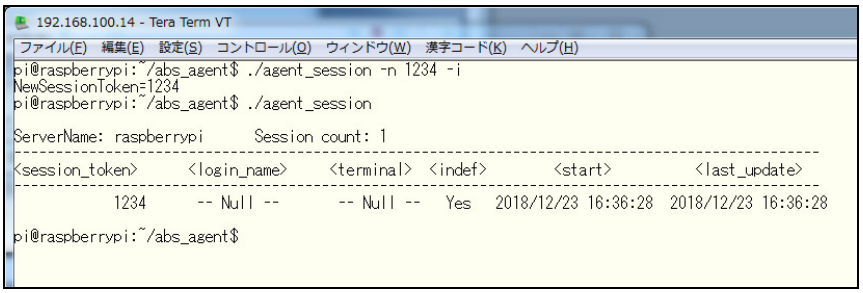
クライアントから送信されたデータを処理する場合には上記のイベントハンドラをカストマイズします。詳しいイベントハンドラとスクリプトパラーメータ詳細については、"イベント"の章中の WEBSOCKET_DATA の項を参照してください。

abs_agent(WebSocketサーバー) 側から WebSocketクライアント(Webブラウザ等)にテキストやバイナリフレームを送信するときは、websocket_emit_text(), websocket_emit_binary() ライブラリ関数を使用します。これらのライブラリ関数を使用すると、abs_agent のイベントハンドラやユーザースクリプト中から現在接続中の WebSocket クライアントに対してテキストまたはバイナリフレームを送信することができます。これらの詳しいライブラリ関数の仕様については "WebSocketサーバーAPI" の章を参照してください。

35.4 WebSocket使用例

abs_agent 上の WebSocket サーバーは WebSocketServerPort に設定したポート番号で常に接続を受け付けています。接続のために abs_agent 側にセッション情報が必要になりますが、本来は Webアプリケーションからログイン認証 WebAPI をコールして取得したり Lua ライブラリ関数で作成します。ここでは簡単に動作確認するために agent_session クライアントプログラムを使用して作成します。

コマンドプロンプトから agent_session プログラムを起動して、"1234" と名前をつけたセッションを作成します。-n <session_name> オプションに作成したいセッション名を指定します。-i オプションはセッションの自動タイムアウトを無効にするために指定します。



上記のセッション名を含んだ URL(ws://<abs_agentホスト名orIP>:9090/<任意のチャンネル名>/1234) を指定することで、PC やスマートフォン等の Webブラウザから abs_agent の WebSocket サーバーに接続できます。

ここではWebSocket 接続試験用のテストスクリプトを下記の様な HTML ファイル中に記述します。この html ファイルはインストールキット中の webroot/test/echo_test.html に格納されていますので作成する必要はありません。

HTML ファイルの内容は下記の様になっています。

echo_test.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>WebSocket クライアント</title>
        <h2>WebSocket クライアント(/my_app/1234)</h2>
        <script>
            function run() {
                output = document.getElementById("output");
                ws = new WebSocket("ws://" + location.hostname + ":9090/my_app/1234");

                ws.onopen = function (e) {
                    log("Connected");
                    // WebSocketサーバー接続時にテキストフレームを送信する
                    ws.send("こんにちは WebSocket!");
                    log("Text sent");
                }

                ws.onclose = function (e) {
                    log("Disconnected: " + e.reason);
                }

                ws.onerror = function (e) {
                    log("Error ");
                }

                // バイナリフレーム受信時のログ出力用(簡易版)
                myReader = new FileReader();
                myReader.onload = function(event){
                    log("Binary received: " + JSON.stringify(myReader.result));
                };

                // WebSocketデータフレームを受信した
                ws.onmessage = function (e) {
                    if (typeof e.data == "string"){
                        log("Text received: " + JSON.stringify(e.data));
                    } else {
                        myReader.readAsText(e.data);
                    };
                }
            }

            // ログメッセージをWebページ中の output に出力
            function log(s) {
                var p = document.createElement("p");
                p.style.wordWrap = "break-word";
                p.textContent = s;
                output.appendChild(p);
                console.log(s);
            }
        </script>
    </head>
    <body>
        <div id="output"></div>
        <script>
            run();
        </script>
    </body>
</html>

```

(echo_test.html ファイルの内容)

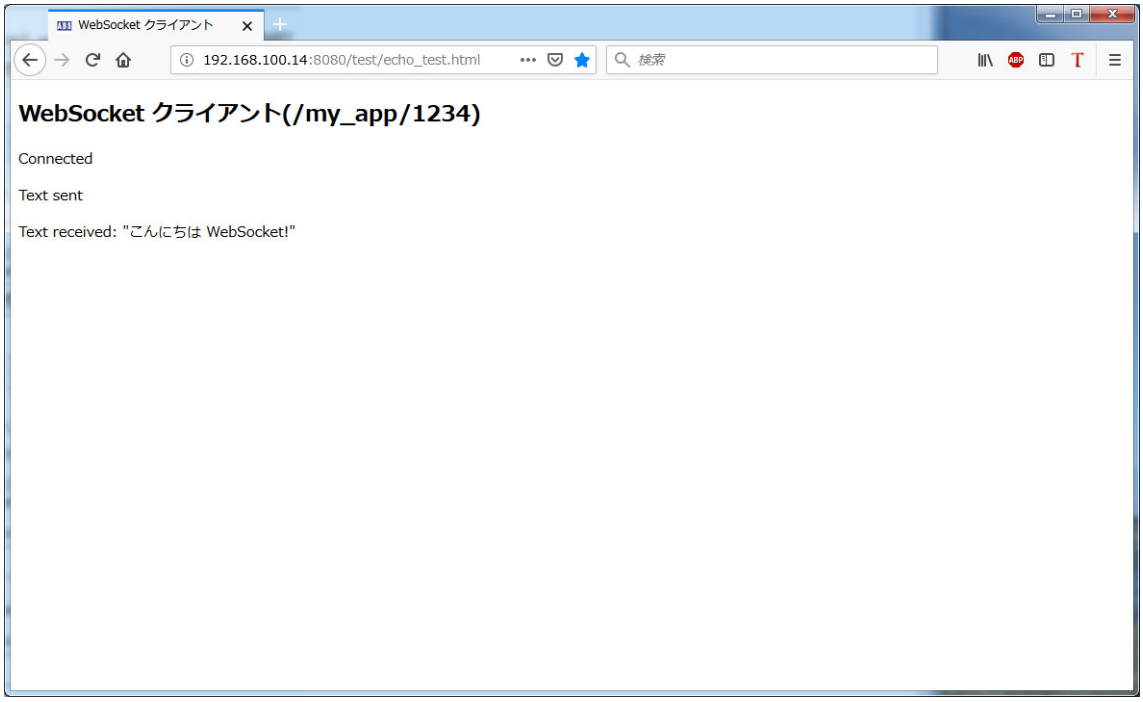
また、HTML 中の JavaScript からは WebSocket テキストフレームを送信していますが、このとき abs_agent の WebSocketサーバー側でフレーム受信時に実行されるイベントハンドラ(WEBSOCKET_DATA)が下記になります。

インストール直後のデフォルトイベントハンドラ(scripts/WEBSOCKET_DATA.lua) の最後の３行のコメント部分は、クライアントが送信してきた文字列をそのままエコーバックするサンプルになっています。このエコー機能を試験に利用するためにコメントを外しておきます。

```
-------------------------------------------------------------------------------
-- 受信したPayloadType が "text" の場合には
-- バイナリデータ列を UTF-8 文字列としてデコードしたものを PayloadString 変数に格納する
-------------------------------------------------------------------------------
local PayloadString = ""
if g_params["PayloadType"] == 'text' then
    local pub_len = tonumber(g_params["PayloadSize"])
    PayloadString = readUTF_hex(bit_tohex(pub_len,4) .. g_params["PayloadData"])
end

if PayloadString ~= "" then
    log_msg("/" .. g_params["Channel"] .. "/" .. g_params["SessionToken"] .. " [" .. g_params["WebSocketID"] .. "] " .. PayloadString,g_script)
else
    log_msg("/" .. g_params["Channel"] .. "/" .. g_params["SessionToken"] .. " [" .. g_params["WebSocketID"] .. "] " .. g_params["PayloadData"],g_script)
end

-------------------------------------------------------------------
-- 送信してきたメッセージ内容をそのままクライアント側に送り返す(Echo)例
-------------------------------------------------------------------
if PayloadString ~= "" then
    if not websocket_emit_text(g_params["WebSocketID"],PayloadString) then error() end
end

```

(WEBSOCKET_DATA イベントハンドラに記述されているエコーバック例を有効にした状態)

PC 等の Webブラウザから http://<abs_agentホスト名またはIP>:8080/test/echo_test.html にアクセスすると上記の HTML/JavaScript が実行されます。試験では、abs_agent 内蔵の Webサーバーで上記の HTML(WebSocket試験用 JavaScript)を配信しています。この時 WebSocketサーバー(Port:9090) と HTTP サーバー(Port:8080)は、各々の受付ポート番号が異なっていますので混乱しないようにしてください。



(試験用の HTML ファイルを Webブラウザからオープンした様子)

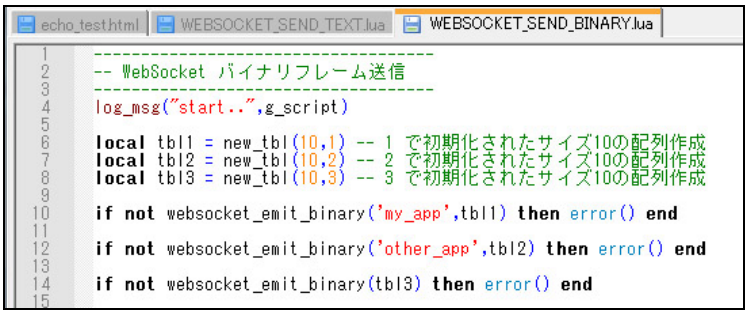
Webブラウザ側では HTML 内の JavaScript が実行されて、abs_agent の WebSocket サーバーに接続します。接続完了時に WebSocket サーバー宛にメッセージを送信しています。送信したメッセージ(テキストフレーム)は、abs_agent 側のイベントハンドラ中に記載されたエコーバックするスクリプトによってブラウザ側に送り返されている様子が判ります。

この状態で abs_agent のユーザースクリプトを使用して、WebSocket サーバー(abs_agent)からWebブラウザ(WebSocketクライアント)宛てに文字列を送信してみます。送信するためのスクリプト(scripts/TEST/WEBSOCKET_SEND_TEXT と scripts/TEST/WEBSOCKET_SEND_BINARY) は以下の様になっています。

echo_test.html | WEBSOCKET_SEND_TEXT.lua | WEBSOCKET_SEND_BINARY.lua

