ABS-9000 DeviceServer ユーザーマニュアル

ABS-9000 DeviceServer

DeviceServer User Manual Rev A.2.29

2017/11/19

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 このマニュアルについて

1.1 著作権および登録商標

WindowsXP, Windows2000, Windows 2003 Server, Windows7 はマイクロソフト社の登録商標です。

1.2 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

2 使用条件およびライセンス

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パーソナルコンピュータ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I/O デバイス用ファームウエア (UIOUSB、SigSensor、NetUIO、TDCP等)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の全部又は一部を複製・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。クライアントプログラム (ABLauncher.exe) については任意の数のPC に複製することができます。同時使用可能なクライアントプログラム数は、ライセンスで提供される同時ログインセッション数が上限となります。

オールブルーシステムは、ライセンスの再発行や別のパーソナルコンピュータへのライセンスの変更は行いません。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パーソナルコンピュータ以外に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、お客様のシステムや製品に組み込んで、お客様が販売・サポートすることについては、オールブルーシステムはそれを制限しません。

3　機能概要・イントロダクション

ABS-9000 DeviceServerはネットワークに接続された複数のアラームデバイス(SigSensor, NetUIO[1])や、USB に接続された汎用I/Oデバイス(UIOUSB[2])を統合して管理するためのソフトウエアです。以下に ABS-9000 DeviceServer を使用することで実現できる事を紹介します。

(1) シリアルポートに接続した計測器や I/O 装置、Arduino[3] CPU ボード等を接続することができます。

(2) Digi international Inc. 社製の XBee[4] モデムデバイスの XBee IEEE 802.15.4 OEM RF Module(シリーズ１)または、 XBee-ZB Module(シリーズ２) をDeviceServer に接続して、複数の XBee デバイスを DeviceServerから管理することができます。

(3) シリアルポートに、東京コスモス電機社製の TOCOS[5] ワイヤレスエンジン[6]を使用した TWE デバイス[7]を接続して、複数のTWE デバイスをリモートコントロールすることができます。

(4) MQTT[8] 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 DeviceServer のスクリプトで処理することができます。DeviceServer 側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) DeviceServer は簡単なユーザースクリプトやイベントハンドラを記述することで、様々な機能を持ったセンサーネットワークシステムを作ることができます。ネットワーク接続されたセンサーデバイスの I/O 操作やI/Oポート、ADC の値をリモートから自動取得して防犯システムなどを簡単に作成することができます。様々な種類のセンサーやI/O 装置を管理することができるため、オートメーションシステム等も簡単に作成することができます。

(6) 現在運用中の業務システムなどでエラーが発生した時に、外部のランプや警報機などに通知したい場合にも簡単に組み込むことができます。業務システム側はエラーイベント発生時に DeviceServer 側のスクリプトをリモートから起動するように修正するだけで実現できます。業務システムが運用中でも、警報デバイスの構成変更(出力先 I/O の変更など)を、DeviceServer 側のスクリプトを変更するだけで柔軟に対応できます。

(7) インターネットメール経由でDeviceServer を外出先からコントロールすることもできます。また、センサーノ

---

[1] オールブルーシステムの提供する SigSensor,NetUIO ファームウエアを組み込んだネットワーク I/O デバイス。詳細は SigSensor_NetUIO ユーザーマニュアルをご覧ください。
[2] オールブルーシステムの提供する UIOUSB ファームウエアを組み込んだ USB 接続の汎用 I/O デバイス。詳細は UIOUSB ユーザーマニュアルをご覧ください。
[3] http://www.arduino.cc/
[4] XBee® and XBee PRO® are registered trademarks of Digi, Inc.
[5] TOCOS は東京コスモス電機株式会社の登録商標です
[6] TOCOS Wireless Engine は東京コスモス電機株式会社の登録商標です
[7] 東京コスモス電機株式会社で提供されるワイヤレス製品(TWE-Lite 等)を示します
[8] MQTT http://mqtt.org/

ードで発生したイベント（I/O データ入力の変化やボタンが押された時や、A/D 変換値が設定された制限値を超えた時など..）で、ユーザーが予め定義したスクリプトを実行して処理結果をメールで携帯電話に送信させることもできます。

(8) DeviceServer で実行するスクリプトは、特別な開発環境等は使用しないでテキストエディタのみで作成ができます。 DeviceServerのログ機能を使用してスクリプト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[9]（ver5）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。サンプルスクリプトを多数同梱していますので、簡単にユーザスクリプトを作成できます。

(9) 既存のシステムやアプリケーションに組み込むために、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。

(10) 外部の Oracle[10]データベースへ、スクリプトやAPI から任意の文字列を出力する機能があります。(ABS-9000 DeviceServer エンハンスライセンス時のみ) デバイスから取得した計測データをOracleデータベースに保存して、外部のデータベースシステムから集計やレポート作成などを行うこともできます。Oracle データベース接続機能については、”Oracle接続機能” の章を参照してください。

(11) スマートフォンやタブレット端末から GUI を使用して、DeviceServer をコントロールすることもできます。HTML5 アプリケーション(HTML + JavaScript + CSS ファイルで構成されています) を DeviceServer 内蔵の HTTP サーバー機能と Web API 機能を使用して簡単に作成できます。

DeviceServer の主な機能は以下の通りです:

- アラームデバイス(SigSensor, NetUIOデバイス)、UIOUSBデバイス操作
- シリアルポートに接続された複数のArduino デバイスの管理
- シリアルポートに接続された計測器等の管理
- XBee デバイス(XBee 802.15.4 OEM RF Module シリーズ1) の操作及び、複数の XBeeデバイスの管理
- XBee-ZB デバイス(XBee-ZB Module シリーズ2) の操作及び、複数の XBee-ZBデバイスの管理
- TOCOS TWE ワイヤレスデバイスの操作
- MQTT ブローカへの接続機能
- ユーザースクリプト実行
- アラームデバイスで発生したイベント通知
- アラームデバイス管理
- ユーザー管理、認証
- メールによるコントロールや、メール送信
- API(DLL)経由でのコントロール（EXCEL VBA 等）

[9] Lua (http://www.lua.org/about.html)
[10] Oracle は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。

- Webブラウザからの操作
- サーバー動作ログ管理
- 複数の DeviceServer を使用した分散処理システムの構築

次に、各々の機能について詳しく説明します。

3.1 デバイス操作

DeviceServerで管理されたネットワーク接続された I/O デバイスを、スクリプトやAPI（後述）を使用してコントロールすることができます。コントロール可能な主な操作は以下の通りです。（コントロール対象となるデバイスによって操作可能な機能に違いがあります）

- デジタル I/O データ出力
- デジタル I/O データ入力
- A/D 変換入力
- PWM 出力
- 簡易サーボ信号出力
- アラーム信号(LED, ブザー) 出力
- LCD（液晶表示器）に文字出力
- デバイスのコンフィギュレーション取得/設定

デバイスが複数の場合にも、それぞれのデバイス毎に細かい操作ができます。操作を行う時の詳しい仕様については“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。

XBee デバイスを操作する場合は、DeviceServer のCOM ポートに直接接続されたXBee を経由して、同一PAN(Personal Area Network) 内にある複数の XBee デバイスをリモート操作することができます。XBee デバイスは、シリーズ1(XBee 802.15.4 OEM RF Module)とシリーズ2(XBee-ZB Module)の両方をサーポートしています。同時に両方のタイプの XBee モジュールを DeviceServer に接続して使用することもできます。

3.2 ユーザースクリプト

スクリプトは DeviceServer をユーザーの目的に応じたシステムにカスタマイズする時の、中心となる機能です。DeviceServerは、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、操作のフローコントロールを行うためにスクリプトエンジンを内蔵しています。DeviceServer のスクリプトエンジンには、高速動作やシンプルで柔軟な記述が可能な Lua（ver5.1）を採用しています。

DeviceServer は以下のような時に Lua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（ユーザー認証付）
- Webブラウザで ScriptControl ページをアクセスして、指定したスクリプトを実行（ユーザー認証付）
- 電子メール経由でスクリプト実行を指示（ユーザー認証付）

- デバイスや DeviceServer 自身の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

イベントハンドラとして実行されるスクリプト以外は予め DeviceServer にログイン操作を行って、正しいユーザーであることを認証してから、実行可能になります。ログイン認証を行うことで、セキュリティを確保し、権限のないユーザーから不用意にシステムを操作されることを防いでいます。

DeviceServer 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。これは DeviceServerの持つ機能をフルに使用可能な様にライブラリ関数としてユーザーが利用可能なようになっています。ただし、DeviceServerがWindows サービスで動作しますので、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。

スクリプトはテキストエディタ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-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**。

作成したスクリプトはDeviceServer のインストールフォルダ中の Scriptsフォルダ (“C:¥Program Files¥AllBlueSystem¥Scripts”) に拡張子 (.lua) のファイル名で保存してください。Windows7(64bit)の場合には、“C:¥Program Files (x86)¥AllBlueSystem¥Scripts”フォルダになります。

スクリプトファイルを作成または修正すると、直ぐに DeviceServer から使用可能できます。DeviceServer からスクリプト名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DeviceServer の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 フォルダ内にサブフォルダを作成して、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、サブフォルダ名を ‘/’ 文字で区切ってスクリプト名を指定してください。(Scriptsフォルダ “C:¥Program Files¥AllBlueSystem¥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)

上記Scriptsフォルダ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) とサンプル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。
スクリプトの利用方法の具体的な手順は、”動作確認と簡単な使い方” の章を参照することで、作成の手順を理解することができます。

Lua 言語の詳しい仕様と資料は http://www.lua.org/、http://www.lua.org/manual/5.1/ を参照してください。またDeviceServerで拡張された関数の詳細は “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。エンハンス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することができます。詳しくは、”サーバーソフトウエア” の章中の ”サーバー設定プログラム(ServerInit)” の項目を参照してください。