```
1  ----------------------------------
2  -- WebSocket テキストフレーム送信
3  ----------------------------------
4  log_msg("start..",g_script)
5
6  if not websocket_emit_text('my_app','これは /my_app/xxxxx クライアント向けのメッセージです') then error() end
7
8  if not websocket_emit_text('other_app','これは /other_app/xxxxx クライアント向けのメッセージです') then error() end
9
10 if not websocket_emit_text('これは全クライアント向けのメッセージです') then error() end
11
```

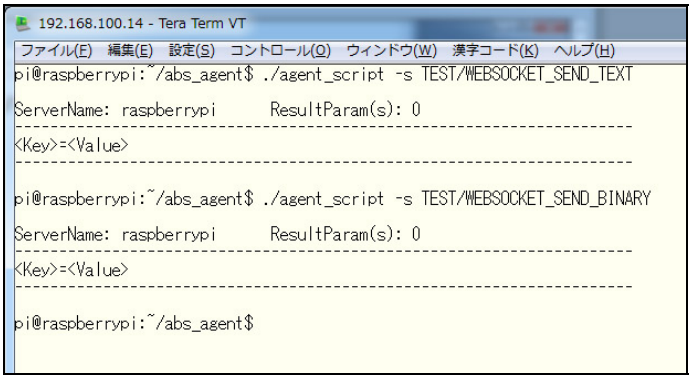
(scripts/TEST/WEBSOCKET_SEND_TEXT スクリプトの内容)



(scripts/TEST/WEBSOCKET_SEND_BINARY内容)

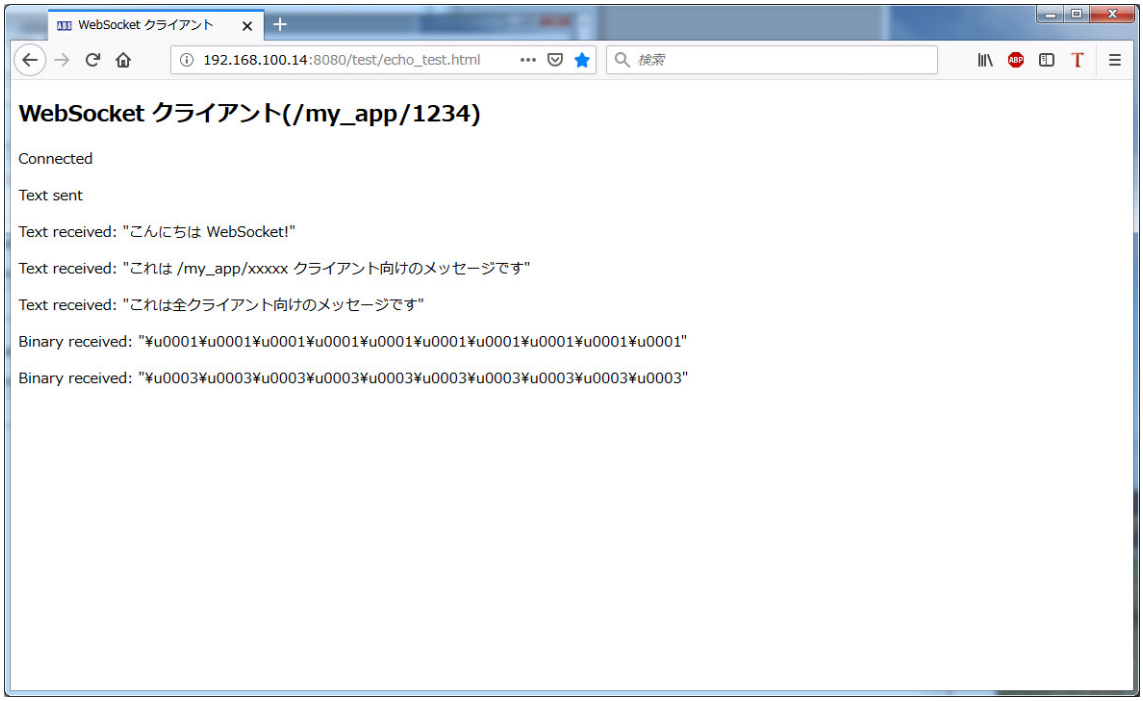
websocket_emit_text(), websocket_emit_binanry() ライブラリ関数を使用して、WebSocket サーバーから現在接続中のクライアントに対してテキストまたはバイナリフレームを送信します。送信時には、クライアント側が WebSocket オープン時に指定した URL 中のチャンネル名を使用して、送信先クライアントを限定させることもできます。

コンソールから agent_script クライアントプログラムを使用して、上記のスクリプトを実行します。



(agent_script プログラムを使用してWebSocket データ送信用スクリプトを実行した様子)

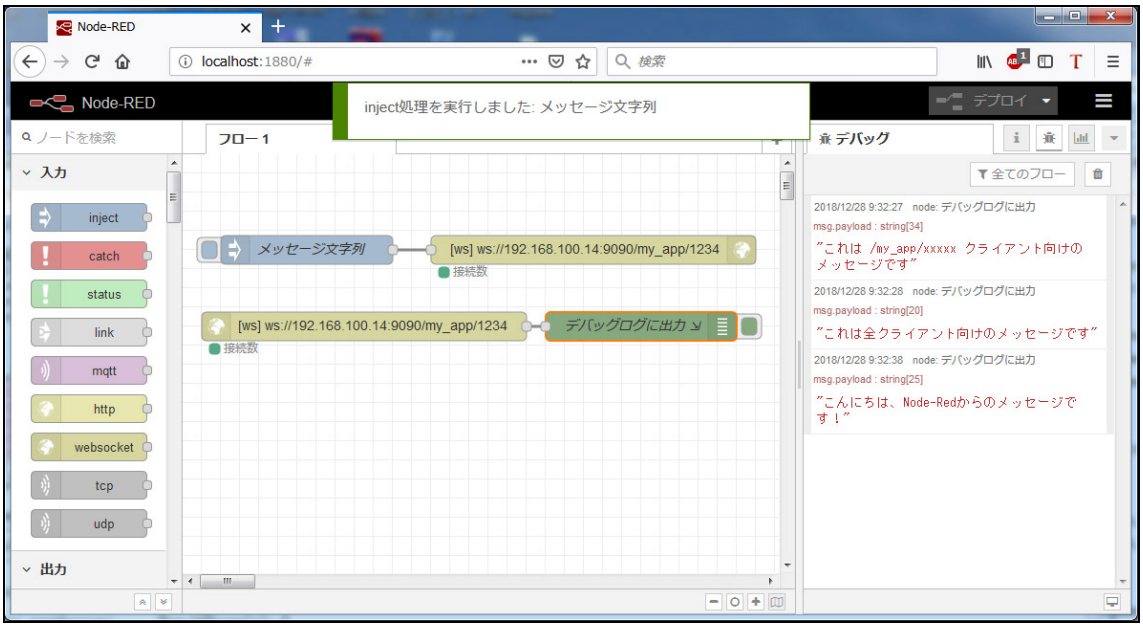
スクリプトを実行すると、WebSocket 接続を行っていたWebブラウザの画面は下記の様になります。



(WebSocket 経由でデータを受信した時の様子)

ここでは、Webブラウザ上の WebSocketクライアントを使用していましたが、他の Webサーバーやツールで WebSocket 接続をサポートしているものであれば、簡単に abs_agent に接続して双方向通信を実現できます。

例えば、オープンソースで開発されている Node-RED[8] を使用して接続した場合は下記の様になります。



(Node-RED を使用して、abs_agent のWebSocketサーバー間でメッセージを送受信した例)

[8] Node-RED https://nodered.org/

36 XBee 802.15.4 デバイス接続

この章では、Digi International の XBee 802.15.4 (Series1) デバイスを abs_agent で使用する方法を説明します。

abs_agent には シリアルポート経由で ローカル XBee デバイスを(最低１つ)接続します。この ローカル XBee デバイス経由で同一 PAN(Personal Area Network) 内の複数の リモートXBee デバイスをリモートから操作できます。AT コマンドの実行や任意のデータパケット送信などを abs_agent のスクリプトからライブラリ関数で実行できます。

リモート XBee モジュールから abs_agent に接続した XBee（ローカル XBee)にデータパケットや IOパケットを送信すると、abs_agent のイベントハンドラ SERIAL_XBEE が起動されますので、Lua スクリプトでデータ処理を記述することができます。

abs_agent では XBee デバイス（リモートデバイスを含む）管理用のクライアントプログラム agent_xbee を同梱しています。agent_xbee を使用すると、コンソールからリモート XBee デバイス一覧やディスカバリとデバイスマスター構築、AT コマンド操作、データパケット送信を簡単に実行できます。

36.1 XBee デバイス初期設定

XBee デバイスを abs_agent に接続する前に、XBee デバイス自身の初期設定を行う必要があります。設定を行うデバイスは、abs_agent のシリアルポートに直接接続するローカル XBeeデバイスとリモート側の全てのXBee デバイスが対象になります。設定する項目は以下の内容です。ここで初期設定する項目以外の XBeeデバイスの設定は、後からクライアントプログラム agent_xbee でリモート AT コマンドを使用して設定することができます。

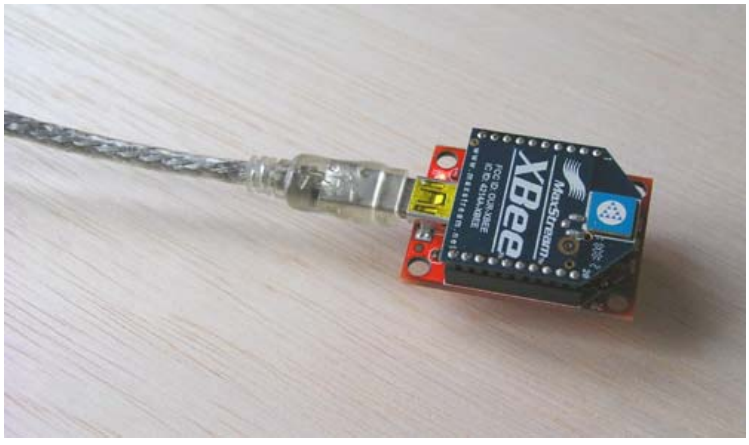
工場出荷時の XBee デバイスのデフォルト値から変更が必要な設定項目は以下になります。

XBee デバイス設定項目	設定値
API モード	1 (default は 0)
PAN(Personal Area Network) ID	任意の値 (default はOx3332) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB90 を使用しています。
16bit Source Address	同一PAN ID 内でユニークな値 (default は 0x0000) この値は、ここで設定しなくても後から agent_xbee プログラムで設定することが可能ですが、デバイス一覧から選択したデバイスがどのデバイスであるかを見分けることが容易になるように便宜的にここで設定します。

	全てのデバイス間で違った値を設定してください。 (0x0000, 0xFFFF, 0xFFFE を除く) 例えば、0x0001, 0x0002, 0x0003 等。
Node Identification	デバイスにつける任意の名前(ASCIIのみ) 16ビットアドレスやシリアル番号ではデバイスの区別が直感的に判りづらいので、判りやすい名前を付けます。 この値は、ここで設定しなくても後から agent_xbee プログラムで設定することが可能です。 例えば Device1, Node_01, DOOR_SENSOR 等
C8 Command (XBee3 Digi 802.15.4 のみ)	0x02 (default は 0x00) この設定は XBee3 に "Digi 802.15.4" ファームウエアを搭載してかつ、ローカルデバイスとして使用する場合に設定してください。また古い XBee Series1と**混在**して使用する場合のみ必要です。このパラメータを設定しない場合にはローカルデバイスで Node Discover 実行時に abs_agent でタイムアウトエラーが発生します。確認ファームウエアバージョン (0x2003)

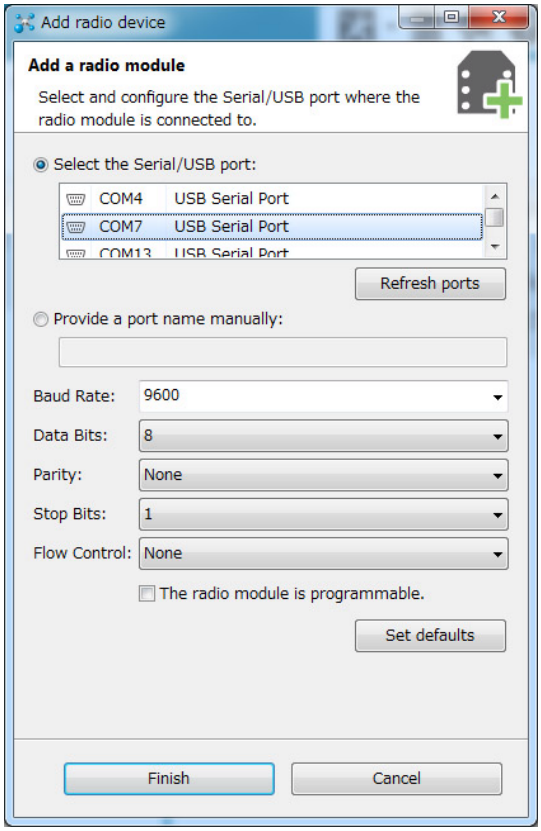
XBee デバイスを初期設定するために、Windows や Linuxデスクトップ、Mac OS が動作している初期設定作業用のコンピュータが必要です。このコンピュータで Digi international Inc. 社製の X-CTU プログラムを動作させます。X-CTU を使用しなくても、汎用のシリアル・ターミナルエミュレータプログラム等 と ATコマンドを使用して設定することも可能ですが、操作が簡単でかつ XBee ファームウエアの更新もできる X-CTU を Windows PC で使用する例で説明します。

このマニュアルでは、Windows PC に Sparkfun Electoronics 社製の XBee Explorer USB を接続して、仮想 USB ポート経由で接続しています。



(初期設定用 Windows PC に接続する XBee Explorer USBに設定対象の XBee デバイスを載せている様子)

次に Digi international Inc. のサイトから X-CTU プログラムをダウンロードしてインストールします。X-CTU プログラムを起動すると最初に XBee が接続されているポート選択画面になりますので、XBee Explorer USB が接続されている仮想 COM ポート(例では COM7)を選択します。



(XBee Explorer USB が接続されている COM7 を X-CTU で選択している様子)

XBee (XBee Explorer USB経由)のシリアルポートのボーレートは XBee デバイス初期設定値の 9600baud, 8bit, 1stop-bit, No-parity を選択します。

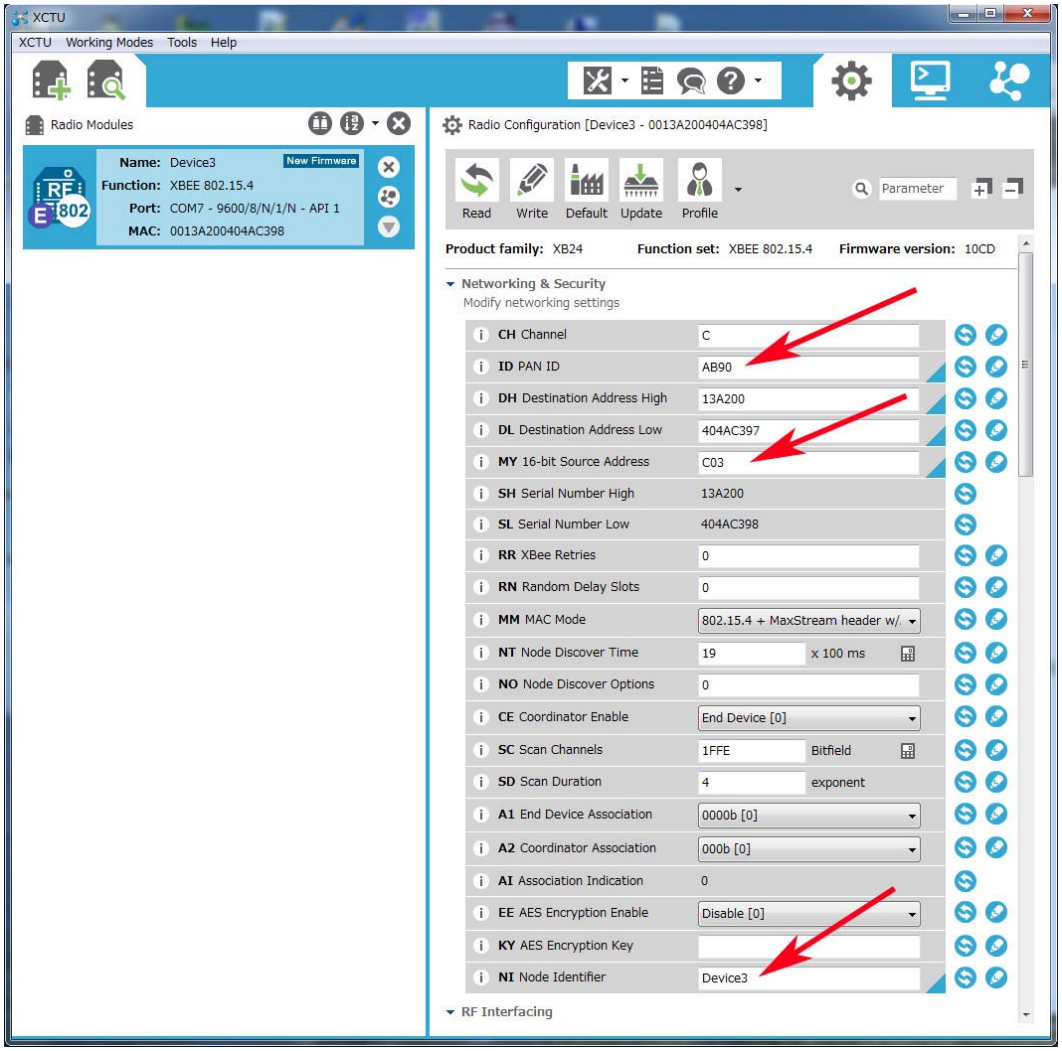
デバイスを接続すると画面左側にデバイスのアイコンが表示されます。Configuration タブを選択して現在の設定値を読み込むと設定項目一覧が右側に表示されて、ここで設定値を変更することができます。

XBee で動作しているファームウエアバージョンによって、X-CTU 設定画面のレイアウトや項目に違いがありますが、abs_agent の動作に必須な最低限の設定を変更するだけで構いません。ここで設定しなかった XBee モジュールの設定項目は、後からローカルやリモート AT コマンドを abs_agent に同梱している agent_xbee クライアントプログラムから実行することで何時でも変更できます。

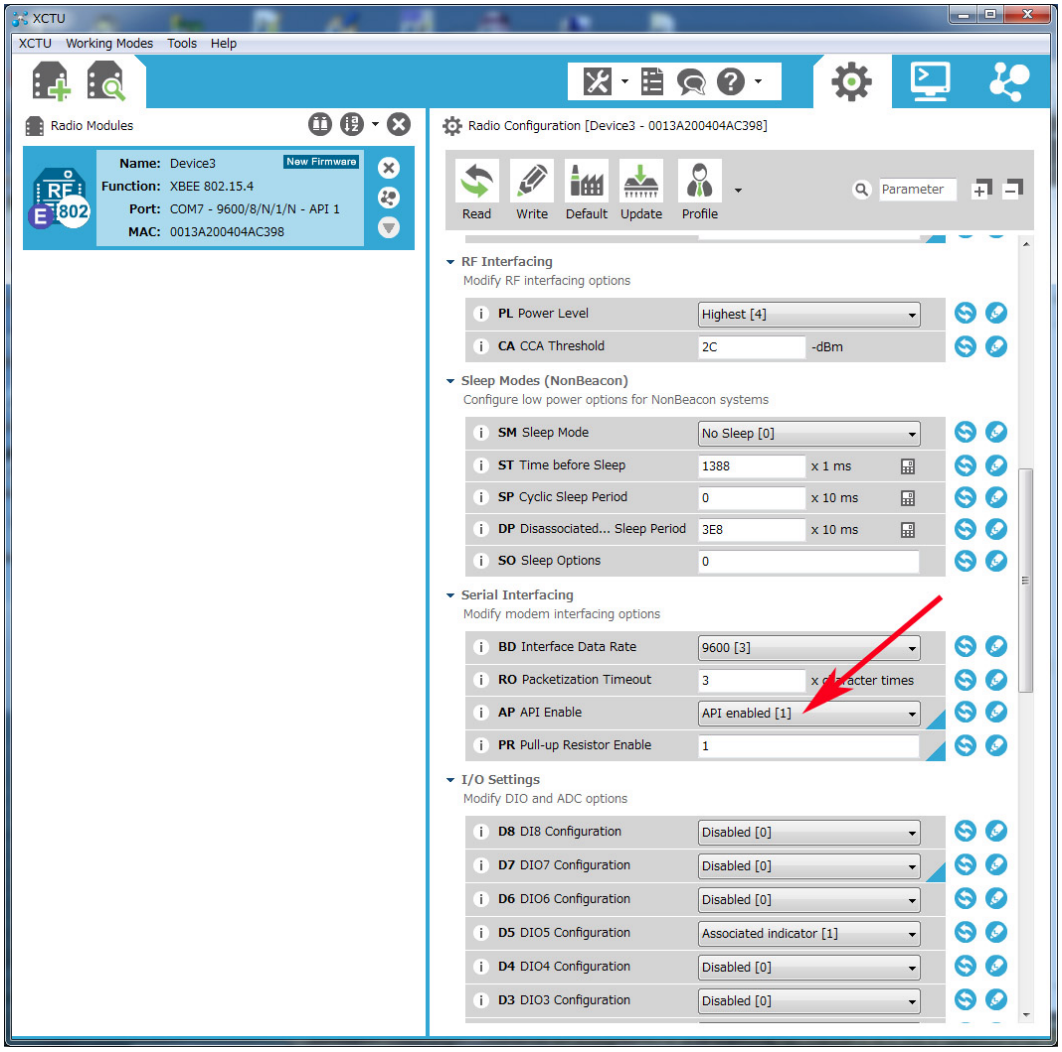
ここでは PAN ID と 16bit Source Address, Node Identification, API Mode をそれぞれ変更します。XBee3 を使用している場合には C8 Command も変更します。

Node Identification を設定するときには、デバイスに最初から設定されているスペース文字に注意してください。

X-CTU で設定するときには、確実に以前の文字列を消してからデバイス名を記入するようにしてください。



(XBee 設定画面 1/2)



(XBee 設定画面 2/2)

設定値を変更したら項目右側のアイコンを押してデバイスの書き込みと更新を行います。全ての項目の変更が終了したら X-CTU プログラムを終了します。

その後、XBee Explorer USB に接続する XBee デバイスを切り替えて、abs_agent で使用するリモート側を含む全ての XBee デバイスについて同様に初期設定を行って下さい。この時 PAN_ID は全ての XBee デバイスで同じ値を指定して、16ビットアドレスと Node Identification は全て違う値を指定するようにします。また API Mode は全ての XBee デバイスで "API Enabled [1]" に設定します。

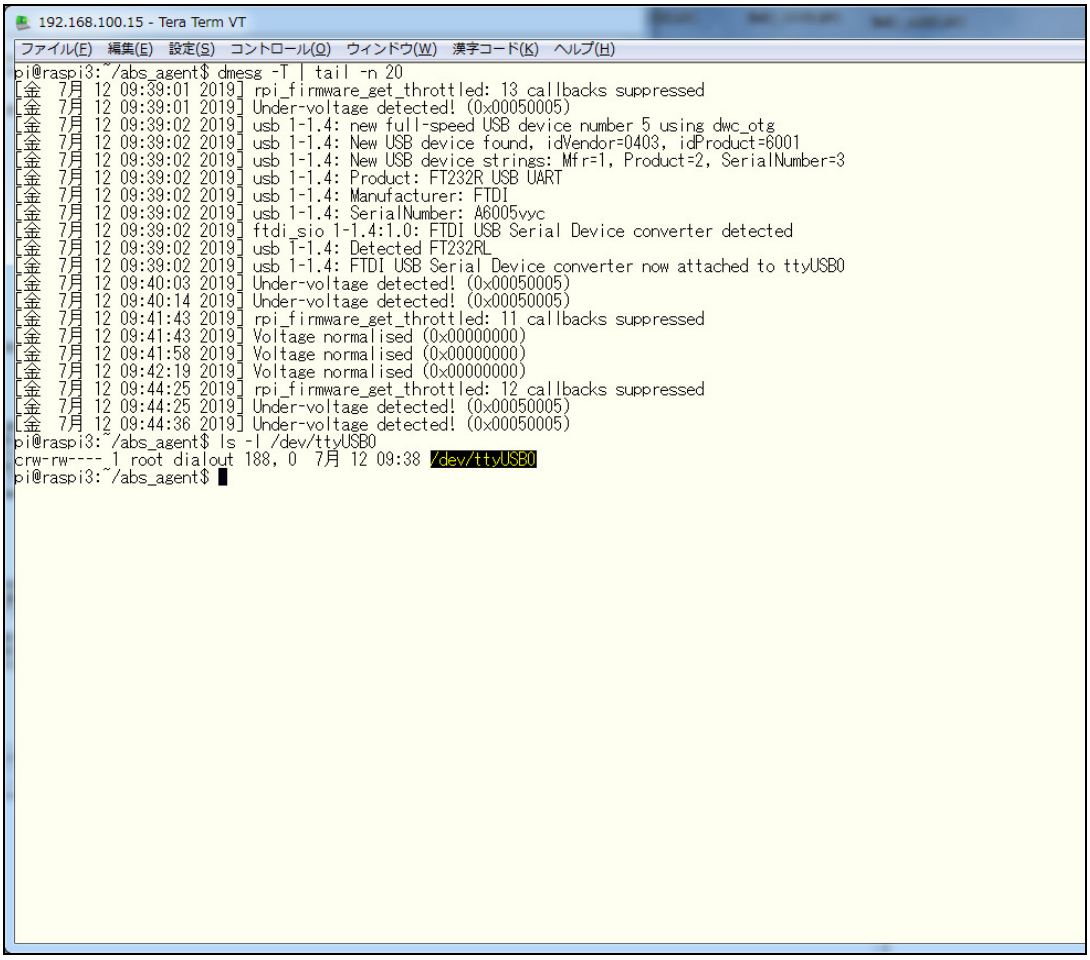
デバイス毎に設定した16bit Source Address の値や Node Identification を XBeeデバイスモジュールにマーキングしておくと、後で agent_xbee プログラムでデバイスを選択するときに、識別し易くなります。

36.2 ローカルXBeeを abs_agentが動作しているコンピュータに接続

XBee デバイスの初期設定が終了したら、初期設定作業を行った Windows PC から XBee Explorer USB を取り外してください。

次に、abs_agent のシリアルポートに接続するローカル XBee モジュールをこの XBee Explorer USB に搭載した後、abs_agent が動作しているコンピュータの USB ポートに接続します。

XBee Explorer USB が正常に接続できているかどうかは、dmesg コマンドでシステムメッセージを確認します。



(dmesg コマンドで XBee Explorer USB が接続できているかを確認している様子)

正常に接続できた場合には OS 側でアサインされたデバイス名が出力されます。上記の実行例の場合には "ttyUSB0" がアサインされていて、abs_agent のシリアルポートで使用するときのデバイスファイル名は "/dev/ttyUSB0" になります。

36.3 ローカルXBee デバイスを abs_agentシリアルデバイスに登録

ローカル XBee デバイスを abs_agent に接続するときには abs_agent のサーバー設定ファイル (abs_agent.xml) に、シリアルデバイスとしてエントリを登録します。シリアルポートで使用する Linux デバイスファイル名や通信条件を XML ファイルのタグとして設定ファイルに書き込みます。

ここでは前項で接続した XBee Explorer USB に搭載した XBee 802.15.4 デバイスを abs_agent に登録する例で説明します。abs_agent プログラムは Raspberry Pi用にビルドされたものを使用しています。PC/AT 互換機上で動作する abs_agent に接続する場合も設定方法は同じですが、接続するシリアルポートのデバイスファイル名は OS 上

で設定されたものに合わせてください。

接続する ローカル XBee デバイスの通信条件は下記の様になります。

ローカル XBee デバイス通信条件	
通信プロトコル	XBee 802.15.4 API Mode 1
通信インターフェース	RS-232C
接続するデバイスファイル	/dev/ttyUSB0
ボーレート	9600
ビット長	8
ストップビット	1
パリティ	無し
フロー制御	無し

サーバー設定ファイルをエディタで開いて上記のシリアルデバイスを登録します。デフォルトのサーバー設定ファイルは abs_agent をインストールしたディレクトリにファイル名 abs_agent.xml で格納されています。

```
vi abs_agent.xml
```

サーバー設定ファイル中の下記の部分を

```
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList/>
  </Serial>
```

以下の様に変更します。<DeviceList/> の部分を <DeviceList><Item> ... </Item></DeviceList>の様に変更して、<Item> .. </Item> 内にシリアルデバイス設定項目を格納します。デバイスタイプを示す <Type>XBEE</Type> 部分が XBee 802.15.4 (API Mode 1)プロトコルで通信することを指定しています。サーバー設定ファイル中の各タグの詳細については、"サーバープログラム・設定ファイル" の章を参照してください。

```
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList>
        <Item>
          <COMPort>/dev/ttyUSB0</COMPort>
          <Type>XBEE</Type>
          <BufferedMode>False</BufferedMode>
          <BaudRate>9600</BaudRate>
          <DataBits>8</DataBits>
```

```
            <StopBits>1</StopBits>
            <ParityBit>NONE</ParityBit>
            <FlowControl>NONE</FlowControl>
            <Title>XBee_raspi3</Title>
        </Item>
      </DeviceList>
    </Serial>
```

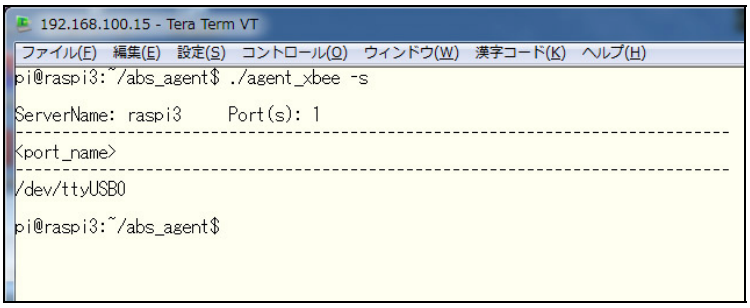
XML フォーマットにエラーがあると abs_agent 起動時にエラーになりますので間違えないようにしてください。XML ファイルを編集したときには、ファイルをWebブラウザ等で読み込ませると簡単にフォーマットのエラーをチェックすることができます。

サーバー設定ファイルの編集が完了したら abs_agent を再起動させて新しいサーバー設定を有効にします。agent_shutdown 停止コマンドを実行して、完全にサーバーが停止するまで 10 秒程度待った後に abs_agent を起動しています。もし、abs_agent の自動起動を /etc/rc.local 等に設定している場合には OS を再起動する方法でも構いません。

```
./agent_shutdown
sudo ./abs_agent -I 192.168.100.45
```

36.4 PAN 内のXBee デバイスをマスターに登録・更新

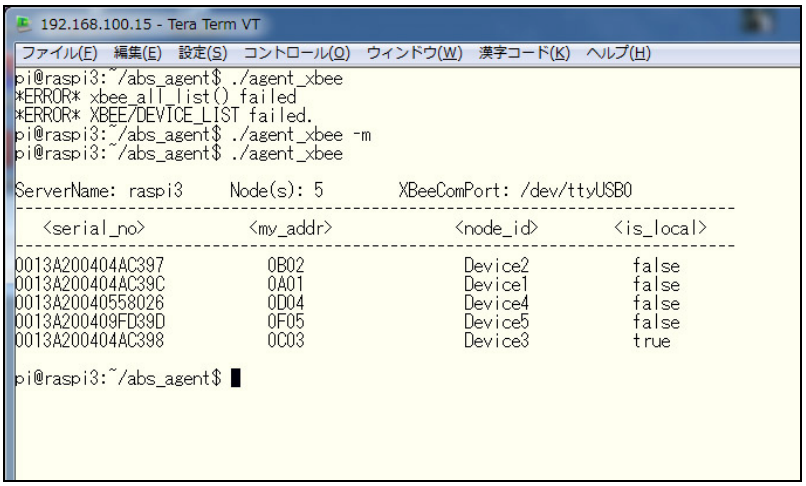
abs_agent 起動した後、正常に ローカル XBee がシリアルポートに接続されているかどうかを確認します。agent_xbee プログラムを -s オプション付きで実行すると、"XBEE"タイプで設定されているシリアルポート一覧を表示します。



(XBEE デバイスタイプのシリアルポート一覧を表示)

次にローカル XBee デバイス(上記シリアルポートに接続している XBee) と同一 PAN 内にある全ての XBee モジュールを探索して abs_agent 側にデバイス・マスターを登録・更新します。このマスターには XBeeデバイスのアドレスや Node Identification の関連付けが記録されていて、ライブラリ関数からリモート XBee デバイスをアクセスするときのアドレス解決に使用します。

agent_xbee コマンドを -m オプション付きで実行すると、abs_agent のシリアルポートに接続された ローカルXBee デバイスで “Node Discover” が実行され、付近にある同一 PAN ID の XBee デバイス情報を取得して、自動的にマスターファイルに登録します。agent_xbee プログラムをオプション指定なしで実行すると、マスターに登録済みの XBee デバイス一覧を表示します。



（マスターに同一 PAN 内の XBee デバイスを登録した様子）

デバイス一覧が表示されると、それらの XBee デバイスに対して AT コマンド実行やデータパケット送受信をライブラリ関数や agent_xbee コマンドでアクセスできます。

上記のマスター登録を実行する時には、必ずリモート側の XBee デバイスに電源を接続して PAN ネットワークに参加している状態で作業してください。

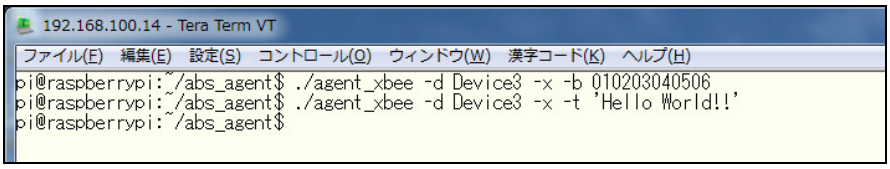
マスターは一度登録すると、XBee デバイスの16ビットアドレスや Node Identification を変更しない限り再実行する必要はありません。PAN 内に新規デバイスを追加した場合やデバイスアドレス(16ビットアドレス), Node Identification 変更時には -m オプション付きで agent_xbee コマンドを再実行して下さい。

36.5 XBee デバイス操作例

ここでは abs_agent を 2 台使用して、それぞれに XBee デバイスを接続した状態でデータの送信を行います。

前項までの設定例では XBee デバイスの Node Identificationが "Device3" のものが 192.168.100.15 の IP アドレスのコンピュータで動作している abs_agent に接続しています。PAN 内のマスター一覧で表示されている Device5 にも同様にセットアップされた abs_agent (IPアドレス 192.168.100.14)が接続されています。ちなみに 192.168.100.14 の abs_agent では Device5 がローカル XBee デバイスになります。

ここでは、192.168.100.14 のコンソールから、Device3 に対してデータを送信してみます。



(agent_xbee コマンドを使用してデータパケット送信)

このとき、受信側の abs_agent のイベントハンドラ SERIAL_XBEE が起動されて送信されたデータフレーム全体または、データフレーム中のペイロードを文字列に変換したものをログに出力します。(デフォルト動作)