3.3 イベント通知

DeviceServerには、アラームデバイスやサーバーの各サービスモジュールで発生するイベントがあります。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サーバーで実行します。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、共通のスクリプトエンジン(Lua5.1) を使用しています。イベントハンドラで実行される場合は、イベントに関する情報がパラメータ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、詳しいパラメータについては “イベント”の章を参照してください。

イベントハンドラ内では、DeviceServer 用に追加されたライブラリ関数を利用して、イベントハンドラ間のデータの受け渡しや、他のシステムへの UDP/TCP データ送信、ログ出力、Oracleデータベースへのデータ保存ができます。DeviceServer で管理されたアラームデバイスや UIOUSBデバイスの操作を行うこともできます。

DeviceServer で定義されたイベントの種類は以下のものがあります。イベントは常に発生するものと、”サーバー設定プログラム” でサービスモジュール毎にイベント通知をするかどうかを設定できるものがあります。

イベント名(スクリプト名)	イベント発生条件
ALARM_BUTTON_PUSH	SigSensor, NetUIO デバイスのプッシュボタンが押された
ALARM_DINPUT_CHANGE	SigSensor, NetUIO デバイスのDINPUT 値が変化した
ALARM_ADRANGE_EXCEED	SigSensor, NetUIO デバイスのA/D 変換値が予め設定された上限値または下限値を超えた
ALARM_NETWORK_ERROR	SigSensor, NetUIO デバイスとの通信でネットワークエラーを検出した
MAIL_NEW_MAIL	メールサーバー(POP3サーバー)に新規メールが届いた
PERIODIC_TIMER	定期的に発生するタイマーイベント(１分毎に発生)
XBEE_MODEM_STATUS	XBee 802.15.4 Series1 デバイスから、データフレーム(APIType = 0x8A) を受信した
XBEE_IO_DATA	XBee 802.15.4 Series1 デバイスから、XBee I/O データフレーム(APIType = 0x82, 0x83) を受信した
XBEE_IO_OTHER	XBee 802.15.4 Series1 デバイスから、XBee I/O データフレーム以外

	(APIType =0x82, 0x83)を受信した
XBEE_PACKET_DATA	XBee 802.15.4 Series1 デバイスから、XBee ReceivePacketデータフレーム(APIType = 0x80, 0x81) を受信した
XBEE_TDCP_DATA	XBee 802.15.4 Series1デバイスから、XBee ReceivePacketデータフレーム(APIType = 0x80, 0x81) を受信してかつフレームデータの最初の文字が “$$$” で始まる場合
ZB_MODEM_STATUS	XBee-ZB Series2 デバイスから、XBee データフレーム(APIType = 0x8A) を受信した
ZB_IO_DATA	XBee-ZB Series2 デバイスから、XBee I/O データフレーム(APIType = 0x92) を受信した
ZB_OTHER	XBee-ZB Series2 デバイスから、XBee データフレーム(APIType = 0x91,0x94,0x95,0xA1,0xA2,0xA3,0xA4) を受信した
ZB_PACKET_DATA	XBee-ZB Series2 デバイスから、XBee ReceivePacketデータフレーム(APIType = 0x90) を受信した
ZB_TDCP_DATA	XBee-ZB Series2 デバイスから、XBee ReceivePacketデータフレーム(APIType = 0x90) を受信してかつフレームデータの最初の文字が “$$$” で始まる場合
UIOUSB_EVENT_DATA	UIOUSB デバイスから “$$$” で始まるイベントデータを受信した
SERVER_START	DeviceServer 起動時に有効にした全てのサービスモジュールが利用可能になったときに最初に１回だけ発生します
SERVER_STOP	DeviceServer 終了時に発生します。
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した
SERIAL_FIRMATA	“FIRMATA” タイプに定義されたシリアルポートからパケットを受信した
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した
SERIAL_TWE	"TWE" タイプに定義されたシリアルポートからTWE ワイヤレスデバイスのアスキー形式のデータパケットを受信した
DEVICE_MESSAGE	ネットワークで接続されたデバイスから、CSVIF プロトコルで “DEVICE_MESSAGE” パケットを受信した
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発生します
MQTT_KEEP_ALIVE_TIMER	MQTT サービスモジュールを有効にしたときに、KeepAliveTimer 項目に設定した間隔で発生します
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発生します

インストールフォルダ中の Scriptsフォルダ (“C:¥Program Files¥AllBlueSystem¥Scripts”または "C:¥Program Files (x86)¥AllBlueSystem¥Scripts") にはそれぞれのイベントに対してデフォルトイベントハンドラ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)このデフォルトイベントハンドラファイルをエディタで修正して使用します。

スクリプトとイベントハンドラは DeviceServer 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やイベントハンドラが同時に実行指示された場合は、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。