```
2019/07/19 10:15:32 raspi3     SERIAL_XBEE          0 Device5[/dev/ttyUSB0] 7E000B810F053600010203040506 1F
2019/07/19 10:15:34 raspi3     SERIAL_XBEE          0 Device5[/dev/ttyUSB0] Hello World!!
```

(XBee データパケット受信時のログ)

ユーザーは SERIAL_XBEE イベントハンドラ中に Lua スクリプトを記述してシステムを作成できます。

これ以外にも agent_xbee プログラムでは、ローカルやリモートの XBee モジュールで ATコマンドを実行して設定値を参照したり変更することができます。詳しい動作例は "クライアントプログラム"章中の agent_xbee の項を参照してください。

37 XBee-ZB Zigbee デバイス接続

この章では、Digi International の XBee-ZB Zigbee(Series2) デバイスを abs_agent で使用する方法を説明します。前章で説明した XBee 802.15.4 デバイスとは独立したシリアルデバイスを使用しますので、XBee-ZB Zigbee ネットワークと XBee 802.1.54 のネットワークを同時に 1 つの abs_agent で管理することもできます。

abs_agent には シリアルポート経由で ローカル XBee-ZB デバイスを(最低１つ)接続します。この ローカル XBee-ZB デバイス経由で同一 PAN(Personal Area Network) 内の複数の リモートXBee-ZB デバイスをリモートから操作できます。AT コマンドの実行や任意のデータパケット送信などを abs_agent のスクリプトからライブラリ関数で実行できます。

ローカル XBee-ZB デバイスには Zigbee ネットワークを構成する 3つのデバイスタイプの中で、"coordinator" と "router" タイプのものを使用できます。"end device" タイプの XBee-ZB デバイスをローカルデバイスとして abs_agent に接続することはできません。リモート側の XBee-ZB デバイスは "end device" タイプを含む全てのタイプのデバイスを abs_agent から操作することができます。

リモート XBee-ZB モジュールから abs_agent に接続した XBee-ZB (ローカル XBee-ZB)にデータパケットや IOパケットを送信すると、abs_agent のイベントハンドラ SERIAL_ZB が起動されますので、Lua スクリプトでデータ処理を記述することができます。

abs_agent では XBee-ZB デバイス（リモートデバイスを含む）管理用のクライアントプログラム agent_zb を同梱しています。agent_zb を使用すると、コンソールからリモート XBee-ZB デバイス一覧やディスカバリとデバイスマスター構築、AT コマンド操作、データパケット送信を簡単に実行できます。

37.1 XBee-ZB デバイス初期設定

XBee デバイスを abs_agent に接続する前に、XBee デバイス自身の初期設定を行う必要があります。設定を行うデバイスは、abs_agent のシリアルポートに直接接続するローカル XBeeデバイスとリモート側の全てのXBee デバイスが対象になります。

XBee-ZB デバイスで Zigbee ネットワークを構成するときには "coordinator", "router", "end device" の３つのデバイスタイプを設定することができます。この中で、abs_agent にシリアルポートで接続するローカルデバイスには "coordinator" または "router" の設定を行って下さい。リモート側の XBee-ZB デバイスタイプにはどのデバイスタイプを設定しても構いません。

設定する項目は以下の内容です。ここで設定しなかった XBee デバイスの設定項目は、後からローカルやリモート AT コマンドを abs_agent に同梱している agent_zb クライアントプログラムから実行することで何時でも変更できます。

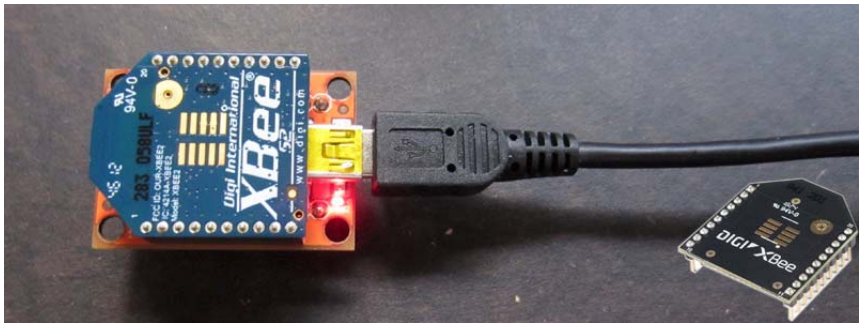
工場出荷時の XBee-ZB デバイスのデフォルト値から変更が必要な設定項目は以下になります。

XBee-ZB デバイス設定項目(対応する ATコマンド)	設定値
XBee-ZB ファームウエア	** XBee3 以前の古いモジュールの場合 ** デバイスタイプ毎に XBee-ZB モジュールで動作させるファームウエアを書き換える必要があります。abs_agent のシリアルポートに直接接続するデバイスには、ZIGBEE COORDINATOR API または ZIGBEE ROUTER API のどちらかのファームウエアを書き込みます。その他のリモート側のデバイスは ZIGBEE END DEVICE API を含む全てのデバイスタイプ用のファームウエア使用できます。 ** XBee3 の場合 ** 以降の AT コマンド設定値によってデバイスタイプが切り替わりますので、ローカルとリモート XBee-ZB ファームウエアには全て同じ Digi XBee3 Zigbee 3.0 xx を使用します。
Device Role(CE)	** XBee3 の場合のみ必要 ** 0(Join Network) または 1(Form Network) に設定する。 "coordinator" デバイスタイプにする場合には 1 に設定して、"router" デバイスタイプの場合には 0 に設定します。
API Enable(AP)	1 (default は 0)

	全ての ローカルとリモート XBee-ZB デバイスは API Mode 1 に設定します。
Extended PAN ID(ID)	任意の値 (default は0x0) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB9000 を使用しています。
Node Identification(NI)	デバイスにつける任意の名前(ASCIIのみ) シリアル番号ではデバイスの区別が直感的に判りづらいので、判りやすい名前を付けます。 この値は、ここで設定しなくても後から agent_zb プログラムで設定することが可能です。 例えば Device1, Node_01, DOOR_SENSOR 等
Sleep Mode(SM)	abs_agent のローカルデバイスに使用する XBee-ZB の場合には必ず 0 を設定します。リモート側のデバイスには任意の値を指定できます。 "coordinator" や "router" デバイスタイプには 0 を指定する必要があります。

XBee-ZB デバイスを初期設定するために、Windows や Linuxデスクトップ、Mac OS が動作している初期設定作業用のコンピュータが必要です。このコンピュータで Digi international Inc. 社製の X-CTU プログラムを動作させます。X-CTU を使用しなくても、汎用のシリアル・ターミナルエミュレータプログラム等 と ATコマンドを使用して設定することも可能ですが、操作が簡単でかつ XBee ファームウエアの更新もできる X-CTU を Windows PC で使用する例で説明します。

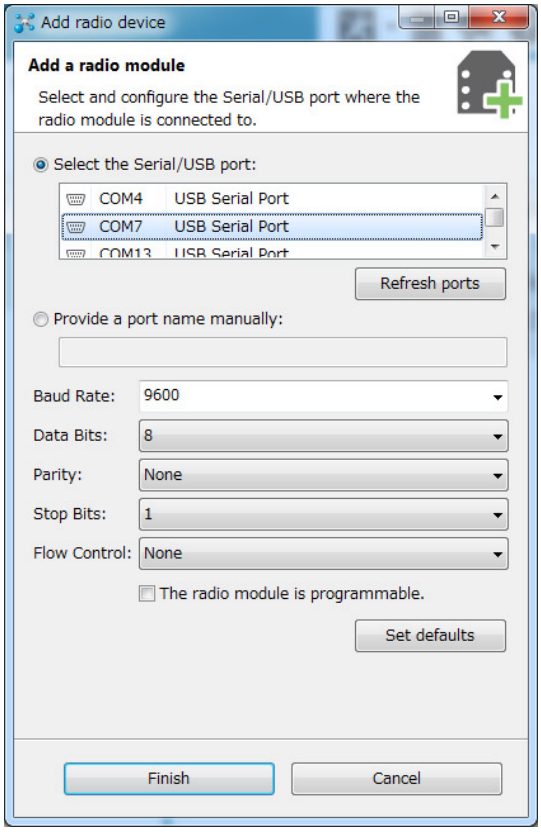
このマニュアルでは、Windows PC に Sparkfun Electoronics 社製の XBee Explorer USB を接続して、仮想 USB ポート経由で接続しています。



(初期設定用 Windows PC に接続する XBee Explorer USBに設定対象の XBee-ZB デバイスを載せている様子。右下の XBee3 デバイスを搭載することもできます)

次に Digi international Inc. のサイトから X-CTU プログラムをダウンロードしてインストールします。X-CTU プログラムを起動すると最初に XBee が接続されているポート選択画面になりますので、XBee Explorer USB が接続さ

れている仮想 COM ポート(例では COM7)を選択します。



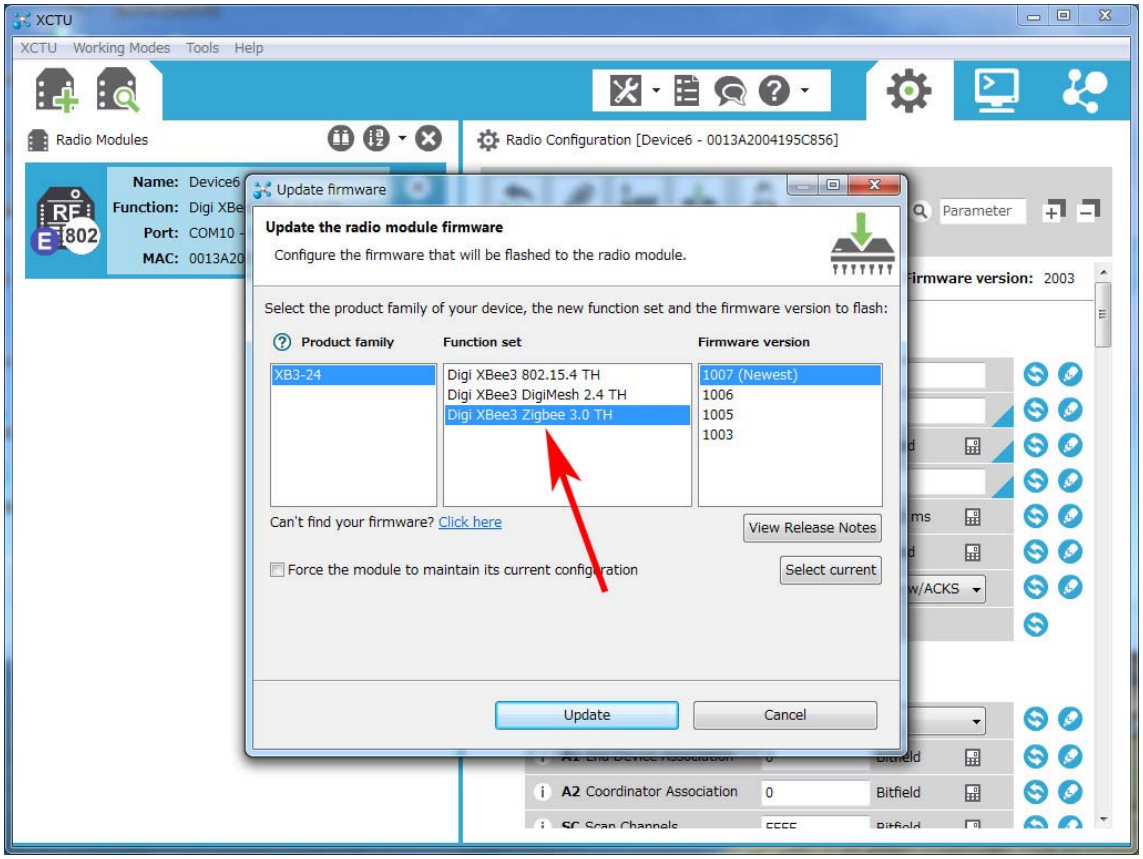
(XBee Explorer USB が接続されている COM7 を X-CTU で選択している様子)

XBee (XBee Explorer USB経由)のシリアルポートのボーレートは XBee デバイス初期設定値の 9600baud, 8bit, 1stop-bit, No-parity を選択します。

デバイスを接続すると画面左側にデバイスのアイコンが表示されます。Configuration タブを選択して現在の設定値を読み込むと設定項目一覧が右側に表示されて、ここで設定値を変更することができます。

XBee-ZB で動作しているファームウエアバージョンによって、X-CTU 設定画面のレイアウトや項目に違いがありますが、abs_agent の動作に必須な最低限の設定を変更するだけで構いません。ここで設定しなかった XBee-ZBモジュールの設定項目は、後からローカルやリモート AT コマンドを agent_zb プログラムから実行することで何時でも変更できます。

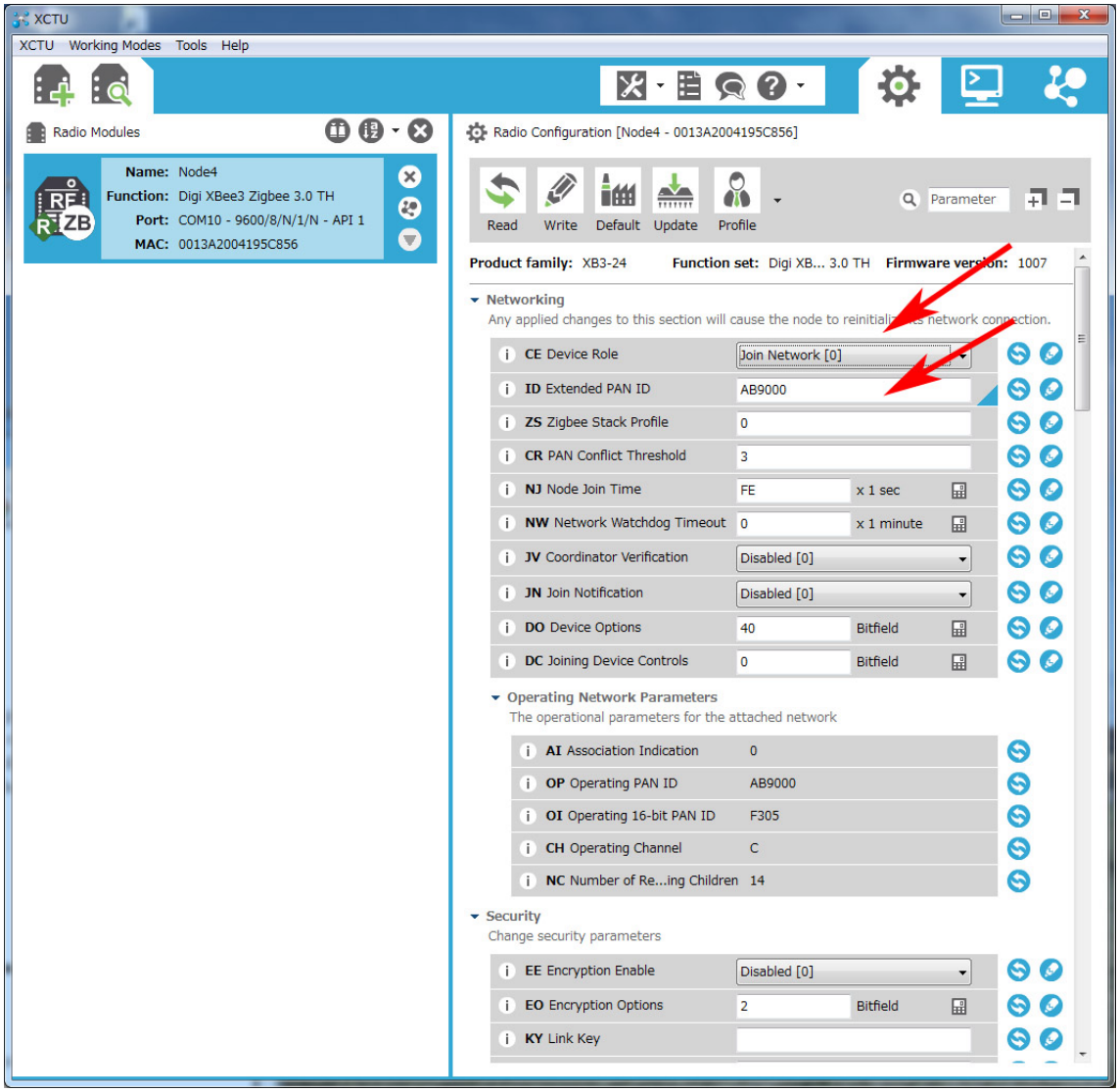
このマニュアルでは XBee3 に Zigbee ファームウエアを書き込んだ後、"router" デバイスタイプで abs_agent に接続します。XBee-ZB Series2 デバイスを使用する場合には ZIGBEE ROUTER API のファームウエアを書きこんだ後設定項目を変更します。



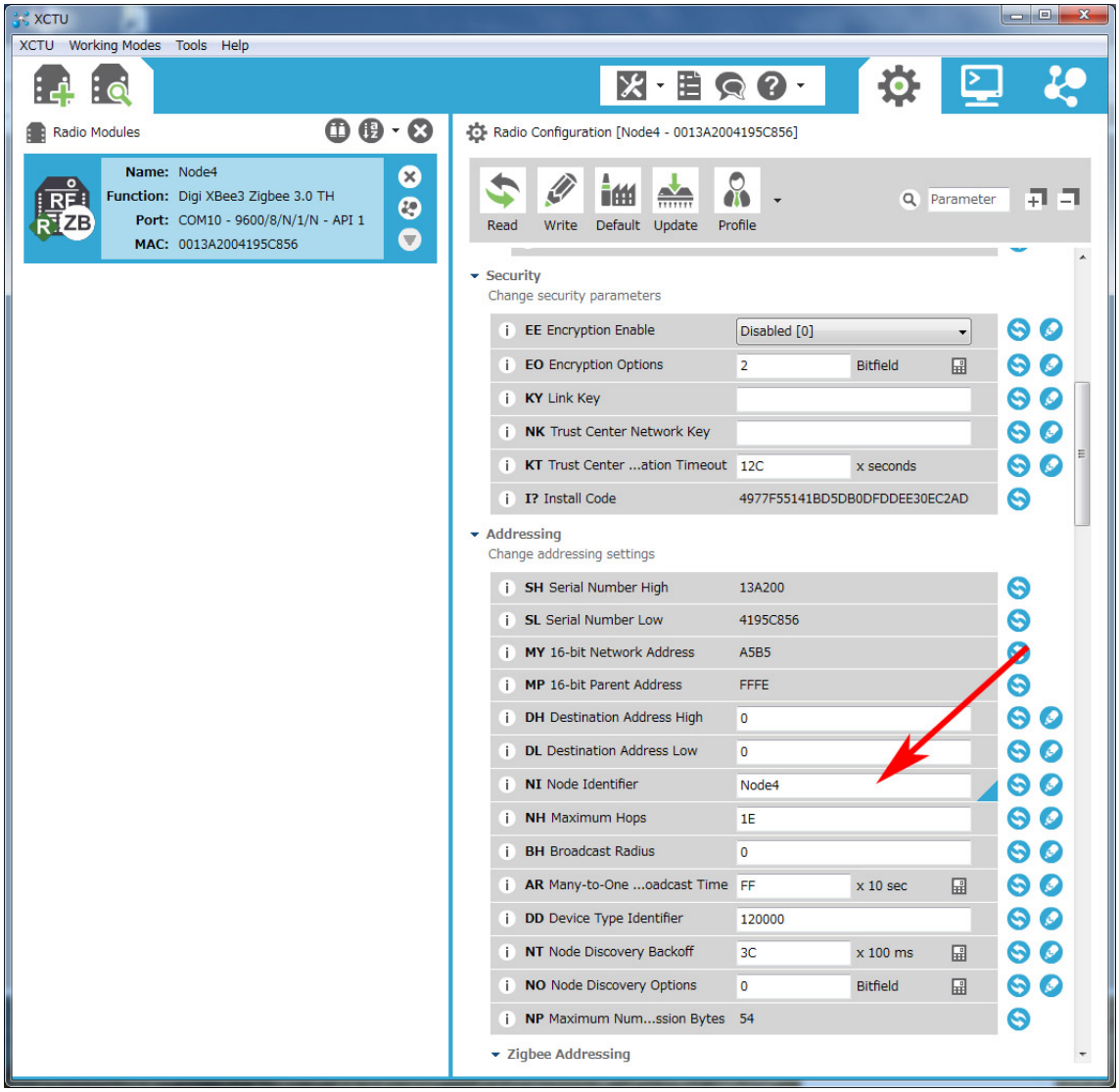
（既存の XBee3 デバイスのファームウエアを上書きして、最新の XBee-ZB Zigbee タイプのファームウエアに変更している様子）

ファームウエアの更新が完了したら、前項で説明した XBee-ZB デバイスの初期設定を行います。ここでは API Enable, Extended PAN ID, Sleep Mode と Node Identification をそれぞれ変更します。XBee3 を使用している場合には Device Role も変更します。

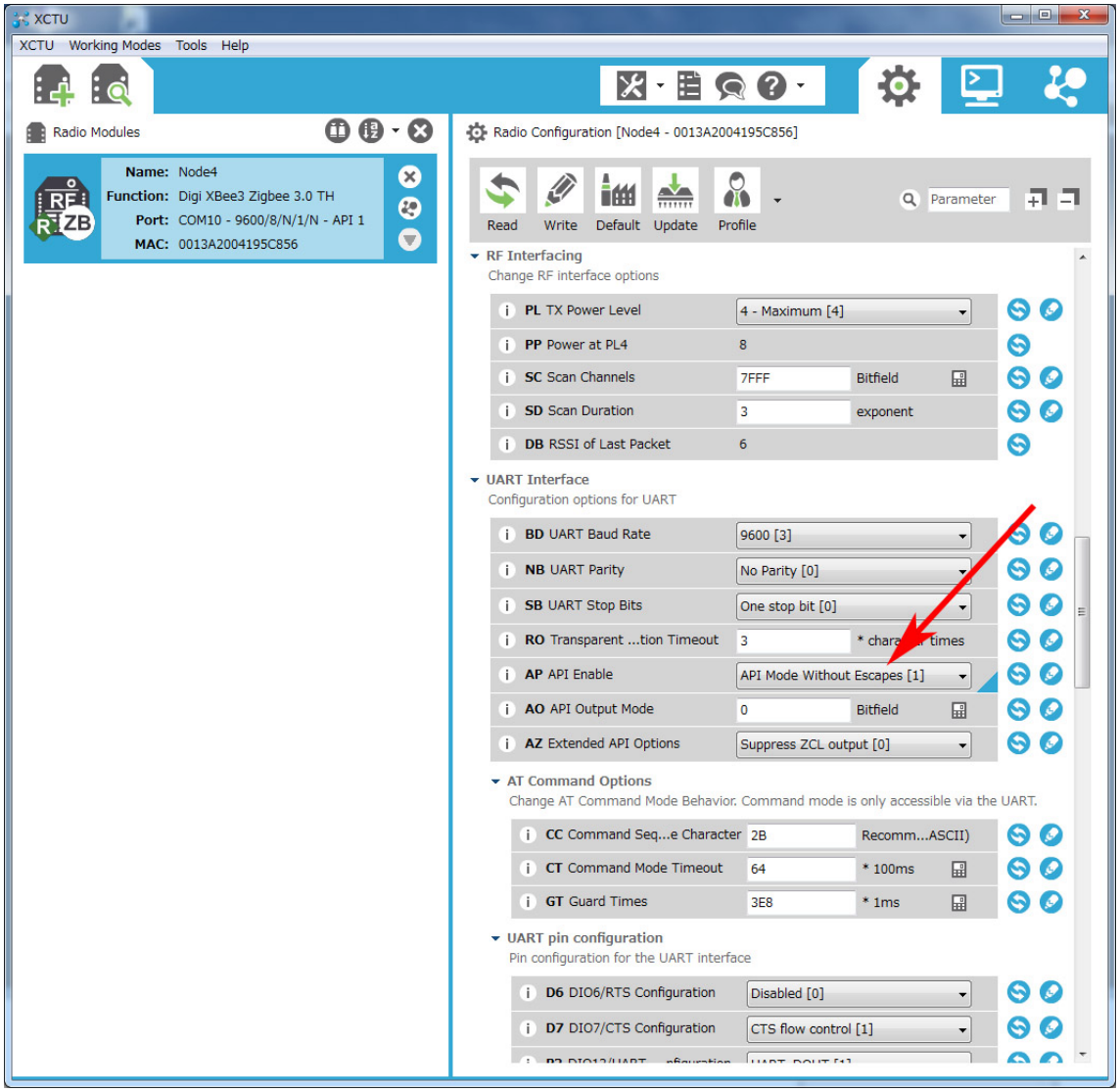
Node Identification を設定するときには、デバイスに最初から設定されているスペース文字に注意してください。X-CTU で設定するときには、確実に以前の文字列を消してからデバイス名を記入するようにしてください。



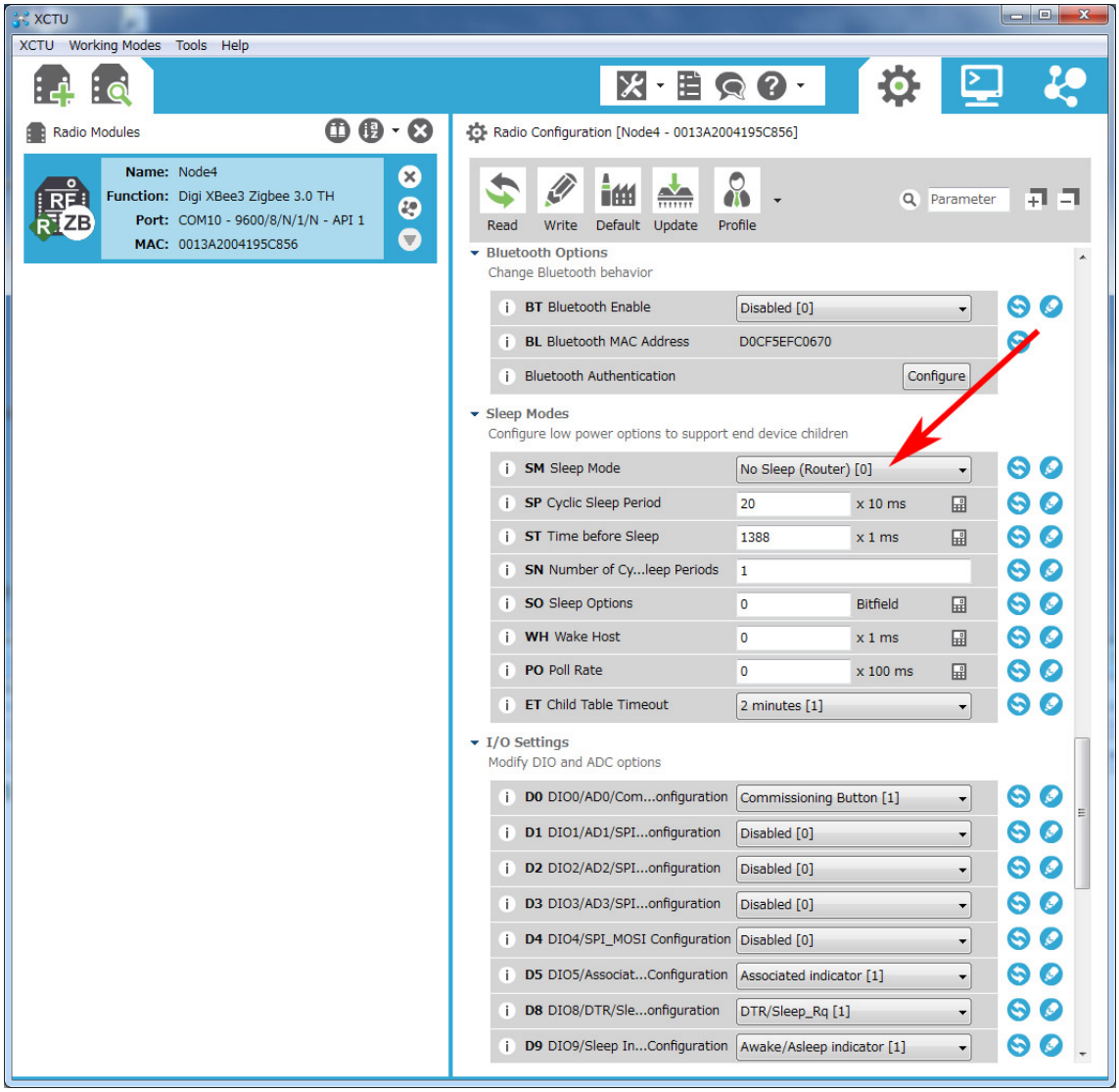
(XBee-ZB 設定画面 1/4)



(XBee-ZB 設定画面 2/4)



(XBee-ZB 設定画面 3/4)



(XBee-ZB 設定画面 4/4)

設定値を変更したら項目右側のアイコンを押してデバイスの書き込みと更新を行います。全ての項目の変更が終了したら X-CTU プログラムを終了します。

その後、XBee Explorer USB に接続する XBee-ZB デバイスを切り替えて、abs_agent で使用するリモート側を含む全ての XBee-ZB デバイスについて同様に初期設定を行って下さい。この時 Extended PAN ID は全ての XBee-ZB デバイスで同じ値を指定して、Node Identification は全て違う値を指定するようにします。また API Mode は全ての XBee デバイスで "API Mode Without Escapes [1]" に設定します。

デバイス毎に設定した Node Identification を XBee-ZB デバイスモジュールにマーキングしておくと、後で agent_zb プログラムでデバイスを選択するときに、識別し易くなります。

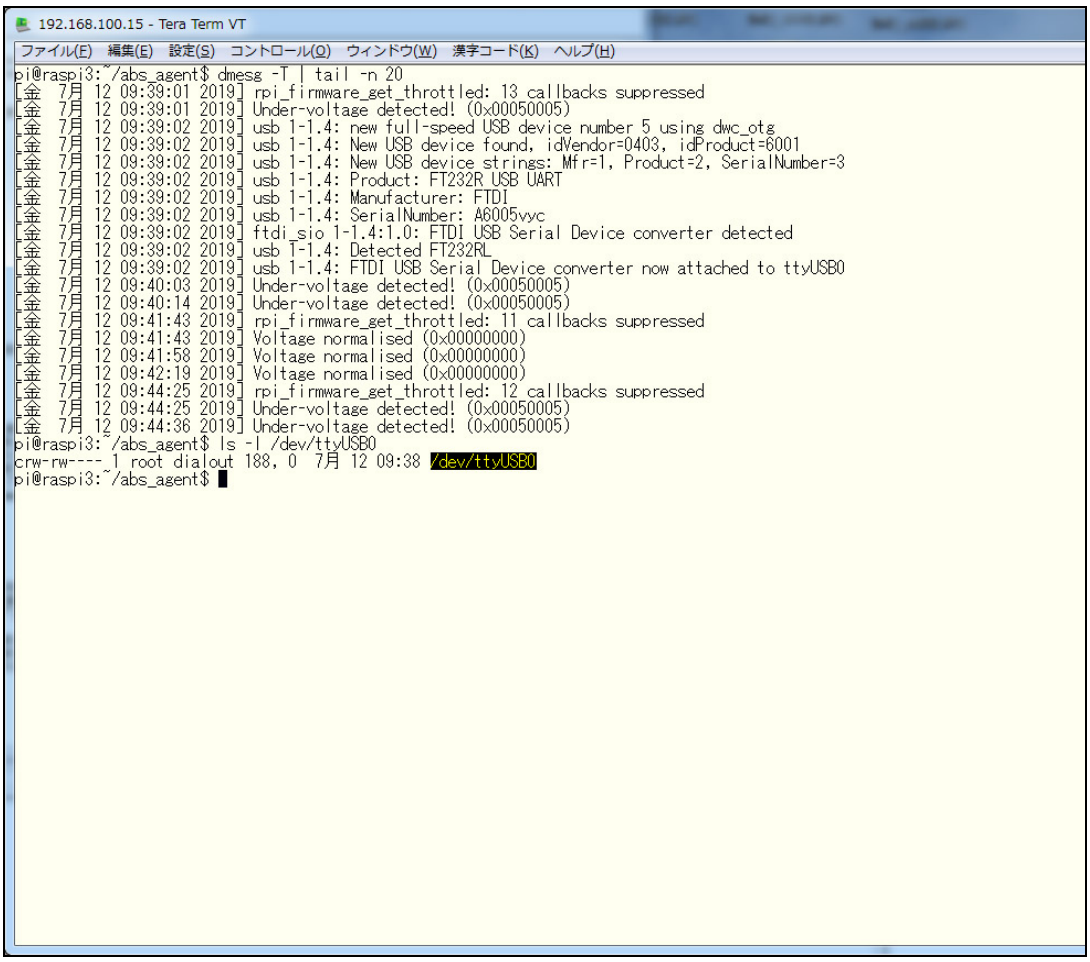
37.2 ローカルXBee-ZBを abs_agentが動作しているコンピュータに接続

XBee-ZB デバイスの初期設定が終了したら、初期設定作業を行った Windows PC から XBee Explorer USB を取り外

してください。

次に、abs_agent のシリアルポートに接続するローカル XBee-ZB モジュールをこの XBee Explorer USB に搭載した後、abs_agent が動作しているコンピュータの USB ポートに接続します。

XBee Explorer USB が正常に接続できているかどうかは、dmesg コマンドでシステムメッセージを確認します。



(dmesg コマンドで XBee Explorer USB が接続できているかを確認している様子)

正常に接続できた場合には OS 側でアサインされたデバイス名が出力されます。上記の実行例の場合には "ttyUSB0" がアサインされていて、abs_agent のシリアルポートで使用するときのデバイスファイル名は "/dev/ttyUSB0" になります。

37.3 ローカルXBee-ZB デバイスを abs_agentシリアルデバイスに登録

ローカル XBee-ZB デバイスを abs_agent に接続するときには abs_agent のサーバー設定ファイル (abs_agent.xml) に、シリアルデバイスとしてエントリを登録します。シリアルポートで使用する Linux デバイスファイル名や通信条件を XML ファイルのタグとして設定ファイルに書き込みます。

ここでは前項で接続した XBee Explorer USB に搭載した XBee-ZB デバイスを abs_agent に登録する例で説明しま

す。abs_agent プログラムは Raspberry Pi用にビルドされたものを使用しています。PC/AT 互換機上で動作する abs_agent に接続する場合も設定方法は同じですが、接続するシリアルポートのデバイスファイル名は OS 上で設定されたものに合わせてください。

接続する ローカル XBee-ZB デバイスの通信条件は下記の様になります。今回ローカル XBee-ZB デバイスには Zigbee "router" デバイスタイプに設定したモジュールを接続していますが、"coordinator" タイプの場合でも abs_agent 側の設定ファイルの内容は同じです。

ローカル XBee デバイス通信条件	
通信プロトコル	XBee-ZB API Mode 1
通信インターフェース	RS-232C
接続するデバイスファイル	/dev/ttyUSB0
ボーレート	9600
ビット長	8
ストップビット	1
パリティ	無し
フロー制御	無し

サーバー設定ファイルをエディタで開いて上記のシリアルデバイスを登録します。デフォルトのサーバー設定ファイルは abs_agent をインストールしたディレクトリにファイル名 abs_agent.xml で格納されています。

```
vi abs_agent.xml
```

サーバー設定ファイル中の下記の部分を

```
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList/>
  </Serial>
```

以下の様に変更します。<DeviceList/> の部分を <DeviceList><Item> ... </Item></DeviceList>の様に変更して、<Item> .. </Item> 内にシリアルデバイス設定項目を格納します。デバイスタイプを示す <Type>ZB</Type> 部分が XBee-ZB (API Mode 1)プロトコルで通信することを指定しています。サーバー設定ファイル中の各タグの詳細については、"サーバープログラム・設定ファイル" の章を参照してください。

```
  <Serial>
    <AutoOnline type="boolean">True</AutoOnline>
    <DeviceList>
      <Item>
        <Enabled>True</Enabled>
```

```
        <COMPort>/dev/ttyUSB0</COMPort>
        <Type>ZB</Type>
        <BufferedMode>False</BufferedMode>
        <BaudRate>9600</BaudRate>
        <DataBits>8</DataBits>
        <StopBits>1</StopBits>
        <ParityBit>NONE</ParityBit>
        <FlowControl>NONE</FlowControl>
        <Title>XBee_ZB</Title>
      </Item>
    </DeviceList>
  </Serial>
```

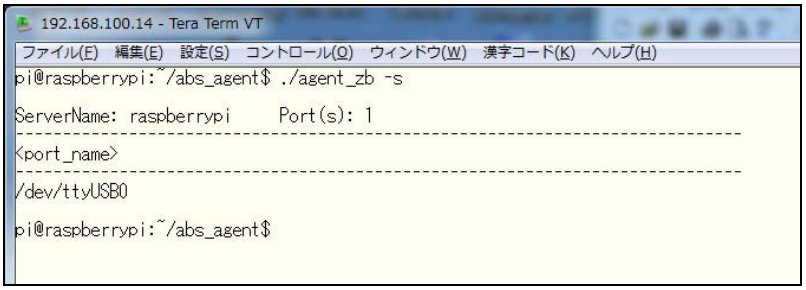
XML フォーマットにエラーがあると abs_agent 起動時にエラーになりますので間違えないようにしてください。XML ファイルを編集したときには、ファイルをWebブラウザ等で読み込ませると簡単にフォーマットのエラーをチェックすることができます。

サーバー設定ファイルの編集が完了したら abs_agent を再起動させて新しいサーバー設定を有効にします。agent_shutdown 停止コマンドを実行して、完全にサーバーが停止するまで 10 秒程度待った後に abs_agent を起動しています。もし、abs_agent の自動起動を /etc/rc.local 等に設定している場合には OS を再起動する方法でも構いません。

```
./agent_shutdown
sudo ./abs_agent -I 192.168.100.45
```

37.4 PAN 内のXBee-ZB デバイスをマスターに登録・更新

abs_agent 起動した後、正常に ローカル XBee-ZB がシリアルポートに接続されているかどうかを確認します。agent_zb プログラムを -s オプション付きで実行すると、"ZB"タイプで設定されているシリアルポート一覧を表示します。

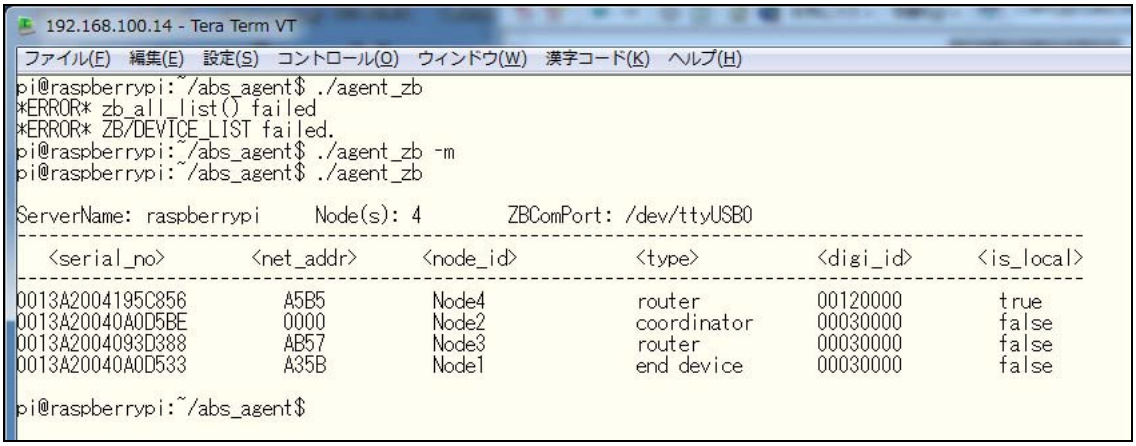


("ZB" デバイスタイプのシリアルポート一覧を表示)

次にローカル XBee-ZB デバイス(上記シリアルポートに接続している XBee-ZB) と同一 PAN 内にある全ての

XBee-ZB モジュールを探索して abs_agent 側にデバイス・マスターを登録・更新します。このマスターには XBee-ZB デバイスのアドレスや Node Identification の関連付けが記録されていて、ライブラリ関数からリモート XBee-ZB デバイスをアクセスするときのアドレス解決に使用します。

agent_zb コマンドを -m オプション付きで実行すると、abs_agent のシリアルポートに接続された ローカル XBee-ZB デバイスで “Network Discovery” が実行され、付近にある同一 PAN ID の XBee-ZB デバイス情報を取得して、自動的にマスターファイルに登録します。agent_zb プログラムをオプション指定なしで実行すると、マスターに登録済みの XBee-ZB デバイス一覧を表示します。



（マスターに同一 PAN 内の XBee-ZB デバイスを登録した様子）

デバイス一覧が表示されると、それらの XBee-ZB デバイスに対して AT コマンド実行やデータパケット送受信をライブラリ関数や agent_zb コマンドでアクセスできます。

上記のマスター登録を実行する時には、必ずリモート側の XBee-ZB デバイスに電源を接続して PAN ネットワークに参加している状態で作業してください。今回の例では、Zigbee "coordinator" デバイスタイプのモジュールがリモート側にありますので、事前に設置を済ませてからデバイスの検索とマスター登録を行います。

マスターは一度登録すると、XBee-ZB デバイスの Node Identification を変更しない限り再実行する必要はありません。PAN 内に新規デバイスを追加した場合や Node Identification 変更時には -m オプション付きで agent_xbee コマンドを再実行して下さい。

XBee-ZB の 16ビットアドレス(Network Address)は abs_agent 内で自動的に管理されます

agent_zb プログラムをパラメータ無しで実行したときに表示される〈net_addr〉(16ビットアドレス)は、XBee-ZB モジュールから受信した最新のフレームデータを元に内部でアドレスキャッシュテーブルとして保存している内容を表示します。マスター登録直後や abs_agent 起動直後はキャッシュテーブルが空のため、〈net_addr〉は "FFFE" と表示されます。

abs_agent のアドレスキャッシュテーブルによって、リモート AT コマンド実行やパケット送信を行う時のアドレ

ス情報を補完して、リモートデバイス操作時のパフォーマンスを向上さています。

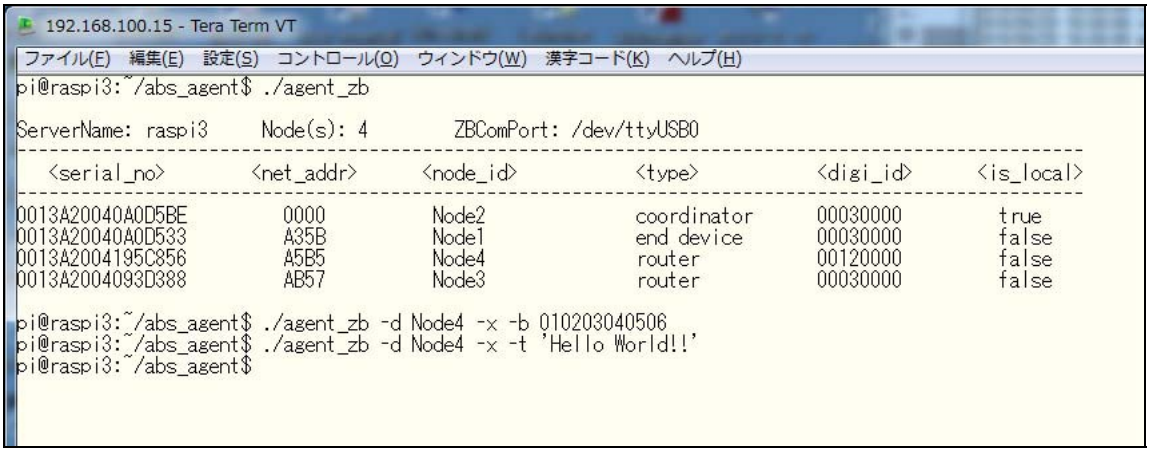
アドレスキャッシュテーブルは自動的に最新情報に更新されるため、通常はユーザーが操作する必要はありません。もし、abs_agent 内のアドレスキャッシュテーブルを全てクリアして、XBee-ZB モジュール自身の持つキャッシュとアドレス解決機能を一時的に利用したい場合には "agent_zb -Z" コマンドを実行してください。

37.5 XBee-ZB デバイス操作例

ここでは abs_agent を 2 台使用して、それぞれに XBee-ZB デバイスを接続した状態でデータの送信を行います。

前項までの設定例では XBee-ZB ("router") デバイスで Node Identification が "Node4" のものが 192.168.100.14 の IP アドレスのコンピュータで動作している abs_agent に接続しています。PAN 内のマスター一覧で表示されている Node2 にも同様にセットアップされた abs_agent (IPアドレス 192.168.100.15)が接続されています。ちなみに 192.168.100.15 の abs_agent では Node2 がローカル XBee-ZB デバイスになります。また、Node2 の XBee-ZB デバイスは "coordinator" として設定されています。

ここでは、192.168.100.15 のコンソールから、Node4 に対してデータを送信してみます。



(agent_zb コマンドを使用してデータパケット送信)

このとき、受信側の abs_agent のイベントハンドラ SERIAL_ZB が起動されて送信されたデータフレーム全体または、データフレーム中のペイロードを文字列に変換したものをログに出力します。(デフォルト動作)

(XBee-ZB データパケット受信時のログ)

ユーザーは SERIAL_ZB イベントハンドラ中に Lua スクリプトを記述してシステムを作成できます。

これ以外にも agent_zb プログラムでは、ローカルやリモートの XBee モジュールで ATコマンドを実行して設定値

を参照したり変更することができます。詳しい動作例は "クライアントプログラム"章中の agent_zb の項を参照してください。

38 スクリプトの繰り返し実行やスケジュール実行

38.1 イベントハンドラの利用 (PERIODIC_TIMER, TICK_TIMER)

abs_agent には決められた間隔で繰り返し処理を行うためのイベントハンドラ機能が用意されています。1 分毎に自動起動される PERIODIC_TIMER イベントハンドラと、1 秒毎に起動される TICK_TIMER イベントハンドラを利用できます。

それぞれのイベントハンドラ・スクリプトファイルを編集して任意の処理を組み込むことができます。スクリプト実行時間は必ず各々のイベントハンドラが実行される間隔よりも短い時間で終了するようにしてください。

数時間や数秒単位などの間隔で処理を行いたい場合には、PERIODIC_TIMER や TICK_TIMER イベントハンドラ・スクリプト中にカウントを行うようなスクリプトを記述することで簡単に実現できます。例えば 1 時間ごとに処理を行いたい場合には PERIODIC_TIMER イベントハンドラ中に下記の様なスクリプトを記述します。

CLEANUP_TASK スクリプトは 1 時間ごとに 1 回実行されます。また script_fork_exec() ライブラリ関数を使用して別スレッドで実行させていますので、CLEANUP_TASK スクリプトの処理時間は次にコールされるまでの 1 時間以内に終了すればよくなります。

```
------------------------------------------------------------
-- 1時間に一回実行する
------------------------------------------------------------
local stat,val = inc_shared_data("TIME_60M")
if (tonumber(val) >= 60) then
  set_shared_data("TIME_60M","") -- カウンタクリア
  ------------------------------------------------------------
  -- 1時間ごとに処理したいスクリプトを別スレッドでコールする
  ------------------------------------------------------------
  script_fork_exec("CLEANUP_TASK","","")
end
```

スクリプト中で使用しているグローバル変数のキー名 "TIME_60M" は別の名前を使用してもかまいません。

数秒間隔で処理したい場合には、同様のスクリプトを TICK_TIMER イベントハンドラ中に記述してください。

38.2 バックグランドで短い間隔の繰り返し処理を行う

センサーからデータを取得してサーバーや MQTT ブローカに送信を繰り返す場合など、実行間隔が 数十ミリ秒～数

百ミリ秒の場合にはバックグランドで動作するタスクとして実現します。スクリプト中で無限ループを作成してその中に wait_time() を入れて繰り返し間隔を調整します。

下記は、Raspberry Pi の SPI#0 ポートに接続したシフトレジスタ IC (74HC595) に、１バイトのデータを約 50ms 間隔で 繰り返し送信するスクリプト "TEST/RASPI_HC595_COUNTER" です。

```
file_id = "RASPI_HC595_COUNTER"
-- ２重起動防止用チェック
if not exclusive_check(g_script) then
  log_msg("*ERROR* exclusive_check() failed. script = " .. g_script,file_id)
  return
end
log_msg("start..",file_id)
-- SPI 設定
if not raspi_spi_config(true,"mode0",150,"cs0","low") then error() end
while true do
  wait_time(50)
  stat,val = inc_shared_data("HC595_COUNTER")
  local cntr = tonumber(val)
  if cntr > 255 then
    cntr = 0
    set_shared_data("HC595_COUNTER","")
  end
  if not raspi_spi_write(bit_tohex(cntr,2)) then break end
end
```

このスクリプトを実行すると、Raspberry Pi の SPI#0 ポートに接続した 74HC595 IC に、0 から 255 の値を順に１バイトデータとして送信します。74HC595 のラッチ出力に LED を接続するとバイナリカウンタが表示されます。

スクリプトの先頭部分では２重起動防止チェックのための exclusive_check() ライブラリ関数を使用しています。これはバックグランドで常駐(無限ループで終了しない) タスクとして起動させるので、２重に起動すると不具合が発生する為です。

このスクリプト実行は終了しないので起動時には別スレッドで実行させます。手動で起動するときは下記の様にします。このとき、agent_script コマンドに "-t" オプションを忘れずに指定してください。

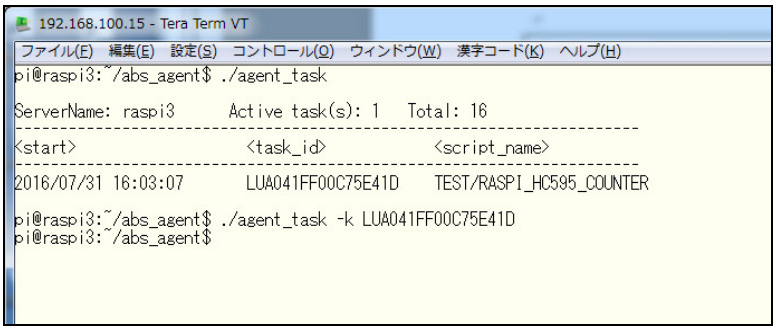
```
./agent_script -s TEST/RASPI_HC595_COUNTER -t
```

スクリプトはバックグランドで実行され続けて、約 50ms 間隔でバイナリカウンタが表示されます。通常この様にバ

ックグランドでタスクを常駐させる場合には、サーバー起動時から実行させますので　SERVER_START イベントハンドラ中にスクリプト起動を記述します。