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）。イベントハンドラの処理が終了するまで、イベントを送信したアラームデバイス(SigSensor, NetUIO)では次のイベント送信をペンディングします。ペンディング中に発生したイベントは、デバイス内部でキューイングされていますが、制限値を超えた場合にはそれらのイベントは破棄されますので、注意してください。アラームデバイスのイベント送信の仕様については、”SigSensor_NetUIOユーザーマニュアル” を参照してください。

XBee、XBee-ZB デバイスからフレームを受信した場合も同様にイベントハンドラを実行しますが、XBeeデバイスではイベントハンドラの終了を待たずに、次に到着したフレーム処理を並行して別スレッドで行います。イベントハンドラの処理時間が長くなると、同じイベントハンドラを複数同時実行します。

3.4 デバイス管理

DeviceServerで操作可能な I/O やアラームデバイス、エンドポイントは以下のものがあります。

- **UIOUSB**

  DeviceServerの動作しているPC のUSBポートに接続された汎用 I/O デバイス(最大一台)接続できます。UIOUSBの詳しい仕様については、”UIOUSBユーザーマニュアル”を参照して下さい。

- **SigSensor**

  ネットワーク接続された、アラーム機能と汎用 I/O 機能を持ったデバイスを接続できます。SigSensor, NetUIO, アラームシグナルデバイスをスタンダードライセンスで最大２台、エンハンスライセンスで最大１０台まで接続可能

  SigSensor の詳しい仕様については、” SigSensor_NetUIOユーザーマニュアル”を参照して下さい。

- **NetUIO**

  ネットワーク接続された、汎用 I/O 機能を持ったデバイスを接続できます。SigSensor, NetUIO, アラームシグナルデバイスをスタンダードライセンスで最大２台、エンハンスライセンスで最大１０台まで接続可能です。

  NetUIO の詳しい仕様については、” SigSensor_NetUIOユーザーマニュアル”を参照して下さい。

- **シリアル接続された Arduino CPUボードや計測器、I/O 装置**

  シリアルポート経由で接続された、Arduino 互換デバイスや計測器などを接続できます。スタンダードライセンスで最大２台、エンハンスライセンスで最大１０台まで接続可能です。

- **XBee 802.15.4 OEM RF Series1 モジュール**

  Digi international Inc.社製の XBee 802.15.4 シリーズ１ デバイスを接続することができます。接続には、DeviceServer の COM ポート経由で１つの XBee デバイスを接続します。このデバイスを経由して、同一PAN(Personal Area Network)内に設置された、複数のリモートXBee デバイスを全て操作することができます。DeviceServer で登録可能なリモートXBee デバイスの数に制限はありません。XBee デバイスの仕様のみに依存します。

XBee デバイス上の 8つの DIO について、I/O 機能と A/D 変換の設定ができ、2チャンネルのPWM出力が可能です。リモートXBee デバイスの入力ポートデータの変化で、DeviceServer 側でイベントハンドラを実行することができます。(XBee デバイスの Change Detect 機能を利用します)

- **XBee-ZB (ZigBee 対応 Series2)モジュール**

  Digi international Inc.社製の XBee-ZB シリーズ２ デバイスを接続することができます。接続には、DeviceServer の COM ポート経由で１つの XBee-ZB デバイスを接続します。このデバイスを経由して、同一PAN(Personal Area Network)内に設置された、複数のリモートXBee-ZB デバイスを全て操作することができます。DeviceServer で登録可能なリモートXBee-ZB デバイスの数に制限はありません。XBee-ZB デバイスの仕様のみに依存します。

  XBee-ZB デバイス上の 8つの DIO 機能と 4つのA/D 変換の設定ができます。リモートXBee-ZB デバイスの入力ポートデータの変化で、DeviceServer 側でイベントハンドラを実行することができます。(XBee-ZB デバイスの Change Detect 機能を利用します)

- **TWE ワイヤレスデバイス**

  TWEワイヤレスデバイスを DeviceServerに接続して、このデバイスを経由して同一アプリケーションIDに設定された複数の TWEワイヤレスデバイスを操作することができます。DeviceServerで操作可能なリモート側のTWEワイヤレスデバイスの数に制限はありません(TWE ワイヤレスエンジン側の制限のみに依存します)。

  DeviceServer に接続する TWEワイヤレスデバイスはシリアルデバイスとして複数同時に接続できますので、それぞれのシリアルポートに別々のアプリケーションID を持った TWE ワイヤレスデバイスを接続して、複数のアプリケーションID グループ内のリモートデバイスを同時に全て制御することもできます。

- **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 DeviceServer 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。

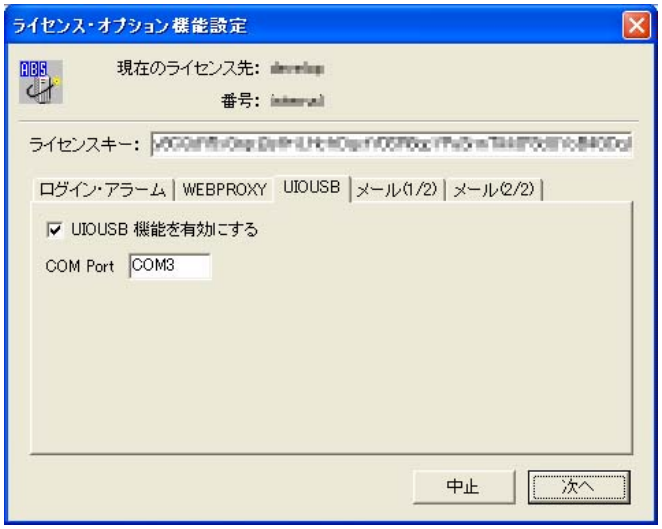
- **アラームシグナルプログラム**

  任意のPC で動作させることが可能な、ネットワーク接続されたアラーム機能を持ったアラームシミュレータプログラム。SigSensor のランプの点灯と点滅、２種類のブザー出力ができます。同じPC 上に別ポート番号アサインすることで、同時に複数のアラームシグナルを起動できます。SigSensor ,NetUIO, アラームシグナルデバイスをスタンダードライセンスで最大２台、エンハンスライセンスで最大１０台まで接続可能です。

アラームシグナルプログラムについては、”デバイス管理”の章中の “アラームシグナルプログラム”の項目と、 ”その他のプログラム”の章中の”アラームシグナル(AlarmSignal)”の項目を参照してください。

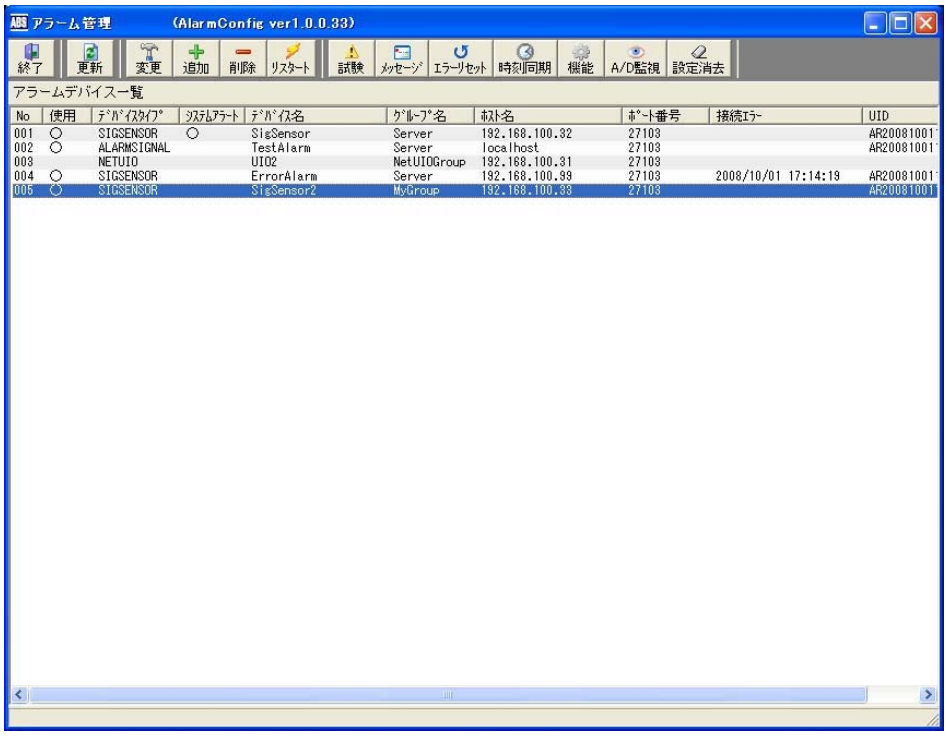
UIOUSB デバイスは、DeviceServer の動作しているPC に接続してドライバをインストールするだけで使用できます。

ドライバーインストール時に作成された仮想 COM ポートの値を “サーバー設定” プログラムで指定します。UIOUSB の作成方法やインストール方法については “UIOUSBユーザーマニュアル” を参照してください。