```
-- HC595 カウンタスレッド起動
script_fork_exec("TEST/RASPI_HC595_COUNTER","","")
```

このスクリプトは abs_agent が停止するまで自動では終了しないので、途中で終了させたい場合には agent_task プログラムを使用します。最初に現在実行中のタスク一覧を agent_task コマンドで表示させます。タスク一覧から、停止させたいタスクの task_id 文字列を確認します。次に、"agent_task -k <task_id>" の様に "-k" パラメータで停止させたいタスクの task_id を指定して実行すると、実行中のタスクを強制終了できます。下記はバックグランドで実行中のタスク "TEST/RASPI_HC595_COUNTER" を終了させたときの様子です。



38.3　決められた日時にスクリプトをスケジュール実行 (crontab)

abs_agent のスクリプトを指定した日付・時刻に実行したい場合や、決められた間隔で定期的にスクリプトを実行したい場合には Linux OS の crontab 機能を利用することで実現できます。

abs_agent 機能の PERIODIC_TIMER や TICK_TIMER イベントハンドラでも繰り返しジョブを実現できますが、決められた日時にスクリプトを実行したり、曜日や時間をスケジュール実行する機能はありません。(ただし、ユーザーが PERIODIC_TIMER イベントハンドラ中に日時や曜日を判断するスクリプトを記述すれば可能です)

Linux の crontab を操作するためには、コンソールからスケジュール設定用コマンド “crontab” を使用します。crontab で指定する実行プログラムは abs_agent のクライアントプログラム agent_script を使用します。agent_script の実行パラメータに、任意のスクリプトやリクエストパラメータを指定することができます。下記の設定例では、毎時 0, 30分毎に SAMPLE スクリプトを自動実行させています。また、１つのスクリプトパラメータを実行時に指定しています。

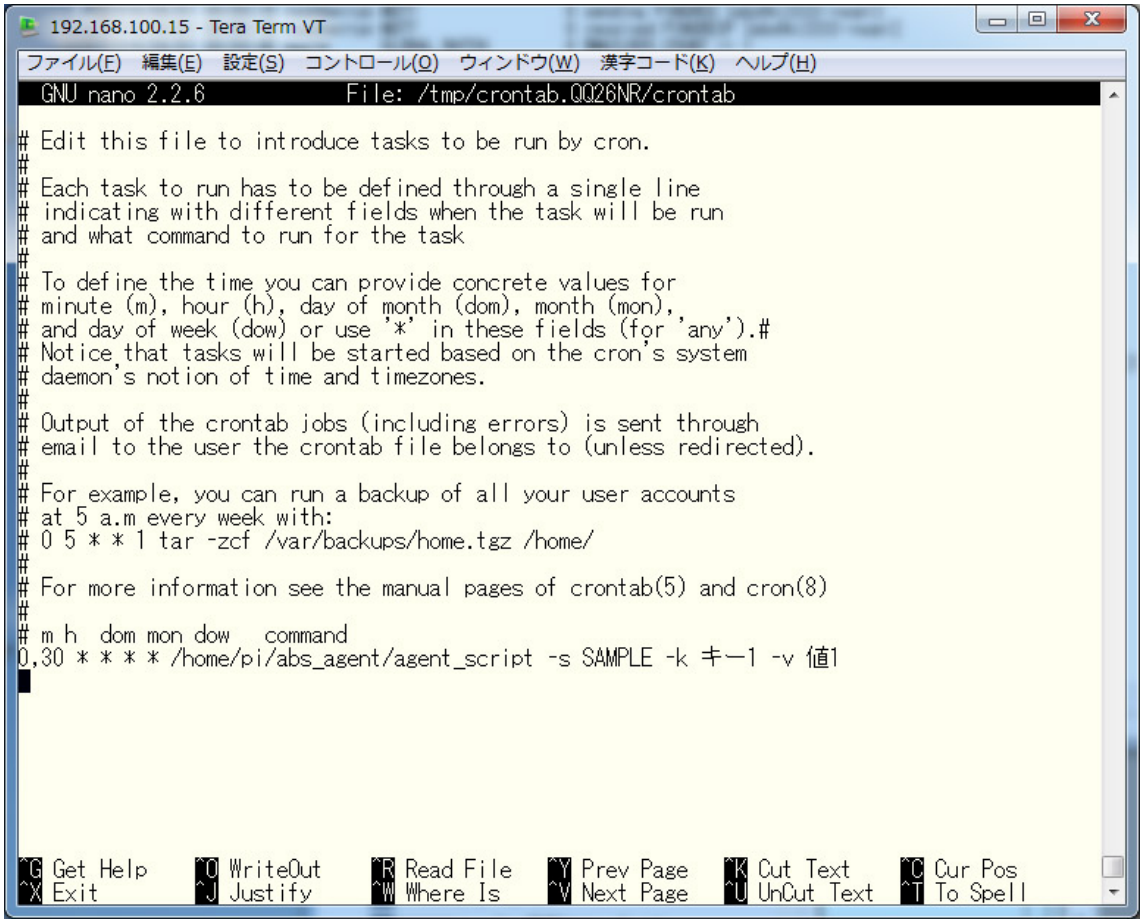
"crontab -e" コマンドでユーザー毎に保存されたスケジュールを編集します。agent_script コマンドを含む全ての abs_agent クライアントプログラムは、どのユーザーアカウントでも実行可能です。このため、ユーザー毎に別のスクリプト実行をスケジュールすることも出来ます。

```
crontab -e
```

デフォルトのエディタが起動して crontab 編集画面になります。ここにスケジュールを追加指定します。実行プログラムの agent_script はインストールしたディレクトリの絶対パス名を使用してください。追加指定するスケジュールは以下の様になります。"-s <script>" で実行したいスクリプト名を指定します。"-k <key> -v <val>" で実行時に指定したいリクエストパラメータを指定していますが省略可能です。

```
0,30 * * * * /home/pi/abs_agent/agent_script -s SAMPLE -k キー1 -v 値1
```

以下に編集中の様子を示します。



ここでは crontab へのタスク登録は abs_agent が動作しているコンピュータで行っていますが、agent_script コマンドで "-r <remote_host>" パラメータを指定して別コンピュータで動作している abs_agent のスクリプトを実行させることもできます。例えばスケジュール指定を下記の様にすると、192.168.100.14 の IP アドレスで動作している abs_agent 上の SAMPLE スクリプトを毎分起動するようになります。

```
* * * * * /home/pi/abs_agent/agent_script -s SAMPLE -r 192.168.100.14
```

38.4 スクリプト自動起動(異常終了時)

abs_agent で無限ループ構造のスクリプトを作成して別スレッドで実行させると、ポーリング方式でセンサーデータ

を取得して処理を行う機能等を実現できます。これらのスクリプトを自動起動させるためには、前項で説明した様にSERVER_START イベントハンドラ中からこれらのスクリプトを別スレッドで起動させます。

```
-- センサーデータ取得タスク起動
script_fork_exec("GET_SENSOR_DATA","","")
```

通常はこれだけで実現することができますが、スクリプト実行中に予期しないエラーが発生したり、そのエラー処理で失敗したり、ハードウエアエラーなどの理由でスクリプトが異常終了する場合が考えられます。

このときに、同じスクリプトを単に再起動させてとりあえず動作を継続したい場合(Respawn) には下記の様にして実現できます。

abs_agent で 1 秒毎に実行される TICK_TIMER イベントハンドラ中に下記の様なスクリプトを記述すると、10秒ごとに指定されたスクリプトが実行中であるかをチェックして、もしスクリプトが実行中でなければ別スレッドで起動させます。

```
file_id = "TICK_TIMER"


------------------------------------------------------------
-- 10秒に一回実行する
------------------------------------------------------------
stat,val = inc_shared_data("TIME_10S")
if not stat then error() end
if (tonumber(val) >= 10) then
    if not set_shared_data("TIME_10S","") then error() end -- counter clear
    -- チェック対象のスクリプト名を格納したテーブル
    local respawn_tasks = {}
    table.insert(respawn_tasks,"RASPI/ENVSENSOR_ALARM_TASK")
    table.insert(respawn_tasks,"RASPI/ENVSENSOR_DISPLAY_TASK")
    -- 現在実行中のタスクリストにチェック対象のスクリプトが存在しない場合には起動させる
    local stat,idlist,namelist = script_task_list()
    if not stat then error() end
    for k,v in ipairs(respawn_tasks) do
        local found = false
        for ck,cv in ipairs(namelist) do
            if cv == v then found = true end
        end
        if not found then
            log_msg("*WARNING* " .. v .. " script is not running, re-start the script..","RESPAWN_CHECK")
```

```
            script_fork_exec(v,"","")
        end
    end
end
```

(TICK_TIMER イベントハンドラ中にスクリプトの Respawn 機能を記述した様子)

上記の例では "RASPI/ENVSENSOR_ALARM_TASK" と "RASPI/ENVSENSOR_DISPLAY_TASK" の２つのスクリプトが実行中であるかを 10秒毎にチェックして、実行されていない場合には再起動させます。

実際に使用する場合にはスクリプト中に記述されている、チェック対象のスクリプト名配列やチェック間隔の秒数などを適宜変更してから使用してください。

39 ユーザーライブラリ関数・定数作成 (preload 機能)

39.1 独自のライブラリ関数や変数(定数)の作成

abs_agent で参照しているスクリプトフォルダ内の "scripts/preload" フォルダにスクリプトファイルを格納しておくと、サービスモジュール起動前にこれらのスクリプトを予め実行することができます。（以下、preload 機能）
preload 機能 は、ユーザースクリプトやイベントハンドラ中で下記の様なことをしたいときに利用できます。

- 自分で作成したライブラリ関数やソースが公開されているライブラリ関数を使用したい
- 予め値が設定された Lua グローバル変数を作成したい(ホスト名やポート番号、各種定数、マスターテーブル、フォントテーブル、変換テーブル、環境設定情報など)
- Lua スクリプトファイルのコール部分をライブラリ関数コールに代えて実行速度を上げる

abs_agent で提供しているライブラリ関数の一部でも、preload 機能を使用してライブラリ関数を定義しています。このフォルダにはユーザーが独自に作成した Lua スクリプトファイルを自由に格納することができます。

"scripts/preload" フォルダとそのサブフォルダ内の Lua スクリプトファイルは、フォルダ名とファイル名を昇順で検索して、先に見つけたものから順に実行していきます。サブフォルダを作成した場合にはその中に格納されたフォルダやスクリプトを優先的に昇順で検索します。

preload 機能で実行対象となるスクリプトファイルはファイルの拡張子が ".lua" のものだけで、その他のファイルは無視されます。また、"agent_copycfg" クライアント・プログラムの作業用ファイルとして利用される、ファイル名先頭が "_app_", "_cpy_" で始まるファイルも preload 機能ではスキップされます。

"scripts/preload" フォルダ内にサブフォルダを作成するときに、アンダースコア文字 '_' で始まるフォルダ名を使用すると、そのフォルダ内にあるスクリプトやサブフォルダを preload 検索対象から除外します。一時的にライブラリを abs_agent から取り除きたい場合に利用できます。

preload フォルダに格納するスクリプト内でライブラリ関数をコールする場合には、コールされる側のライブラリ関数は予め定義済みである必要があります。Lua 標準ライブラリで定義された関数や、abs_agent で独自に提供しているライブラリ関数はほぼ全て使用可能です。ただし、一部の abs_agent ライブラリ関数については preload フォルダ中のスクリプトで定義しているものがあります。

preload フォルダ中に配置したスクリプト中から、同じく preload フォルダ中で定義されたライブラリ関数をコールする場合には、コールされるライブラリ関数の定義を行っているスクリプトを先に実行させておく必要があります。このため、preload フォルダにスクリプトを配置する場合には、スクリプトファイルを格納するサブディレクトリを作成してください。そのサブディレクトリの名前の先頭には数字を使用して、既存のサブディレクトリ名に使われているよりも大きな数字をアサインすることで、既存のスクリプトよりも後にロード（実行）させることができます。

39.2 preload フォルダに格納するスクリプト作成時の留意事項

preload 機能を使用する場合には下記の点に留意してください

(1) Lua の基本機能のみが実行できます

abs_agent の全モジュールが有効になる前に実行されますので、関数定義や変数の代入など Lua 言語で提供される基本機能のみを使用(実行) するようにして下さい。もちろんユーザー関数を定義するときに、そのユーザー関数内では abs_agent で使用可能な全てのライブラリ関数の呼び出し文を記述できます。あくまでも "preload" フォルダ内のスクリプトをコンパイル＆実行 して Lua インスタンス内にロードする時のみの制限です。

たとえば、"scripts/preload/MY_LIB.lua" ファイル中には下記の様な記述を行うことはできません。

```
------------------------------------------------
-- Raspberry Pi GPIO#18 を High に設定 --
------------------------------------------------
raspi_gpio_write(18,true)
```

この場合には下記の様に記述します。

```
------------------------------------------------------------------
-- Raspberry Pi GPIO#18 を High に設定する関数を定義 --
------------------------------------------------------------------
function LED_on()
  raspi_gpio_write(18,true)
end
```

preload 機能によってこのスクリプトが実行されると LED_on() 関数が定義されて、全てのイベントハンドラやユーザースクリプトから LED_on() 関数をコールできるようになります。

サーバー起動時に単にスクリプトを実行したい場合には、preload 機能の代わりに "SERVER_START" イベントを利用することができます。詳しくは "SERVER_START" イベントハンドラの項を参照して下さい。

(2) 変数(定数) を設定するときは "local" 指定をつけない

たとえば、"scripts/preload/MY_LIB.lua" ファイル中に変数 my_remote_host 変数を定義するときの下記の様な記述は誤りです。

```
  ----------------------------------------
  -- 変数(定数) my_remote_host を設定  --
  ----------------------------------------
 local my_remote_host = "192.168.100.99"
```

このときは、下記の様に記述します。

```
  ----------------------------------------
  -- 変数(定数) my_remote_host を設定  --
  ----------------------------------------
 my_remote_host = "192.168.100.99"
```

このスクリプトでは my_remote_host 変数が定義されますので、全てのイベントハンドラやユーザースクリプトからこの変数を利用できます。たとえば下記の様な使い方ができます。

```
  ----------------------------------------
  -- 変数(定数) my_remote_host を利用  --
  ----------------------------------------
 local stat,val = get_net_data("Key1", my_remote_host)
```

(3) preload 機能で作成した変数や関数の上書きは無効です

"preload"フォルダに格納したスクリプトは、abs_agent内のデフォルトで16個ある全スクリプトプールエントリ(Luaスクリプト実行インスタンス)を対象に実行されます。このため、ユーザースクリプトやイベントハンドラ中から preload 機能で作成された変数を更新した場合には、１つのスクリプトエントリのみを更新したことになります。その後、別のスクリプトエントリがシステムで自動アサインされると、意図しない値が変数に格納されていることになります。

このような事を防ぐために、preload 機能で設定する内容を変更する場合には、必ず "preload" フォルダ内のスクリプト自身を変更して、その後 abs_agent を再起動するようにしてください。

40 APPENDIX(A) Webアプリケーション

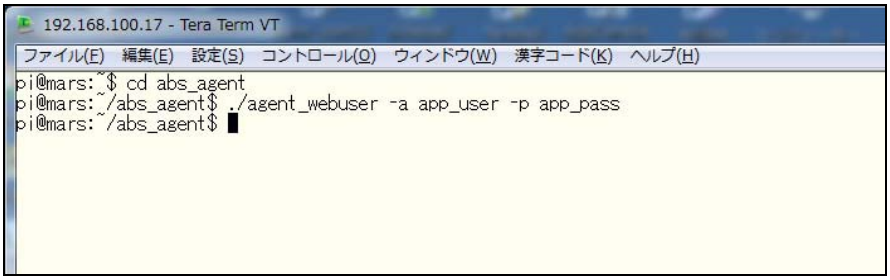
abs_agent のインストールキットにはユーザーが直ぐに利用可能な Web アプリケーションが同梱されています。これらの Web アプリケーションを利用すると、abs_agent 内で登録したセンサーデータを集計してグラフ表示を行う事ができます。Windows やスマートフォン、タブレットから Web ブラウザを使用して、abs_agent で公開している HTTP サーバーにアクセスすると、何時でもこれらの Web アプリケーションを実行できます。

インストールキットに含まれるデフォルトの Web アプリケーションは下記になります。

Web アプリケーション名	機能概要
FASTDB 集計グラフ	FASTDB に登録した任意のセンサーデータ等を、指定した期間で集計してグラフを作成する。
FASTDB RawDataチャート	FASTDB に登録した任意のセンサーデータの値を時系列にプロットする。 集計グラフとは違って加工前の、登録した値をそのまま表示したい場合に使用します。

これらの Web アプリケーションの詳細な説明は以降の項で行います。また、上記のWeb アプリケーションのソースファイル一式(javascript, HTML, CSS ファイル等) は abs_agent をインストールした "webroot" 以下に格納されています。これらのソースファイルを元にユーザーが独自の Web アプリケーションを作成することもできます。
このマニュアル中の Webアプリケーション画面は、最新版のインストールキットに同梱されているものとは一部異なる場合があります。これらの違いはボタンや選択項目が追加されている程度ですので、操作時は実際の画面を参照してください。

Web アプリケーションを利用する場合には abs_agent の HTTP サーバー機能を有効にしておく必要があります。インストール直後のデフォルト設定では、ポート番号 8080 で HTTP サーバーが動作しています。また、Webアプリケーションでユーザー認証機能を使用する場合には、予め abs_agent に Web ユーザーアカウントを登録しておきます。agent_webuser クライアントプログラムを使用して Web ユーザーアカウントを登録する場合には下記の様にコマンドを実行します。



(ユーザー名 "app_user" 、パスワード "app_pass" で新規 Webユーザーアカウントを作成)

この章で紹介する Web アプリケーションは abs_agent 自身の HTTP サーバーで配信されています。Webアプリケーションでは JavaScript から abs_agent 側の Web API をコールして各種機能を実現しています。

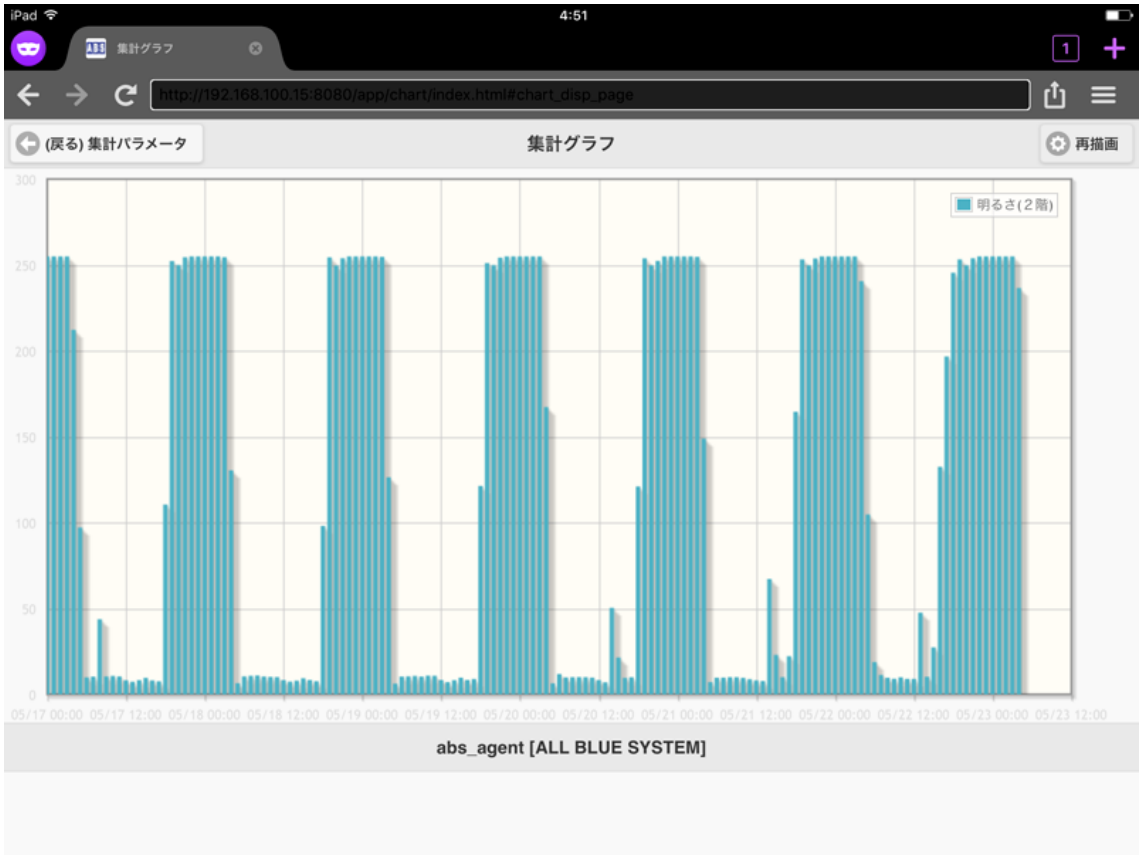
Web アプリケーションを配信するサーバー機能と、abs_agent 側の Web API 機能は別々のコンピュータに設置する

こともできます。この場合には、Web アプリケーション内からコールする Web API のパス部分を一部変更する必要があります。詳しくはWeb アプリケーション・ソースファイル中のコメントを参照してください。

40.1 FASTDB 集計グラフ作成 Webアプリケーション

abs_agent の集計用データベース(FASTDB) に格納したデータを集計してグラフ表示を行うための Web アプリケーションです。abs_agent のスクリプトやイベントハンドラ中から、FASTDB に任意のキー名を付けたデータを登録しておくと、この Webアプリケーションで集計グラフを作成することができます。

下記は、照度センサーの計測データを定期的に FASTDB に保存して、これを iPad のWeb ブラウザから abs_agentのWebアプリケーションを実行した様子です。



(FASTDB 集計グラフ作成 Webアプリケーションの実行例)

abs_agent インストール直後には、コンピュータのフリーメモリサイズとフリーディスク容量を定期的に計測してFASTDB に登録するスクリプトが実行されています。PERIODIC_TIMER イベントハンドラは 1 分毎に自動起動されていて、このスクリプト中の下記のスクリプト部分によってコンピュータのリソース情報が定期的に FASTDB に格納されています。

```
------------------------------------------------------------
-- OS のステータス情報を取得して FASTDB に格納(5分に一回実行)
------------------------------------------------------------
```

```
local timer = "STORE_OS_STATUS_TIMER"
local val = stat_check(inc_shared_data(timer))
if (tonumber(val) >= 5) then
  stat_check(set_shared_data(timer,""))
  local CPUSTAT = stat_check(script_exec2("OS/GET_STATUS","",""))
  --if not fastdb_add("CPULoad",tonumber(CPUSTAT["CPULoad"])) then error() end
  if not fastdb_add("FreeMemMB",tonumber(CPUSTAT["FreeMemMB"])) then error() end
  if not fastdb_add("FreeDiskGB",tonumber(CPUSTAT["FreeDiskGB"])) then error() end
end
```

(PERIODIC_TIMERイベントハンドラ中に記述された CPU リソースを5分毎にFASTDB に保存する部分)

FASTDB に格納されたデータは、デフォルト設定では10 日間を経過すると自動で削除されます。もし集計対象期間をそれよりも長くしたい場合には、自動削除の日数を延ばす必要があります。詳しくは、"時系列集計用インメモリデータベース API"の項をご覧ください。

上記のスクリプトからコールされる、CPU リソース情報を取得するスクリプト (OS/GET_STATUS) は以下の様になっています。

GET_STATUS.lua

```lua
--[[
●機能概要
abs_agent が動作している OS ステータスを取得する

●リクエストパラメータ
無し

●リターンパラメータ
------------------------------------------------------------------------------------
キー値            値                                                        値の例
------------------------------------------------------------------------------------
CPULoad         CPU ロード平均値                  0.01
FreeMemMB       フリーメモリサイズ(MBytes)        211
FreeDiskGB      root ファイルシステムのフリーディスクサイズ(GBytes) 12

●備考

●変更履歴
2017/06/10  初版作成
]]

-- CPULoad
local top_val = ""
local cmd = [[top -bn1 | grep load | awk '{printf "%.2f",$(NF-2)}']]
local f = assert(io.popen (cmd))
for line in f:lines() do
    top_val = line
end
f:close()
script_result(g_taskid,"CPULoad",top_val)

-- FreeMem
local free_mem = ""
cmd = [[free -m | awk 'NR==2{printf "%g",$4}']]
f = assert(io.popen (cmd))
for line in f:lines() do
    free_mem = line
end
f:close()
script_result(g_taskid,"FreeMemMB",free_mem)

-- FreeDisk
local free_disk = ""
cmd = [[df -h | awk '$NF=="/"{printf "%g", $4}']]
f = assert(io.popen (cmd))
for line in f:lines() do
    free_disk = line
end
f:close()
script_result(g_taskid,"FreeDiskGB",free_disk)
```

Lua source File　length : 1320　lines : 66　Ln : 1　Col : 1　Sel : 0　UNIX　UTF-8N　INS

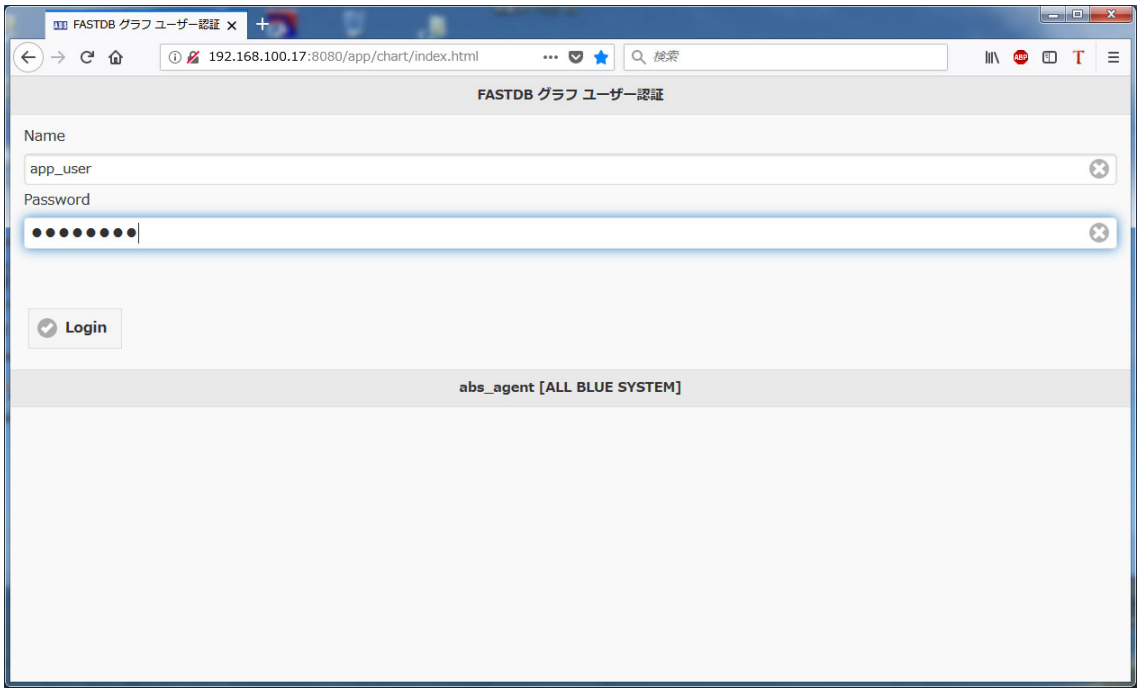
(abs_agent が動作しているコンピュータのリソース情報を取得するスクリプト)

これらのスクリプトはインストール直後から有効になっていますので、FASTDB を集計することで過去の計測値を簡単に確認することができます。

集計用 Web アプリケーションを実行するには、PC やタブレットの Webブラウザから下記の URL にアクセスしてください。

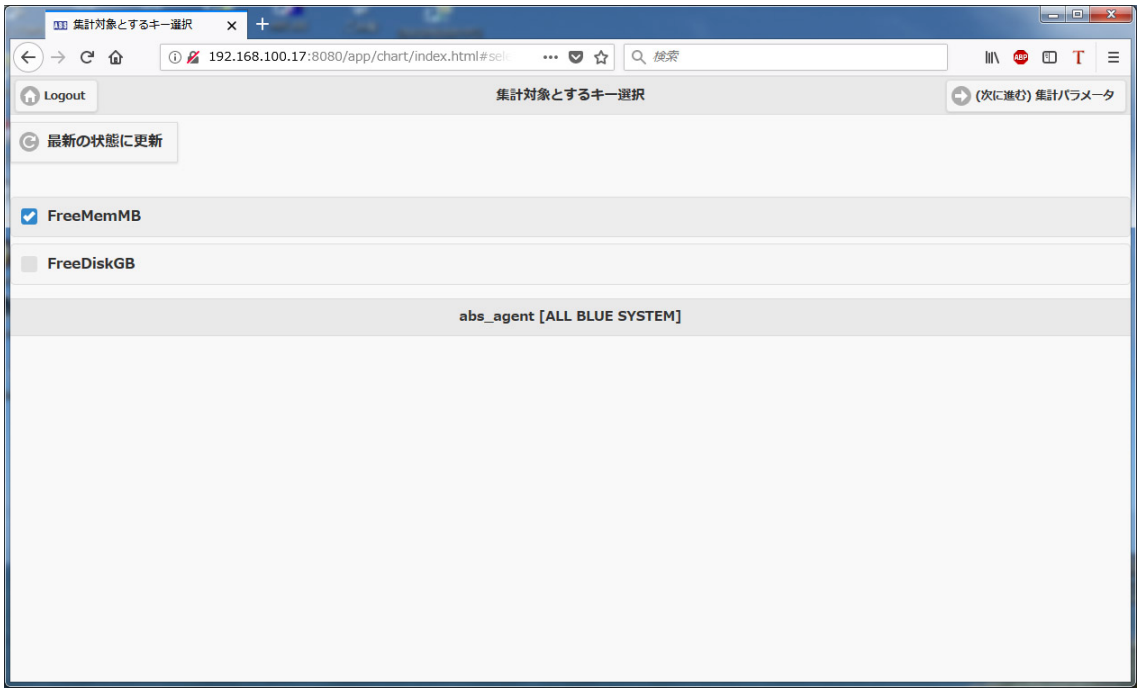
Webアプリケーションを実行する URL (HTTP サーバーのポートが 8080の場合)
http://<abs_agentが動作しているコンピュータの IPアドレス>:8080/app/summary/index.html

最初に Web アカウントのログイン認証画面が表示されます。ここで、先に登録した Webユーザー名 ″app_user″ 、パスワード ″app_pass″ を入力します。



（ログイン認証画面）

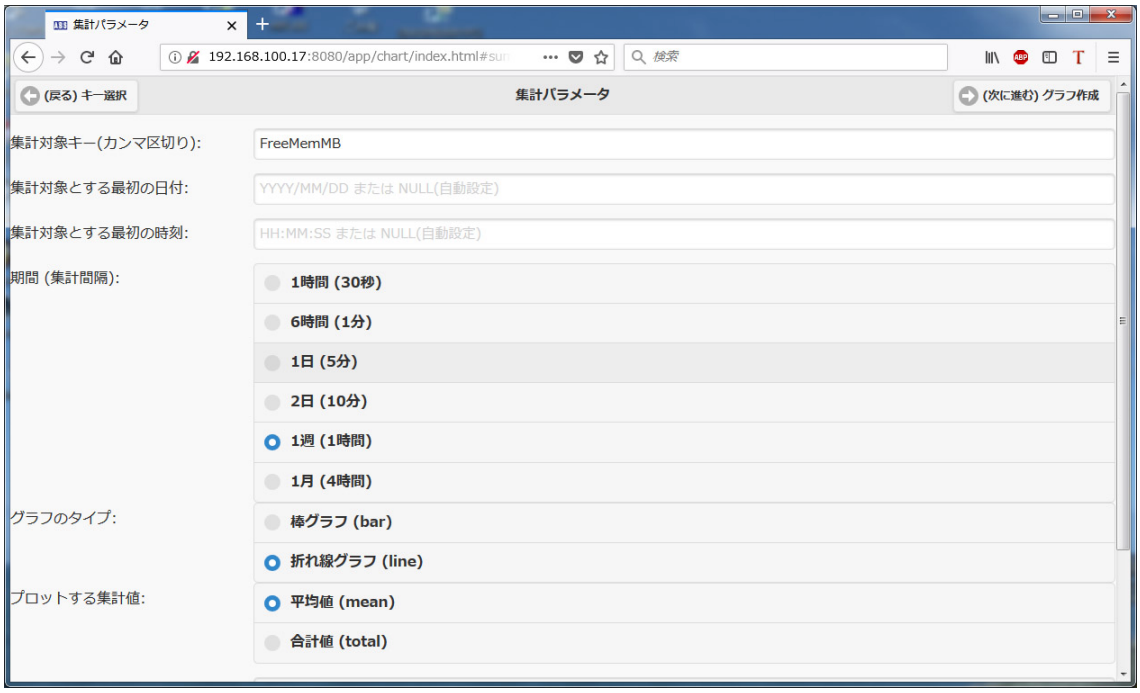
ログインに成功すると、現在 FASTDB に登録されているデータのキー名一覧が表示されます。



(FASTDB のキー名一覧と集計対象選択画面)

ここで、集計を行いたいデータ列を示すキーをチェックボックスで選択します。複数のキーを選択すると同じグラフ中に複数のデータ列のグラフを描画します。

"集計パラメータ" ボタンを押すと集計方法を選択する画面が表示されます。



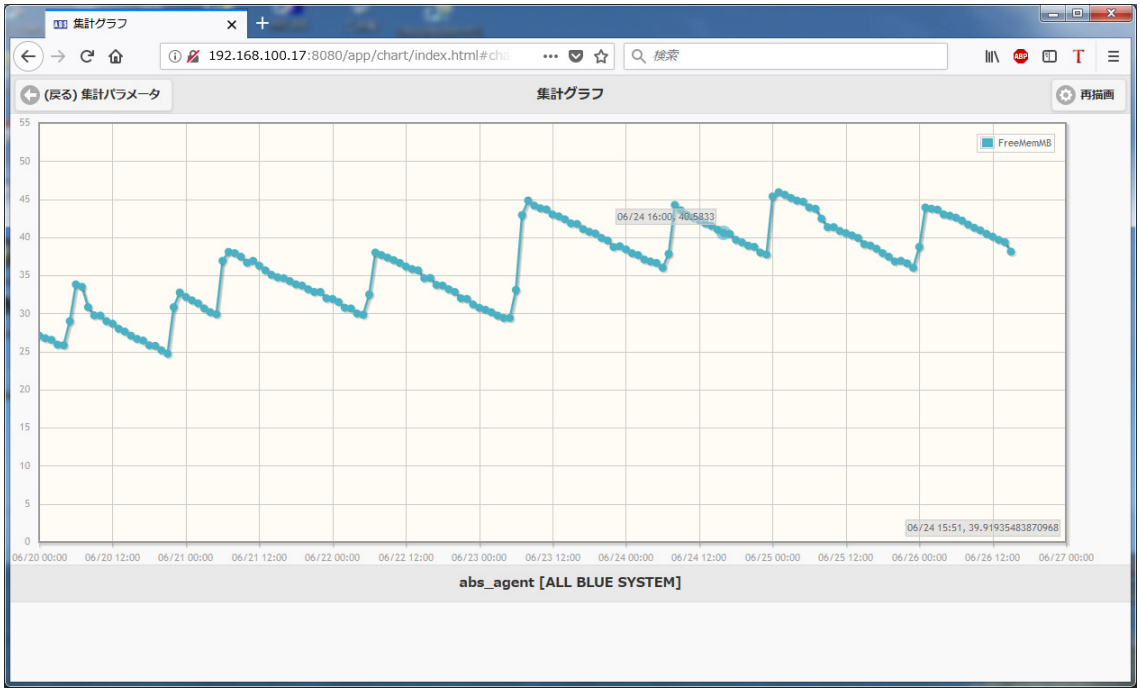
（集計方法選択画面）

この画面で指定できる集計パラメータは以下になります。

選択項目	説明
集計対象キー	この前のキー選択画面で選択したキー名がカンマ区切りで表示されています。 編集することで、集計対象キーをここで変更することができます。
集計対象の最初の日付	集計グラフに表示される最初の日付を指定します。 空白のままにしておくと、集計期間と現在日を元に自動で集計対象開始日付を計算します。
集計対象の最初の時刻	集計グラフに表示される最初の時刻を指定します。 空白のままにしておくと、集計期間と現在日・現在時刻を元に自動で集計対象開始時刻を計算します。
期間	集計対象期間を選択します。ここで選択した期間内のタイムスタンプ値をもった FASTDB データを集計します。() 内の時間は集計間隔を示します。集計間隔毎に平均値または合計値を集計して、グラフにプロットします。期間を選択するとデフォルトの集計間隔が設定されますが、この値は後の項の"集計間隔を変更"に値を設定することで別の集計間隔を指定することができます。
グラフのタイプ	棒グラフまたは折れ線グラフを選択します。
プロットする集計値	集計期間毎にプロットする値を指定します。 "平均値"は、集計間隔内に見つかった全てのデータの平均値をプロットします。 "合計値"は、集計間隔内に見つかった全てのデータの合計値をプロットします。 "累計値"は、集計対象の開始日・時刻から集計間隔毎の合計値を加算していった値をプロットします。累計値の初期値は 0 で、集計間隔内にデータが見つからなか

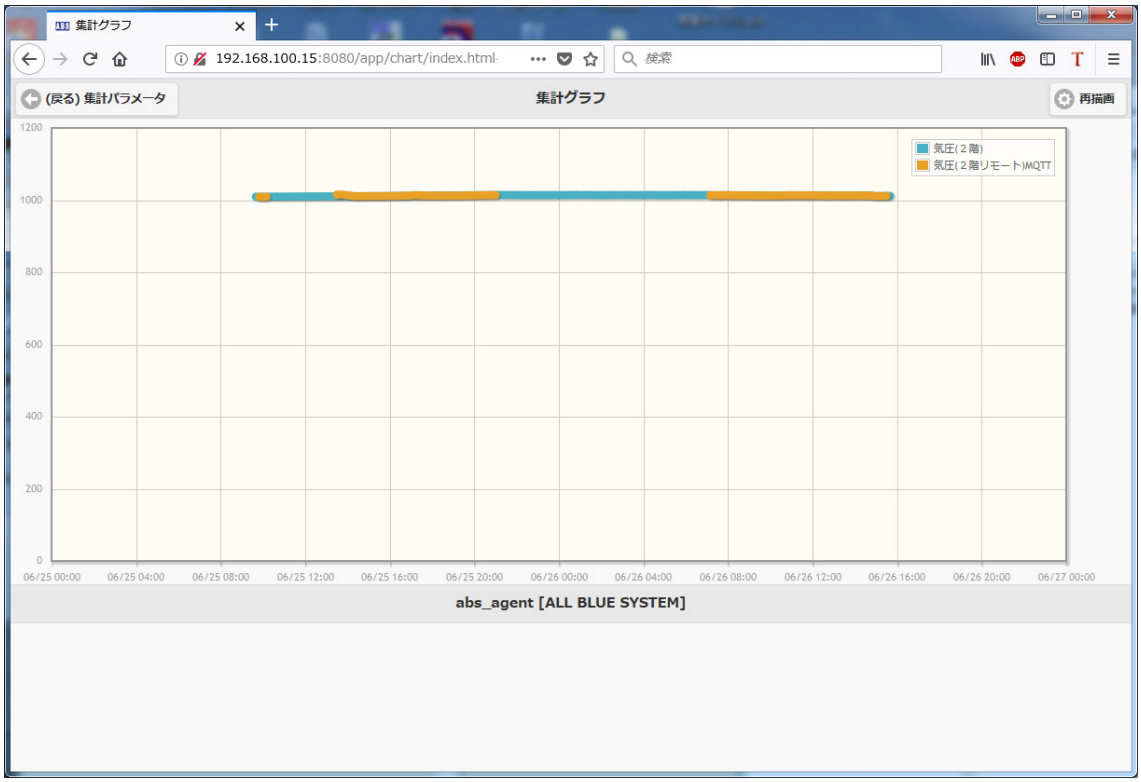
	った場合でもその期間の合計値を 0 として累計計算を行います。 "最初のデータ値"は、集計間隔内で最も古いタイムスタンプを持った１つのデータ値をそのままプロットします。
集計間隔を変更	"期間"を選択したときのデフォルト集計間隔をここで指定した値に変更します。 空白にするとデフォルト値を使用します。
データ値のスケール最小値	グラフを描画するときに、縦軸のスケールに使用する最小値を指定します。 テキストエリア右端の（x）を押すとデフォルト値の 0 を消して空白にすることができます。空白を指定すると、FASTDB の集計値をもとに自動で縦軸のスケールが決定されます。

ここでは、集計期間を１週間にして後のパラメータはデフォルト値を使用してみます。"グラフ作成" ボタンを押すとグラフ画面が表示されます。このとき、集計期間に比べて集計間隔を極端に短くしたり、集計対象キーを多く選択するとグラフ作成に時間がかかる場合があります。



（集計グラフ）

データが存在しない部分のグラフは描画されません。また、PC 上の Web ブラウザを使用している場合には、マウスでドラッグしてグラフを囲むことで、部分拡大して表示することもできます。例えば、気圧データを集計した場合にデフォルトで表示されるグラフは下記のようになります。



（気圧変化のグラフ、デフォルト表示）

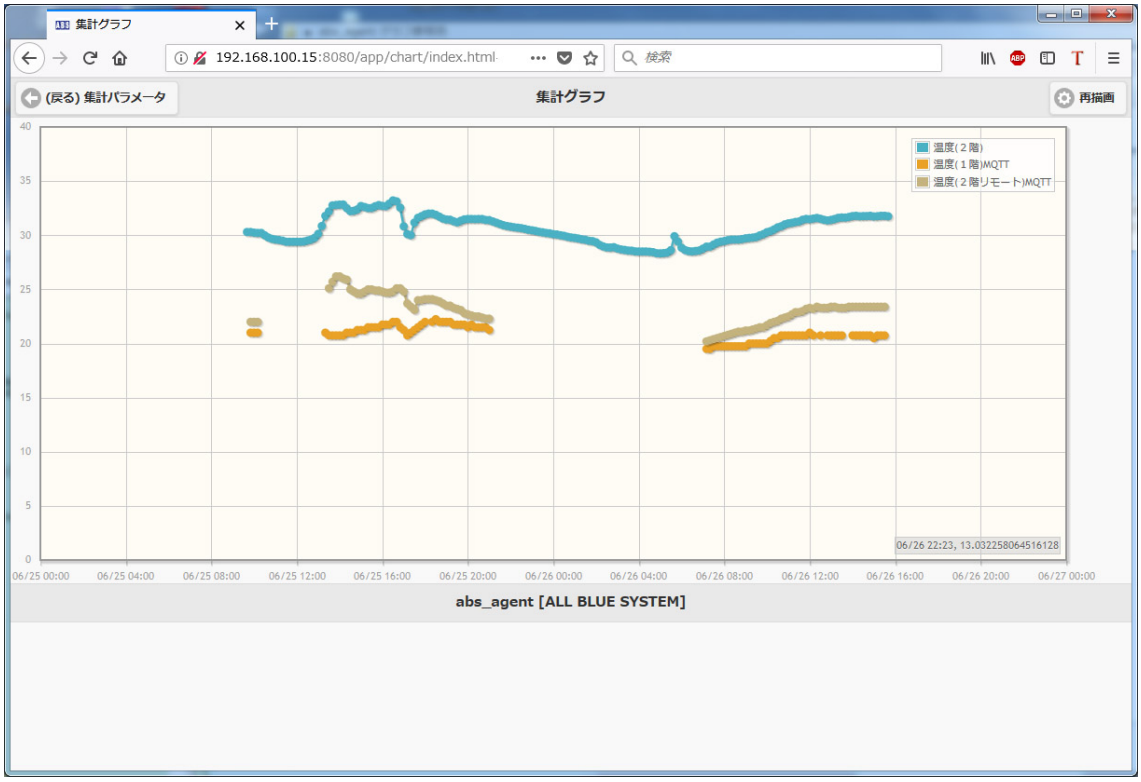
これをマウスで部分拡大して表示させると、下記の様になって気圧変化が見やすくなります。

（気圧変化のグラフ、マウスで部分拡大）

集計パラメータの〝データ値のスケール最小値〟を空白にすると、自動でスケールされて同様の結果を得ることがで

きます。

複数の集計データキーを１つのグラフに表示した場合には以下の様なグラフが得られます。

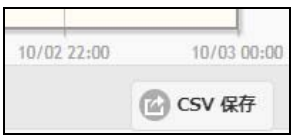


(複数のキーを選択してグラフを作成)

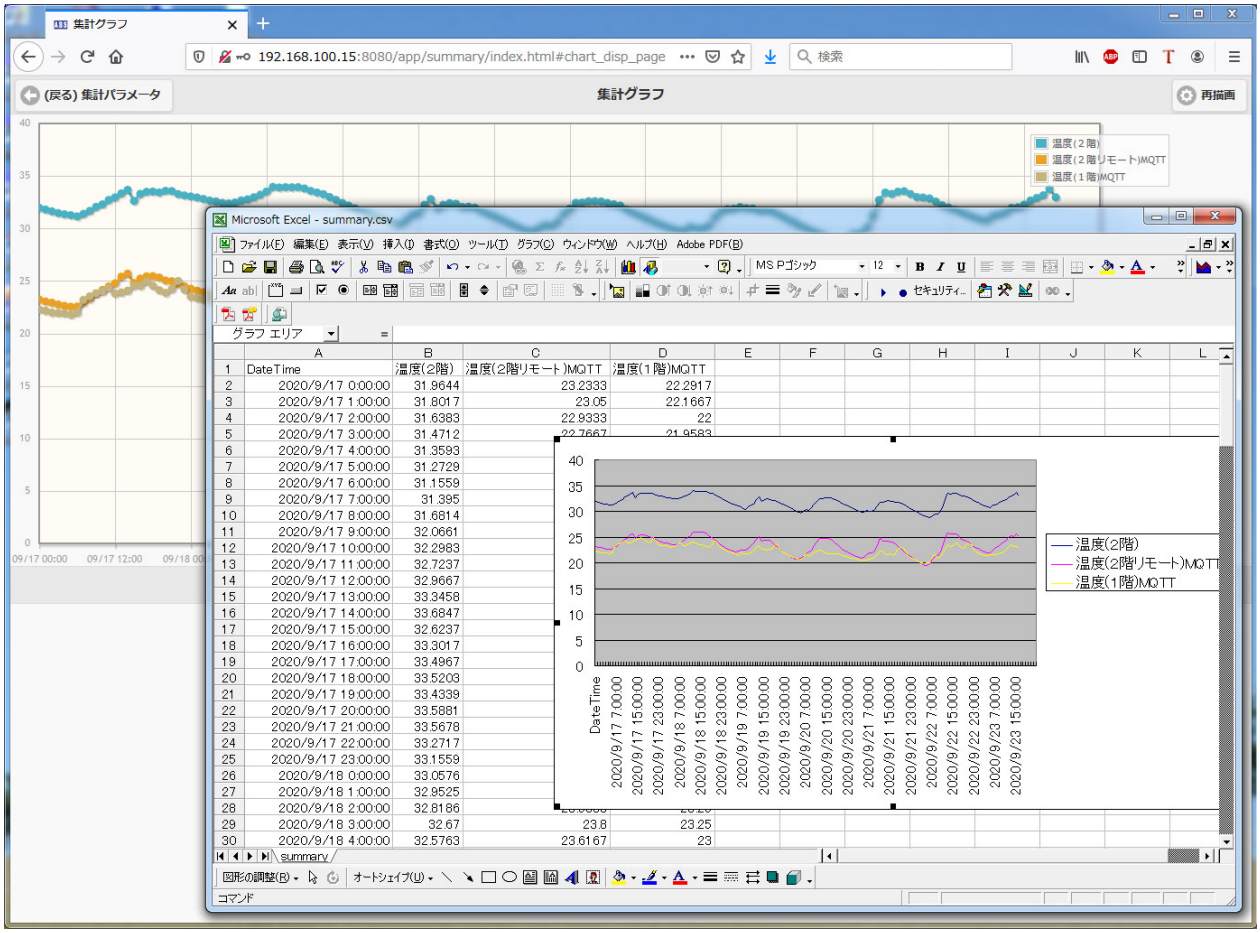
複数のキーを１つのグラフにする場合には、同じデータタイプ(温度、湿度、気圧 etc...)のものだけを選択するようにしてください。

グラフ画面で "集計パラメータ"ボタンを押すと、集計パラメータ設定画面に戻ることができます。また、"再描画"ボタンを押すと、集計計算とグラフの描画をもう一度実行します。

グラフ画面右下の "CSV保存"ボタンを押すと、集計結果を CSV データとしてブラウザからダウンロードすることができます。ダウンロードした CSV データはファイル(UTF-8 形式, BOM付)に保存できますので、表計算ソフト等から集計データを利用できます。使用する表計算ソフトによっては、サポートする文字コードへの変換作業が別途必要になる場合があります。



(グラフ画面の CSV データ保存ボタン)



(CSV データを表計算ソフトでロードしてグラフを作成した様子)

Web アプリケーションを終了する場合には、キー選択画面まで戻った後 "Logout" ボタンを押します。下記の様に確認メッセージが表示されてログアウトすることができます。

（ログアウト確認ダイアログ）

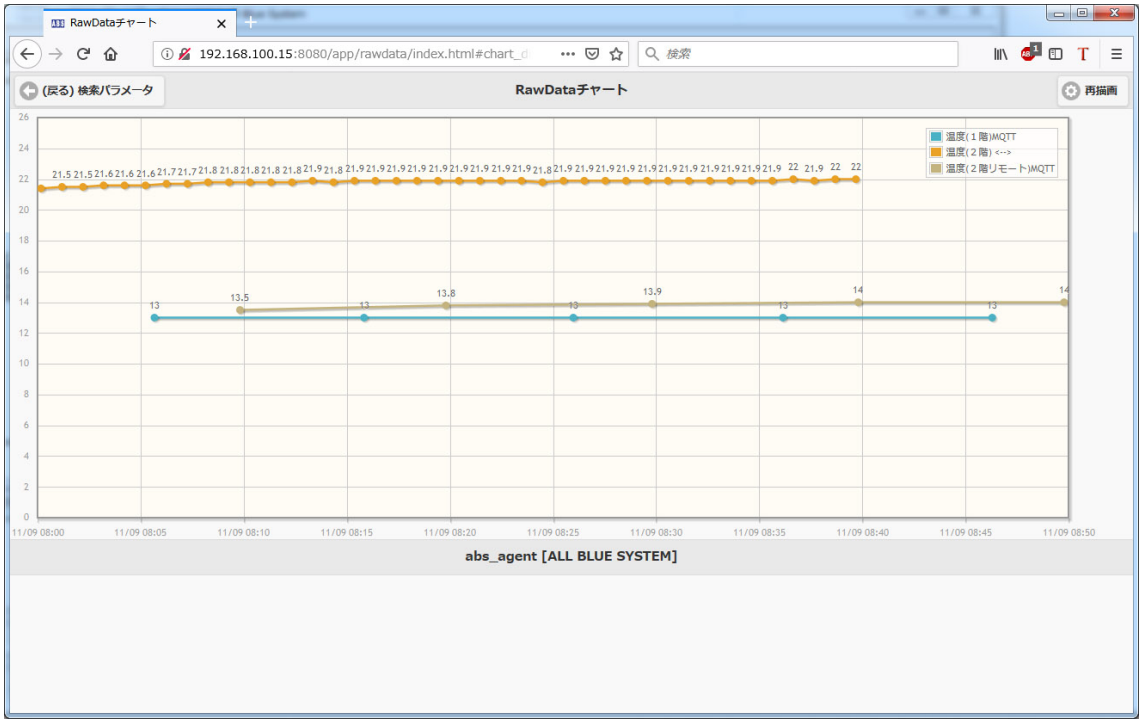
Web アプリケーション実行中に Web ブラウザを強制終了した場合でも、ログイン中のセッションは abs_agent 側で自動削除されますので問題は発生しません。

40.2 FASTDB RawData チャートWebアプリケーション

abs_agent の集計用データベース(FASTDB) に格納したデータの値を(集計等を行わないで)そのままチャートにする Web アプリケーションです。前項の"集計グラフ作成 Webアプリ"は集計間隔毎に平均値や合計値を計算して、その集計結果を集計間隔ごとにプロットしていました。これに対して "RawData チャート Webアプリ"では FASTDB に登録したデータのまま表示します。X 軸の時系列に沿ってデータの登録時刻ごとに(等間隔ではなく)プロットされますので、センサーデータが意図したタイミングで正常に登録されているかを簡単に確認することができます。

複数のデータ系列を同時に RawData チャートに表示することもできます。この場合にはそれぞれのデータ登録タイミングとデータ値の関係を簡単に把握することができます。

下記は、複数の温度センサーデータを FASTDB に保存しているときに、PC のWeb ブラウザから abs_agentの "RawDataチャートWebアプリケーション" を実行した様子です。



(FASTDB RawDataチャート Webアプリケーションの実行例)

RawDataチャート Web アプリケーションを実行するには、PC やタブレットの Webブラウザから下記の URL にアクセスしてください。

Webアプリケーションを実行する URL (HTTP サーバーのポートが 8080の場合)
http://<abs_agentが動作しているコンピュータの IPアドレス>:8080/app/rawdata/index.html

最初に Web アカウントのログイン認証画面が表示されます。ここで、先に登録した Webユーザー名 "app_user" 、パスワード "app_pass" を入力します。

以降の動作画面はタブレット(iPad) から実行した例です。Windows PC 等の Web ブラウザから動作させた場合も画面や操作方法は同じです。

（ログイン認証画面）

ログインに成功すると、現在 FASTDB に登録されているデータのキー名一覧が表示されます。

(FASTDB のキー名一覧が表示され検索対象キーを選択する)

ここで、チャートに表示したいデータ列を示すキーをチェックボックスで選択します。複数のキーを選択するとチャート中に複数のデータ列のグラフを描画します。

"検索パラメータ" ボタンを押すとデータ検索条件を設定する画面が表示されます。

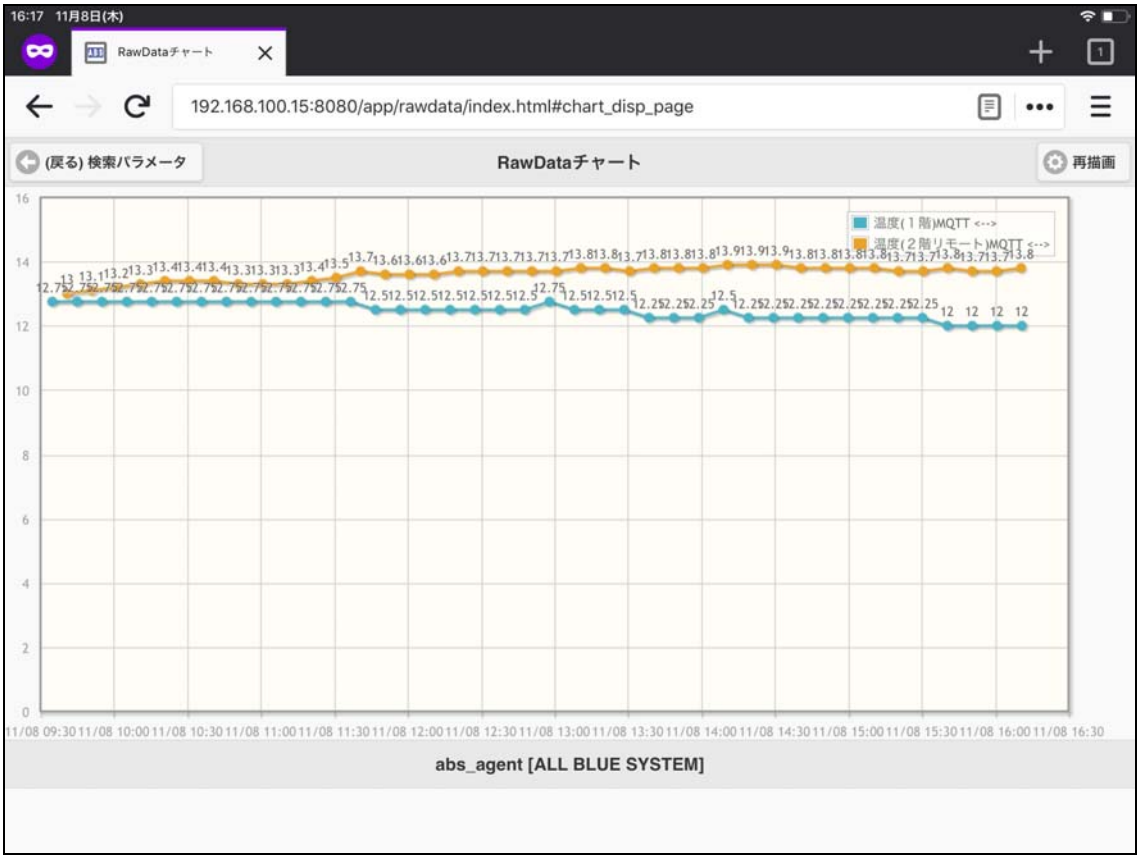
（検索パラメータ設定画面）

この画面で指定できる検索パラメータは以下になります。デフォルト値のまま何も変更しない場合には、最も最近登録された直近 40 件のデータを表示します。

選択項目	説明
検索対象キー	この前のキー選択画面で選択したキー名がカンマ区切りで表示されています。 編集することで、集計対象キーをここで変更することができます。
表示データ数(最大)	チャートに表示するデータ数の最大値。 検索条件に一致するデータがここで設定した数を超えた場合には、"表示データ数"分のみがチャートにプロットされます。このときチャート上の表示データ数が制限されたことを示すために、チャート右上に表示されるキー名に "<-->" 文字列が追加されます。 複数のキーをチャートに同時に表示している場合にも、全てのキー毎にこの表示データ数(最大)が適用されます。このため、データ登録頻度が違う複数のデータ列を

	チャートに表示している場合には、"<-->" マークが表示されているキーのデータ列は、X軸の時系列中にデータは格納されているのにも関わらず表示されていない場合があります。この場合には検索範囲を縮小するか、表示データ数を大きくしてください。
検索対象の開始日付	データを検索するときに、ここで指定した日付以降のデータを対象にします。 空白のままにしておくと登録したデータの最も古いタイムスタンプの日付または、現在日が設定されます。
検索対象の開始時刻	データを検索するときに、ここで指定した時刻以降のデータを対象にします。 空白のままにしておくと登録したデータの最も古いタイムスタンプの時刻または、0:0:0 が設定されます。
検索対象の終了日付	データを検索するときに、ここで指定した日付以前(指定した日付を含まない)のデータを対象にします。 空白のままにしておくと登録したデータの最も新しいタイムスタンプの日付を含むデータまでが対象になります。
検索対象の終了時刻	データを検索するときに、ここで指定した時刻以前(指定した時刻を含まない)のデータを対象にします。 空白のままにしておくと登録したデータの最も新しいタイムスタンプの時刻を含むデータまでが対象になります。
検索順	検索期間内のデータを検索するときに、検索開始日時側から順にデータを取得してプロットデータにする場合には "昇順" を指定します。 検索終了日時側から順にデータを取得してプロットデータにする場合には "降順" を指定します。 "自動設定"を選択すると、検索条件に応じて"昇順"または"降順"を自動で決定します。 どの検索順を選択した場合でも、チャート上のデータは常に X 軸(時系列) の左側が古い日時で、右側が新しい時刻に揃えられます。
チャート中のデータポイントに値を表示	チャートのデータをプロットするときに、データポイント上部にデータ値を表示する場合にチェックを付けます。データポイントが密集しているときなどはチェックを外すことでチャートが見やすくなります。
データ値のスケール最小値	チャートを表示するときに、縦軸のスケールに使用する最小値を指定します。 テキストエリア右端の (x) を押すとデフォルト値の 0 を消して空白にすることができます。空白を指定すると、実際のデータ値をもとに自動で縦軸のスケールが決定されます。

ここでは、２つの温度センサーのデータを表示するために、２つのキーを選択しています。当日の 9:00 以降に取得したデータを表示するために "検索対象の開始時刻"を設定しています。"チャート作成" ボタンを押すとチャート画面が表示されます。



(RawDataチャートで２つの温度センサデータを表示)

チャートを表示すると右上にキー名一覧が表示されますが、このときキー名の後に "<-->" 文字列がつく場合があります。上記の例では、２つのセンサーデータのキー名両方にこの文字列が表示されていて、検索結果で取得したデータはグラフ中に全て表示しきれていないことを示しています。

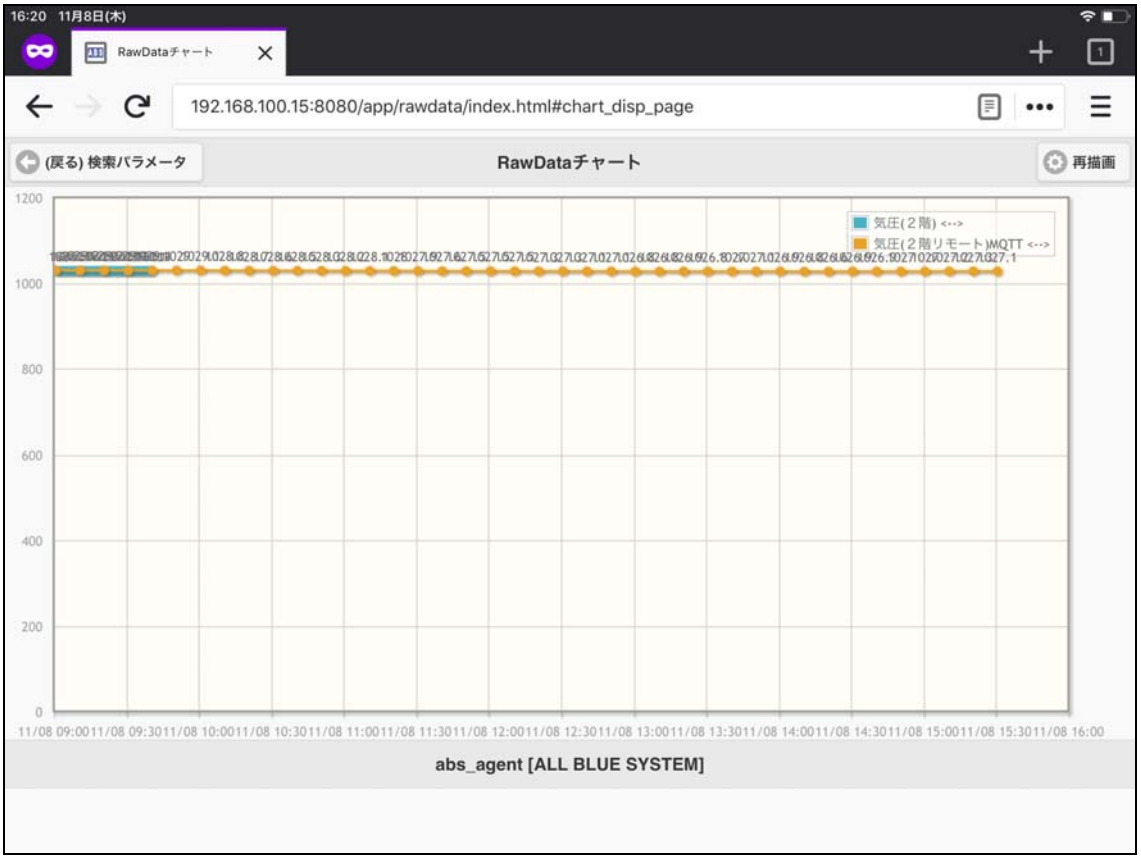
ここで、"検索パラメータ"ボタンを押して検索条件を変えてみます。

現在時刻付近をもう少し詳しく見るために、"検索対象の開始時刻"を 12:00 に変更します。また、データプロット数を 40 から 100 に増加させます。２つのデータの温度差を詳しくみるため "データ値のスケール最小値" を 0 から 8 に変更します。設定後 "チャート作成" ボタンを押してチャートを作成し直します。

（２つの温度センサのデータプロット、検索条件を変更後）

現在時刻付近の２つの温度変化が見やすくなりました。またチャート右上のキー名に ″<-->″が表示されていないので、検索条件に一致したデータは全てチャート中にプロットされていることが判ります。

次の実行例は、２つの気圧センサのデータを表示しています。



(異なるサンプリング周期の気圧センサデータを表示した様子)

ブルーとオレンジのラインが接近していてチャートが見づらくなっています。また、ブルーのプロットデータはオレンジのデータに比べてサンプリング周期が短いので、明らかにデータの途中で表示が切れています。オレンジのデータも検索結果に一致したデータは全て表示しきれていません。

ここで検索パラメータ設定画面に戻って、検索条件を変更してみます。

(検索条件を変更している様子)

最初に表示データ数を 1000 に増加させて、9:00 以降の全てのデータを表示できるようにします。また、縦軸の変化を詳しく表示できるようにデータ値のスケール最小値を消して空白(Null)にします。次に、データプロット上部の数値表示を消してグラフを見やすくします。

ここで、再度"チャート作成"ボタンを押します。

(検索条件を変えた後に、２つの気圧センサデータを表示)

これでチャートが見やすくなりました。またキー名に "<-->" が付いていないので、9:00 以降の全てのセンサデータがチャート中に表示されていることが判ります。

Web アプリケーションを終了する場合には、キー選択画面まで戻った後 "Logout" ボタンを押すと確認メッセージが表示された後ログアウトすることができます。

Web アプリケーション実行中に Web ブラウザを強制終了した場合でも、ログイン中のセッションは abs_agent 側で自動削除されますので問題は発生しません。

41 APPENDIX(B) LCD表示アプリケーション(Raspberry Pi 専用)

この章では Raspberry Pi 向けの abs_agent で動作可能な LCD 表示アプリケーションについて説明します。

Raspberry Pi 用 abs_agent インストールキットには、各種 LCD デバイス用のドライバ(Luaスクリプト)が予め同梱されていて scripts/preload/100_RASPI ディレクトリ内に格納されています。ユーザーが使用する LCD デバイスに対応したフォルダ名先頭の "_" アンダースコアを取り除いた後再起動することでこれらのデバイスへの表示が可能になります。

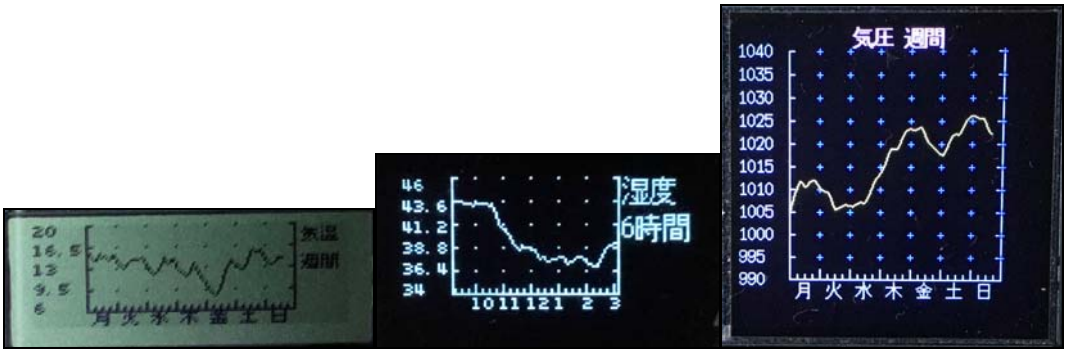
abs_agent インストールキットで提供されている LCDドライバの詳細は、"カラー(RGB)・モノクロ(Bitmap)グラフィックAPI" の章の説明を参照してください。使用したい LCD デバイス用のドライバがインストールキットで提供されていない場合でも、既存のライブラリを修正することで簡単に使用可能になります。Intel X86用の環境で abs_agent

を使用する場合でも、グラフィックライブラリが使用するディスプレイバッファ(abs_agent 内のグローバル共有メモリエリア)の内容を LCD に転送する部分をユーザー側で記述するだけで使用可能になります。

次項で説明するアプリケーション(スクリプト)を動作させるためには、abs_agent が動作するハードウエアに LCD が接続され、そのデバイスに対応したドライバがセットアップされている必要があります。詳しくは"カラー(RGB)・モノクロ(Bitmap)グラフィックAPI" の章中の説明を参照してください。

41.1 集計グラフLCD表示 (RASPI/FASTDB_LCD_CHART)

abs_agent の集計用データベース(FASTDB) に格納したデータを集計して LCDデバイスにグラフ表示を行うスクリプトです。abs_agent のスクリプトやイベントハンドラ中から、FASTDB に任意のキー名を付けたデータを登録しておくと、このスクリプトを実行するだけで集計グラフを LCDに表示することができます。FASTDB に保存されているデータであれば、データレンジや保存間隔を問わず最適な集計グラフを手元の LCD デバイスに表示することができます。



スクリプトファイル実行例。左からAQM1248ビットマップLCD,モノクロOLED, RGBグラフィックLCDへの表示例です。

グラフ表示を行うデータは、FASTDBに登録されているデータであれば直ぐに表示できます。1桁から4桁の数値であればスクリプト内で自動的にスケールが調整されて最適なグラフになるようにしています。もし、表示スケールを変更したい場合でもスクリプトパラメータを指定することで望みのグラフにすることができます。

集計グラフ作成用のスクリプトファイルは以下に格納されています。スクリプト内で全ての集計・表示機能が記述されています。このためスクリプトをエディタ等で変更することで、別のデバイスへの表示機能やデザインの変更も簡単に実現できます。

集計グラフLCD表示スクリプト名	Lua スクリプトソースファイル名
RASPI/FASTDB_LCD_CHART	scripts/RASPI/FASTDB_LCD_CHART.lua

下記の説明では、Raspberry Pi に OLEDモノクロ・グラフィックディスプレイ(SSD1306 128x64)を接続しています。最初に表示したいデータが格納されている FASTDB のキー一覧をコンソールで確認します。