（サーバー設定プログラムの UIOUSB 設定部分）

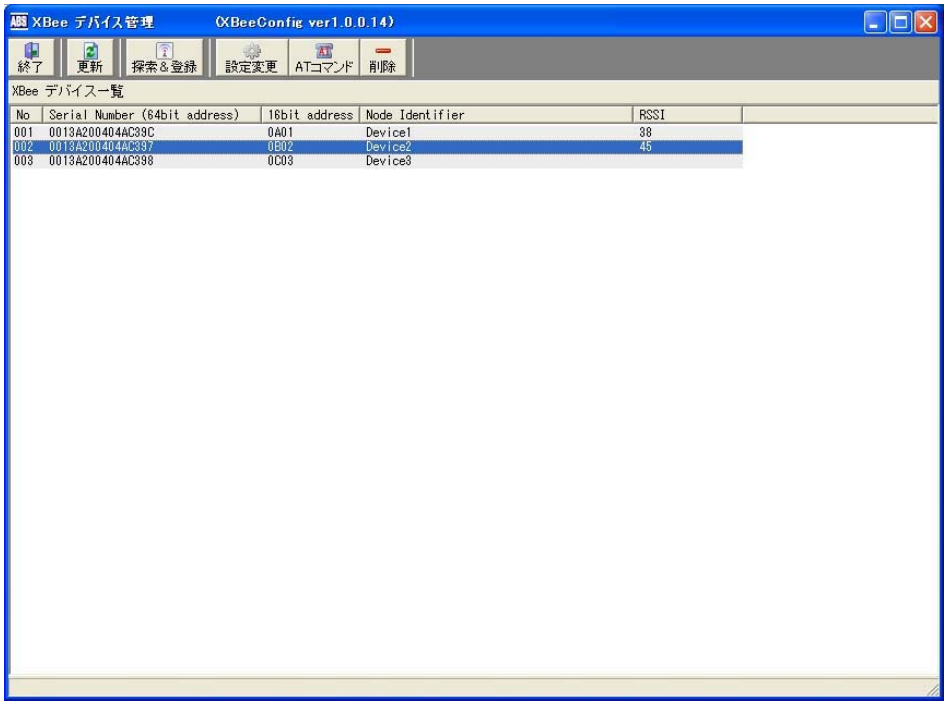
アラームデバイス(SigSensor ,NetUIO, アラームシグナル) の管理は、クライアントソフトを起動して、アラーム管理(AlarmConfig)プログラムを使用して行います。デバイスの追加や削除、設定内容の変更ができます。SigSensor, NetUIO デバイス自身の設定を変更した場合は、デバイス内の不揮発メモリ(EEPROM) に設定内容が書き込まれます。その後、デバイスを手動でリセットすることで新しい設定内容でデバイスが動作します。



（アラーム管理プログラム画面）

アラーム管理プログラムの詳細は”クライアントソフトウエア”の章中の”アラーム管理(AlarmConfig)”を参照してください。

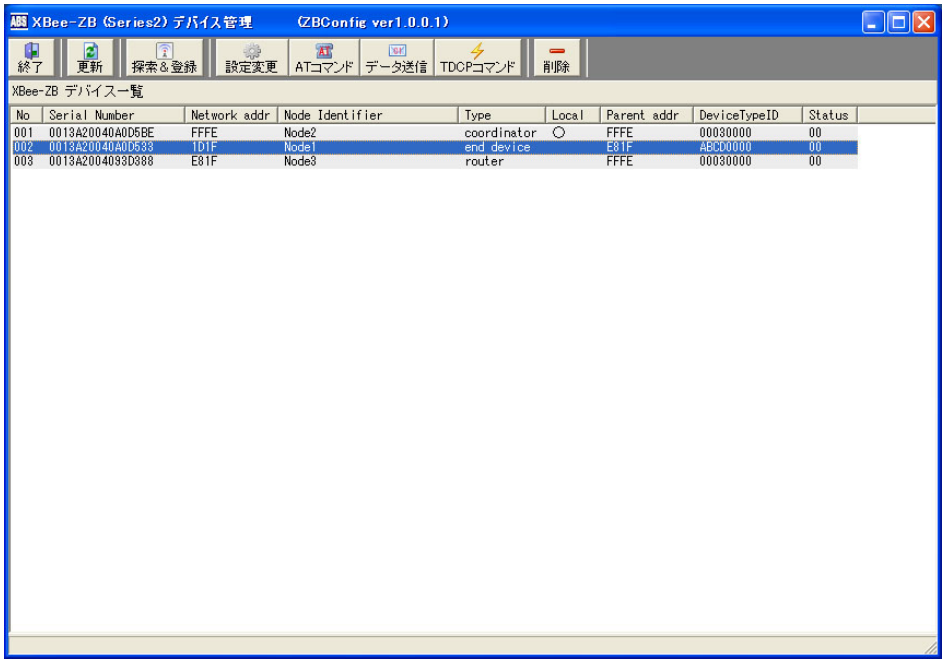
XBee 802.15.4シリーズ１デバイスの管理は、クライアントソフトを起動して、XBeeデバイス管理(XBeeConfig)プログラムを使用して行います。デバイスの探索・登録や削除、設定内容の変更ができます。また、リモート XBee デバイスの設定も同様にこのプログラムから操作することができます。



(XBeeデバイス管理プログラム画面)

XBee デバイス管理プログラムの詳細は”クライアントソフトウエア”の章中の”XBee デバイス管理(XBeeConfig)”を参照してください。

XBee-ZBデバイスも同様に、XBee-ZBデバイス管理(ZBConfig)プログラムを使用して行います。デバイスの探索・登録や削除、設定内容の変更ができます。また、リモート XBee-ZB デバイスの設定も同様にこのプログラムから操作することができます。