```
192.168.100.18 - Tera Term VT
ファイル(F)  編集(E)  設定(S)  コントロール(O)  ウィンドウ(W)  漢字コード(K)  ヘルプ(H)
pi@mercury:~/abs_agent$ ./agent_fastdb

ServerName: mercury      Keys: 9
--------------------------------------------------------------------------------
<Key>=<RecordCount>
--------------------------------------------------------------------------------
温度_MQTT=14534
気圧_MQTT=14534
湿度_MQTT=14534
明るさ_MQTT=14534
T=0
RASPI4温度=15234
FreeMemMB=3047
FreeDiskGB=3047
CPUTemp=3047

pi@mercury:~/abs_agent$
```

上記では agent_fastdb コマンドでキー一覧を表示しています。この中でインストール時に自動で登録されるフリーメモリー容量を定期的に記録している "FreeMemMB" を選択してグラフ表示させます。

```
pi@mercury:~/abs_agent$ ./agent_script -s RASPI/FASTDB_LCD_CHART -k device,key -v 2,'FreeMemMB'

ServerName: mercury      ResultParam(s): 9
--------------------------------------------------------------------------------
<Key>=<Value>
--------------------------------------------------------------------------------
SumCount=96
SumMin=546
SumMax=598.667
Round=10
YMin=530
YMax=610
YLabel=530,546,562,578,594,610
XLabel=0,4,8,12,16,20,0
Status=OK

pi@mercury:~/abs_agent$
```

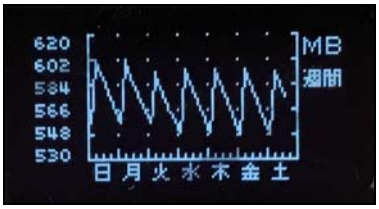
ここではスクリプト実行時に２つのスクリプトパラメータ "device" と "key" を指定しています。"device" は Raspberry Pi に接続しているディスプレイタイプで、"2" を指定することで SSD1306 128x64を選択しています。"key"は FASTDB に保存されている時系列データキーで、"FreeMemDB" を指定しています。スクリプト実行に成功すると指定したデバイスにグラフが描画され、グラフ表示時の各種パラメータがリターン値として返されます。

グラフ X軸には時系列(時間や曜日)が設定され、Y軸に集計値がアサインされます。集計範囲のパラメータを省略した場合には、デフォルトで当日１日分のデータを集計してグラフ表示します。

次は集計範囲を１週間にしてグラフタイトルを同時に指定した例です。集計範囲変更とタイトル表示のスクリプト・パラメータは "range"と"title" で、実行結果は以下の様になります。

```
pi@mercury:~/abs_agent$ ./agent_script -s RASPI/FASTDB_LCD_CHART -k device,key,title,range -v 2,'FreeMemMB','MB','week'

ServerName: mercury      ResultParam(s): 9
--------------------------------------------------------------------
<Key>=<Value>
--------------------------------------------------------------------
SumCount=84
SumMin=549.375
SumMax=603.708
Round=10
YMin=530
YMax=620
YLabel=530,548,566,584,602,620
XLabel=日,月,火,水,木,金,土
Status=OK

pi@mercury:~/abs_agent$
```

描画されたグラフ例は下記の様になります。



この他にもスクリプトパラメータを指定することで、様々な表示を行うことができます。スクリプト実行時に指定可能なパラメータは下記になります。

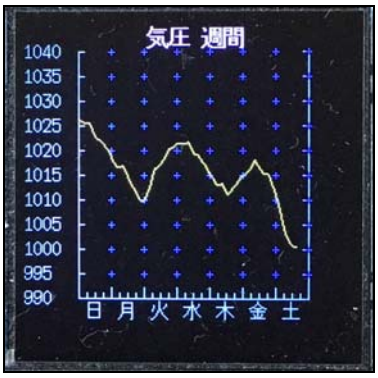
パラメータキー名	パラメータ値・説明
key(必須)	集計対象のデータが格納された FASTDB キー名
range	グラフの集計範囲を指定する。下記の値が有効で省略時は "day" を設定する。 "6h"　直近6時間 "day"　当日(24時間) "2d"　昨日と今日の二日間 "week"　直近1週間
title	グラフタイトル文字列。ASCII4文字 or 日本語2文字程度で指定 省略時または空文字列指定時はタイトルと集計範囲をチャートに表示しない。
round	Y軸ラベルの数字をどれぐらいの精度で丸めるかを決める。省略時は自動で決定する。
y_min	Y軸の最低値を指定する。省略時は自動で決定する。
y_max	Y軸の最大値を指定する。省略時は自動で決定する。
device	デバイスタイプを表す数値を指定します。省略時は "1" を設定する。 LCD 表示には abs_agent の graphic2 ライブラリを使用しています。現在 Raspberry Pi H/W 向けインストールキットに同梱されている gprahic2 用ドライバ(preloadライブラリ)で、デフォルトでサポートされている LCD デバイスは下記の通りです。 "1"　AQM1248A(128x48) "2"　OLED1306(128x64) "3"　ST7789 Color TFT (240x240) "4"　Pimoroni製 Display HAT Mini LCD(320x240)

grid	グラフ中の格子目盛の表示指定。省略時は"1"または"2"を自動設定する "0" 格子目盛を表示しない "1" ドットで表示 "2" 十字で表示

次の例では気圧センサーから取得した気圧データを表示します。ここでは Raspberry Pi に ST7789 タイプの TFT LCD(240x240) を接続して表示します。

```
pi@mercury:~/abs_agent$ ./agent_script -s RASPI/FASTDB_LCD_CHART -k device,key,title,range -v 3,'気圧_MQTT','気圧','week'

ServerName: mercury      ResultParam(s): 9
------------------------------------------------------------------
<Key>=<Value>
------------------------------------------------------------------
SumCount=84
SumMin=1000.62
SumMax=1026.13
Round=10
YMin=990
YMax=1040
YLabel=990,995,1000,1005,1010,1015,1020,1025,1030,1035,1040
XLabel=日,月,火,水,木,金,土
Status=OK

pi@mercury:~/abs_agent$
```

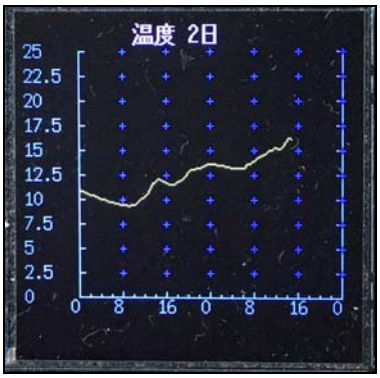
時系列データの "気圧_MQTT" では定期的に MQTT ブローカから配信されている環境センサーデータ中の気圧データを保存したもので、グラフ表示結果は以下の様になります。



グラフY軸のスケールとラベル文字は見やすい用に自動計算されますが、スクリプトパラメータで指定することもできます。下記は、グラフ中に表示する温度データのスケール値(最高値と最低値)を両方指定した場合の例です。

```
pi@mercury:~/abs_agent$ ./agent_script -s RASPI/FASTDB_LCD_CHART -k device,key,title,range,y_min,y_max -v 3,'温度_MQTT','温度','2d',0,25

ServerName: mercury      ResultParam(s): 9
------------------------------------------------------------------
<Key>=<Value>
------------------------------------------------------------------
SumCount=96
SumMin=9.39667
SumMax=16.3133
Round=2
YMin=0
YMax=25
YLabel=0,2.5,5,7.5,10,12.5,15,17.5,20,22.5,25
XLabel=0,8,16,0,8,16,0
Status=OK

pi@mercury:~/abs_agent$
```

y_min で グラフY軸の最低値を、y_max で最高値を指定します。データ値がスケール範囲を超えた場合にはその部分のグラフはプロットされません。表示結果は以下の様になります。



LCDグラフ表示スクリプトは、別のスクリプトやイベントハンドラからコールできますので、定期的な環境データ表示器などに応用できます。また、スクリプト内容をユーザーが変更することで、別のデバイスタイプへの対応や、集計範囲の追加なども簡単に対応可能です。

42 ライセンス・サポート

オールブルーシステムは、正規ライセンスを取得されたお客様の所有するコンピュータ１台に本ソフトウエアをインストールし使用する権利を提供します。

正規のライセンスを取得されたお客様は、本ソフトウエアと同時に使用する場合に限りオールブルーシステムの提供するクライアントプログラムや DLL ライブラリファイルを任意の個数のハードウエアにインストールすることができます。ただし、本ソフトウエアで同時に使用可能な数はライセンスで提供される接続可能デバイス数が上限となります。

お客様は、サーバーソフトウェア(サーバープログラム本体)を複製・複写することや、これに対する修正・追加等の改変をすることはできません。インストールキット中に含まれるイベントハンドラやスクリプトファイル、コンフィギュレーションファイル等については自由に複製や改変を行って、お客様のシステムで使用できます。クライアントプログラムについては任意の数のコンピュータに複製することができます。アクセス可能なクライアントホスト数は、ライセンスで提供される数が上限となります。

オールブルーシステムから発行したライセンスを、ライセンスに記載されたお客様とコンピュータ以外に ″オールブルーシステム製品″ として再ライセンスを行う事や、貸与又はリースその他の方法で第三者に使用させることはできません。ただし、オールブルーシステムからライセンスされたプログラムを組み込んで、お客様の製品やシステムとしてお客様が販売・サポートすることはできます。

オールブルーシステムから発行した法人向けライセンス製品をお客様の製品やシステムに組み込む場合に、オールブルーシステムで提供しているキットやライセンスを複製して第三者に提供することができます。このとき、イベントハンドラやスクリプトファイル、コンフィギュレーションファイル、ドキュメント、Webアプリケーション等をカストマイズした状態で提供することもできます。法人向けライセンス製品についての詳しい内容は問い合わせ下さい。

42.1 サポートについて

オールブルーシステムは上記ライセンス事項と取得された正規ライセンスファイルに記述された範囲においてサポートを行います。スタンダードライセンスではメールによる問い合わせ等のサポートを提供します。サポートで詳細な調査をするために、お客様の動作環境の資料やログ等の送付をお願いする場合があります、予めご了承願います。

abs_agent のスクリプトライブラリ関数(Luaのライブラリ)とEXCEL VBA から利用するための API ライブラリ関数(XASDLCMD.DLL)を利用する場合の、プログラミング方法やサンプルプログラムのソースコードに関する解説など、お客様のソフトウェア開発自身に関するサポートにつきましては、別途有償対応となります。詳しくはお問い合わせください。

abs_agent の正規ライセンスをご購入になる前に、目的の機能がお客様の環境で動作するかどうかを確認されることをお勧めします。オールブルーシステムの abs_agent は、全ての機能を事前にお客様が検証可能なように、フリー版ライセンスをキットに同梱して提供しています。業務使用用途や、スクリプト同時実行数を上げた場合の動作、MQTT の QoS を実運用と同等の条件で評価したい方には、期間を限定した評価用ライセンスの発行も可能です。詳しくはお問い合わせ下さい。

動作環境を満たしている場合でも、お客様の環境やハードウエアによっては正常に動作しない場合があります。最善のサポートをご提供致しますが、こちらで再現できない問題につきましては十分なサポートができない場合があります、事前にご了承下さい。

43 本製品に使用したソフトウェアライセンス表記

43.1 License for Lua 5.2.3

43.2 LovyanGFX ORIGINAL LIBRARY LICENSE

Software License Agreement (FreeBSD License)

Copyright (c) 2020 lovyan03 (https://github.com/lovyan03)

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

44 更新履歴

REV A.2.36 2025/2/19

new_tbl() で初期値に浮動少数値を指定した時に、整数に丸められる不具合を修正した

REV A.2.35 2025/1/13

abs_agent コンフィギュレーション項目に Script/APISearchOffset を追加した。

REV A.2.33 2024/11/25

Win32 アプリケーションやエクセル VBA 等から利用可能な XASDLCMD.DLL ライブラリに以下の関数を追加

AG_get_shared_strlist(),AG_size_shared_strlist()

REV A.2.32 2024/10/31

agent_shmem クライアントコマンドに IEEE-754 倍精度浮動少数値フォーマットを指定する -N オプションを追加した。

REV A.2.30 2024/10/21

Win32 アプリケーションやエクセル VBA 等から利用可能な XASDLCMD.DLL ライブラリに以下の関数を追加

AG_get_shmem_numarr(),AG_set_shmem_numarr(),AG_size_shmem_numarr(),,AG_remove_shmem_numarr(),

AG_log()

REV A.2.10 2024/2/24

shmem_store_num(), shmem_copy_num()ライブラリ関数を追加した。

REV A.2.08 2023/11/19

graphic2_draw_image() 関数で sec_x, src_y パラメータの説明文に誤りがあったのを修正

graphic2_print() で文字列描画範囲を示す max_x, max_y 座標値を返すようにした。

REV A.2.07 2023/10/30

agent_fastdb でタイムスタンプ値にミリ秒を指定可能にした。

agent_fastdb の検索結果をミリ秒まで表示するように変更した。同時に検索結果のファイルダンプやロード時にもミリ秒まで対応するようにした。

agent_fastdb のバックアップ・リストア・バックアップ削除時にキー指定を可能にした。

fastdb_add() のタイムスタンプ値にミリ秒指定を可能にした。

fastdb_restore_shmem() ライブラリ関数に adj_sec パラメータを追加して、タイムスタンプ値を補正してから登録する機能を追加した。

agent_fastdb のリストア時にタイムスタンプ補正オプション "-a <adj_sec>" を追加した。

REV A.2.05 2023/9/18

raspi_spi_config() の cs パラメータ値に "cs2", "reserved" 追加

agent_fastdb バックアップ時にレコード数が 0 のものを除外するようにした。

graphic2 モジュールの添付ドライバに Pimoroni社製 Display HAT Mini を追加した。

RASPI/FASTDB_LCD_CHART.lua に Pimoroni社製 Display HAT Mini 用パラメータを追加した。

REV A.2.04 2023/9/9

keys_shared_data(),keys_net_data() APIに前方一致検索のための key_prefix パラメータ追加

agent_fastdb プログラムにリモートコンピュータへのバックアップ・リストア機能を追加した。

REV A.2.03 2023/8/6

fastdb_backup_shmem(), fastdb_restore_shmem() ライブラリ関数追加

REV A.2.02 2023/7/15

agent_shmem クライアントプログラムにメモリエリア転送オプション "-X <target>" とチャンネル名変更オプション "-n <new_channel>"を追加した

shmem_transfer(), shmem_rename() ライブラリ関数追加

REV A.2.01 2023/6/27

agent_net クライアントプログラムを追加した

汎用UDPサーバー機能を NETSERVER サービスモジュールに追加した

UDPSERVER_DATA イベントハンドラ追加

REV A.2.00 2023/4/20

agent_data クライアントプログラムで単精度浮動小数点表示に対応した

汎用TCPサーバーや ModbusTCP サーバーとして利用可能な NETSERVER サービスモジュールを追加した

TCPSERVER_DATA イベントハンドラ追加

hex_to_num(), num_to_hex() ライブラリ関数で単精度浮動小数点フォーマットを扱えるようにした

REV A.1.98 2022/11/28

APPENDIX(B)章に集計グラフLCD表示アプリ(FASTDB_LCD_CHART)説明追加

millis_delta() に update パラメータ追加

コンフィギュレーションファイルに RASPI/HWCheckInterval 項目を追加

REV A.1.97 2022/11/6

モノクロビットマップ専用の graphic ライブラリ API を削除した。

上位互換の graphic2 ライブラリに移行。

REV A.1.94 2022/9/6

script_signal() 関数でスクリプト名パラメータを指定可能にした。

REV A.1.93 2022/9/1

シグナル操作関連の機能を追加した。

(script_signal()ライブラリ関数、g_signal 変数、agent_task -v <signal> オプション)

REV A.1.92 2022/7/24

log_msg_strlist()ライブラリ関数を追加した。

REV A.1.91 2022/6/18

serial_available(), serial_readln() の内部動作フローを改良してスピードアップした。

シリアルデバイスをバッファモードで使用する場合に、バッファに保持可能な最大レコード数設定 MaxBufferRecordCount をサーバー設定ファイルに追加した。

シリアルデバイス操作用のクライアントプログラム agent_serial を追加した。

REV A.1.90 2022/6/8

graphic2 を使用したLCDデバイス用のライブラリ関数を追加した(AQM1284A, OLED_SSD1306, ST7789)

REV A.1.87 2022/5/21

millis_delta() ライブラリ関数を追加した。

REV A.1.86 2022/5/11

preload 機能で検索対象となる lua ファイルから "_app_", "_cpy_" のファイル名で始まるファイルを除外するようにした。

"g_dir" グローバル変数を追加した。

now_datetime() ライブラリ関数を追加した。

REV A.1.85 2022/5/1

graphic2_load_rgb_file() 関数を追加した。

REV A.1.84 2022/4/20

load_file_tbl() 関数を追加した。

REV A.1.82 2022/2/6

graphic2 ライブラリに画像描画関数を追加した。

REV A.1.82 2022/2/6

グローバル共有メモリのチャンネルにディスプレイバッファ用の属性値を保存可能にした。

graphic2 ライブラリを新規追加した

REV A.1.81 2022/1/31

shmem_set()関数に offset_adj パラメータを追加した。

shmem_move() 関数を追加した。

agent_shmem コマンドに -m (移動)オプションを追加した。

REV A.1.80 2022/1/15

raspi_spi_xxxx(), raspi_i2c_xxxx() 関数において、動作クロックが高速(約 1MHz 以上)時に転送スピードが低下する問題に対応した。

raspi_spi_write_shmem() 関数に fill_arr パラメータを追加した。

REV A.1.79 2021/11/17

event_set2x(), event_wait2x(),event_wait_either2x() ライブラリ関数をそれぞれ event_set2(), event_wait2(),event_wait_either2() に統合した。(旧関数名でコールすることも可能)

raspi_spi_write_shmem(), raspi_i2c_write_shmem() ライブラリ関数を追加した。

REV A.1.78 2021/10/31

Raspberry Pi のハードウエアタイプに "BCM2710" を追加した。

REV A.1.76 2021/3/3

agent_task コマンドに全スクリプトプールエントリを表示するオプション "-a" を追加した。

FASTDB_PURGE スクリプトで保存期間をグローバル共有変数から取得するように変更

REV A.1.74 2021/2/8

agent_copycfg コマンドにスクリプトファイル内容チェック用オプション "-t" を追加した。

REV A.1.73 2021/1/25

script_exists()ライブラリ関数を追加した。

agent_mgcp コマンドに "heartbeat" イベント送信オプション "-e" を追加した。

ユーザー動作環境作成・更新用クライアントプログラム "agent_copycfg" を新規に作成した

REV A.1.72 2021/1/3

str_to_tbl2() に sjis_char パラメータを追加した。

REV A.1.71 2020/12/21

grahpic_OLED1306_try_init(),graphic_OLED1306_display(),graphic_draw_fontx()ライブラリ関数追加

shmem_get_fontx() ライブラリ関数で半角文字フォントを取得できるようにした。

log_msg(),log_hexdump() ライブラリのモジュール名パラメータ省略時のデフォルト値を、g_script に変更した。

graphic_print2() ライブラリ関数を追加した。

REV A.1.70 2020/11/24

Raspberry Pi4 (BCM2711) の SPI#3-#6, I2C#3-#6 サポート用にパラメータを追加

Raspberry Pi 用ライブラリのI2C(BSC)バスと SPIポートの GPIO Altモードの設定について説明項目を追加

raspi_gpio_pud() ライブラリ関数追加

REV A.1.62 2020/11/2

fastdb_summary() 集計ライブラリで first(最初の値) をリターン値に追加

fastdb_temp_key_list(), fastdb_temp_unlock(), fastdb_sync() ライブラリ関数追加

REV A.1.61 2020/9/13

FASTDB 集計アプリに CSV ダウンロード機能を追加

REV A.1.60 2020/8/3

log_hexdump() ライブラリ関数追加

agent_mgcp クライアントプログラムにダイレクトコマンド実行オプションを追加

REV A.1.59 2020/6/8

ログサーバーの説明を別の章に分離

udp_send_recv_binary() ライブラリ関数追加

REV A.1.58 2020/4/26

udp_send_binary() ライブラリ関数追加

REV A.1.57 2020/4/1

stat_check() ライブラリ関数追加

mgcp_command(), mgcp_event(), mgcp_find_endpoint() ライブラリ関数の記述を削除して、これらの説明を "MGCP ユーザーマニュアル" に移動した。

FASTDB グラフアプリに累計値のプロット機能を追加した。

REV A.1.54 2019/12/21

shmem_fontx_bitmap() ライブラリ関数追加

graphic_set_size() 関数で現在設定されているデバイスサイズを返すようにした。

REV A.1.53 2019/11/22

Raspberry Pi用 abs_agent の ハードウエアタイプに "BCM2711" (Raspberry Pi ver4用)を追加した。

誤字修正・その他

REV A.1.52 2019/9/11

Serial モジュールのデバイスタイプに "ZB" XBee-ZB Zigbee (S2) を追加した。

XBee-ZB Zigbee API 用のライブラリ関数を追加した。

agent_zb クライアントプログラムを追加した。

master_item_list() がマスターに含まれる最大のタグ数分のエントリを返す様に変更した。

master_item_tagval() ライブラリ関数を追加した。

サーバー設定ファイルに UseGlobalWatch 項目を追加した。

REV A.1.51 2019/8/4

"XBee 802.15.4 デバイス接続"の章中に、XBee3 を使用する場合の初期設定項目を追加した。

REV A.1.50 2019/7/27

Serial モジュールのデバイスタイプに "XBEE" XBee 802.15.4 (S1) を追加した。

Serial モジュールのコンフィギュレーションにデバイスタイプ毎の説明文を追加した。

agent_task クライアントプログラムに全タスクを強制終了させるための -K オプションを追加した。

agent_data クライアントプログラムにキー名に前方一致するデータを削除するための -D オプションを追加した。

master_exist() ライブラリ関数を追加した。

XBee 802.15.4 API 用のライブラリ関数を追加した。

agent_xbee クライアントプログラムを追加した。

tbl_to_hex() ライブラリ関数に部分変換用のパラメータを追加した。

REV A.1.42 2019/3/2

グローバル共有変数、グローバル共有文字列リストに、Lua数値型を扱うためのライブラリ関数を追加した。

REV A.1.41 2019/2/23

XASDLCMD.DLL の API 説明から DeviceServer 専用のライブラリ関数を省略した。

Lua数値型(整数値、浮動少数点値)と16進数文字列を相互変換する、num_to_hex(), hex_to_num() ライブラリ関数を追加した

REV A.1.40 2019/1/1

WebSocketサーバー機能を追加した。

websocket_emit_text(), websocket_emit_binary() ライブラリ関数を追加した。

WEBSOCKET_DATA イベントハンドラを追加した。

WebAPI の /command/json/session_login でログインに成功したときに、WebSocketServer機能が有効になっていた場合には WebSocketServerPort タグにWebSocketServer のポート番号を返すようにした。

REV A.1.34 2018/11/12

FASTDB RawDataチャート Web アプリケーションを追加した。

REV A.1.33 2018/10/23

fastdb_search(), fastdb_add_net() ライブラリ関数を追加した。

REV A.1.32 2018/10/22

グローバル共有データ、グローバル共有文字列リスト、FASTDB のキー等や値に指定する文字列長の制限の記述を無くした。

agent_mgcp コマンドにデバイス・コンフィギュレーションのダウンロードとアップロード機能を追加した。

REV A.1.31 2018/9/22

get_shared_data(), set_shared_data() ライブラリ関数に、複数データを同時に取得・更新する機能を追加した。

REV A.1.30 2018/9/18

Web API /command/json/script の noquote=1 指定時に、リターンパラメータのキー名 "Status" を除外するようにした。

event_wait2x(), event_wait_either2x(), event_set2x() ライブラリ関数を追加した。

script_result() 関数の taskid パラメータを省略可能にした。

REV A.1.29 2018/8/9

コンフィギュレーションファイルの設定項目に $CONFIGDIR$ を指定できるようにした。

REV A.1.28 2018/7/25

agent_mgcp コマンドを追加した。

fastdb_key_list() 関数に locked_only オプションパラメータを追加した。

fastdb_unlock() 関数を追加した。

find_shared_strlist(), find_net_strlist() 関数を追加した。

mqtt_mgcp_command(), mqtt_mgcp_event() ライブラリ関数名を mgcp_command(), mgcp_event() にそれぞれ変更。また、MQTT ブローカを自動で選択するための mgcp_find_endpoint() ライブラリ関数追加。

REV A.1.27 2018/6/25

agent_fastdb コマンドに -l オプションを追加した。

g_json.encode() 関数を追加。

REV A.1.26 2018/5/14

マルチバイト文字に対応した str_to_tbl2() ライブラリ関数を追加した。

REV A.1.25 2018/4/23

preload フォルダ検索時にアンダースコア文字 '_' で始まるフォルダを除外するようにした。

REV A.1.24 2018/3/20

abs_agent.xml 設定ファイルの シリアルデバイスと MQTT エンドポイントエントリに "Enabled" 項目を追加した"

new_tbl() ライブラリ関数を追加した。

shmem_fill() ライブラリ関数を追加した。

shmem_set() ライブラリ関数に複数データを同時に登録するパラメータを追加した。

agent_shmen クライアントに -z <fill_byte_data> パラメータを追加した。

REV A.1.23 2018/2/6

agent_script に -x, -y パラメータを追加した。また -k, -v にカンマ区切り文字列を指定することで、複数のリクエストパラメータを指定可能にした。

hex_to_bit() ライブラリ関数を追加した。

REV A.1.22 2018/2/1

abs_agent ライセンスの記述について一部変更。

mqtt_mgcp_command(), mqtt_mgcp_event() ライブラリ関数追加。

REV A.1.21 2017/12/7

agent_fastdb コマンドにロックを強制解除するための "-u" オプションを追加した。

REV A.1.19 2017/11/4

topic_to_tbl()API 関数追加

abs_agent.xml 設定ファイル中の MQTT 自動トピック購読に自ホスト名を示す $HOSTNAME$ を指定可能にした。

REV A.1.19 2017/8/22

raspi_i2c_clock()API 関数追加

REV A.1.18 2017/8/21

グローバル共有メモリエリア操作用のクライアントプログラム agent_shmem 追加

shmem_store() のアペンドモード追加

shmem_load_file(), shmem_save_file() API 関数追加

REV A.1.17 2017/7/27

ライセンス項の一部表現の修正(内容は以前と同じです)

グローバル共有メモリエリア機能追加。(APIライブラリ)

Raspberry Pi のセットアップ手順で Hardware 情報を書き込むように変更した。

REV A.1.16 2017/5/22

TCP,UDP クライアント API ライブラリ関数追加

REV A.1.15 2017/4/19

時系列集計用インメモリデータベース FASTDB 用 API、クライアントプログラム agent_fastdb を追加

REV A.1.14 2017/3/9

event_set(), event_wait() API を追加

REV A.1.13 2017/3/1

グラフィックライブラリ API を追加

REV A.1.12 2017/1/22

ライブラリ関数 raspi_spi_write_byte(), raspi_spi_read_byte() 追加

REV A.1.10 2017/1/11

Web サーバー、Web API 機能追加

クライアントプログラム agent_webuser, agent_session, agent_strlist 追加

Win32 DLLライブラリにグローバル共有リスト関連の関数追加

REV A.1.00 2016/8/11

初版作成