(XBee-ZBデバイス管理プログラム画面)

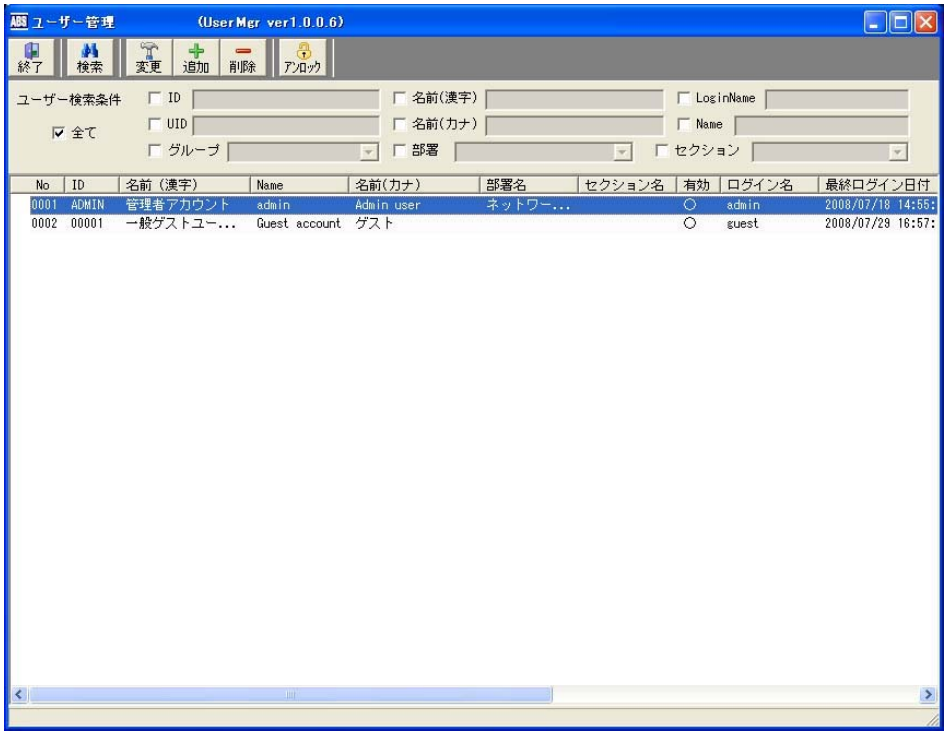
XBee-ZB デバイス管理プログラムの詳細は"クライアントソフトウエア"の章中の"XBee-ZB デバイス管理(ZBConfig)"を参照してください。

3.5 ユーザー管理

DeviceServer ではクライアントやAPI (XASDLCMD.DLL)経由で操作を行うアプリケーションにユーザー認証機能を提供しています。ユーザーの名前や属性情報を一元管理することはもちろんのこと、ログインが有効となる期限（開始と終了の両方）や同一ユーザーの同時ログイン数の制限を設定することもできます。ユーザーのログイン履歴も管理していますので、不正アクセスの監視を行うことができます。

メール経由でスクリプト実行を行うときにも、ユーザー管理で設定されたログイン認証を使用します。そのとき、予め設定されたメールアドレスのみに認証結果をメールで返信することできますので、不正な操作で発信されたメールによる操作を防止できます。

ユーザーの管理はクライアントソフトを起動して、ユーザー管理(UserMgr)プログラムを使用して行います。ユーザーの追加や削除、設定内容の変更ができます。



登録するユーザー数の制限はありません。（ただし、DeviceServerの動作しているサーバーの CPU 能力やメモリ、ディスク容量によってパフォーマンス上の制限は発生します） 同時ログイン可能なユーザーセッション数は、スタンダードライセンスで最大２、エンハンスライセンスで最大１０までです。

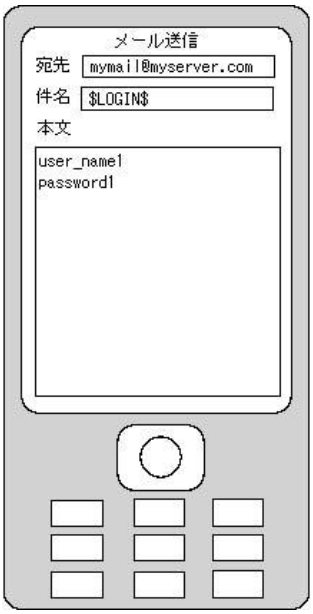
プログラムの詳細は ”クライアントソフトウエア” の章中の ”ユーザー管理(UserMgr)” を参照してください。

3.6 メールコマンド、メール通知

DeviceServer に設定された任意のユーザースクリプトを、携帯電話やPC のインターネットメール経由で実行することができます。また、ユーザースクリプトやイベントハンドラ中で任意の宛先に、スクリプト中で作成した任意の内容のインターネットメールを送信することができます。

DeviceServer ではメール受信と送信用にそれぞれ１つメールサーバー(POP3, SMTP)を設定できます。定期的にPOPメールサーバーをチェックする設定を有効にすると、電子メール経由でログイン認証を行った後に、任意のスクリプト実行を指示することができます。スクリプトパラメータもメールに記述することができますので、スクリプトの動作をメールから調整することができます。

メールから操作する場合は、ログイン認証のメールとスクリプト実行の２つのメールを送信します。
最初のメールには DeviceServer に登録済みのユーザー名とパスワードを入れて送信します。このとき、メールの件名に “$LOGIN$” を指定して DeviceServer が自動でメールを処理できるようにします。



(DeviceServerにログイン認証を行うメールを送信する)

その後、DeviceServer からログイン認証結果がメールで返ってきます。ログインに成功していた場合はセッショントークンが発行され、メール本文２行目に記載されています。セッショントークンはユーザーがDeviceServer にログインしている間、DeviceServer で管理され、コマンド実行時のユーザー認証に使用します。ログイン認証結果のメールが届くまでの時間は、DeviceServer のメールチェック間隔の設定によって変わります。

(ログイン成功した場合にDeviceServerから自動返信されたメール)

正常にログインが終了したら、次にスクリプト実行のメールを送信します。メールの１行目にログイン認証で取得したセッショントークンを記述します。セッショントークンはログイン毎に内容が変わりますので、前回認証に成功したセッショントークンを流用することはできません。２行目にスクリプト名を記入します。３行目以降は任意の数だけスクリプトパラメータを記入できます。一つの行にスクリプトパラメータのキーと値をペアで指定します。このときキーと値の間は空白文字で区切ってください。

メールの件名に “$SCRIPT$” を指定して送信します。その後スクリプトの実行結果がDeviceServer から返されてきます。

（スクリプト実行指示のメール送信画面イメージ、スクリプトは UIOUSB_DOUT_VAL でパラメータはキー値 “出力”、値が “0x0F” の１つが指定されている）

メールで実行された時に、スクリプトには、メールの宛先やリプライ先のアドレス情報をスクリプトパラメータとして自動で追加します、スクリプト中でメールを作成して送信元にリプライメールを送信することも簡単にできます。

メールコマンドの詳細は” メールからの操作”の章を参照してください。

3.7 API(DLL)経由でのコントロール

DeviceServer には API 経由でサーバーをコントロールするためのライブラリ(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、独自のクライアントプログラムを作成して、DeviceServer で管理されたログイン認証機能の利用や、ユーザースクリプトの実行ができます。

ライブラリはDeviceServer とネットワーク接続された任意の数の PC に配布して使用することができます。ライブラリではログイン認証機能やセッション管理機能も提供しています。これによって、ユーザープログラム側ではセキュリティ機能が強化されたシステムを簡単に実現できます。

ライブラリ(XASDLCMD.DLL) で提供される主な機能

- ログイン・ログアウト
- セッション更新やセッショントークンの再発行
- DeviceServer 内の共有データ（メモリ、ＤＢ）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。

APIの詳細は” API (XASDLCMD.DLL)”の章を参照してください。

3.8 Webブラウザからの操作

DeviceServerには、Webブラウザからアラームデバイスやスクリプトを操作するためのインターフェイスが備わっています。これによって、ユーザーが接続したデバイスの状態を簡単にリモートPC から確認することができます。また、スクリプト実行とそのパラメータ指定を行うための画面もありますので、ユーザーがスクリプト作成を行うときに、デバッグ目的に使用することができます。

Web ブラウザからアクセスする場合にも、通常のクライアントプロラムと同様にログイン認証が行われるため、不特定多数に公開する Web 環境においてセキュリティを保つことができます。また、Webブラウザで操作するときの DeviceServer との通信は標準の HTTP(Port80)プロトコルで行われますので、ファイアウォールやルータの設定変更を最低限にすることができます。（ただしセキュリティ確保の為、通信データ内容は暗号化されています）

Webブラウザ経由で操作可能な機能は以下になります

- アラームデバイス操作(SigSensor, NetUIO, アラームシグナルプログラム)
- UIOUSB デバイス操作
- スクリプト実行

(Web ブラウザからアラームデバイス操作パネルを表示)

Web 経由で操作を行う場合に DeviceServerをインストールしたPCに別途 HTTPサーバー(IIS[11], Apache[12]等)をインストールする必要はありません。DeviceServer自身が HTTPサーバー機能(WebProxy)を持っていますのでインストール後直ぐにクライアントからアクセスすることができます。また、既存のHTTPサーバー(IIS, Apache) を使用したい場合、例えば外部のホスティングサービス利用時にHTTPサーバーとDeviceServer(WebProxy機能)のPCを分離して設置することもできます。

Webブラウザからの操作の詳細は"WebProxy(Webブラウザからの操作)"の章を参照してください。

3.9 ログ出力

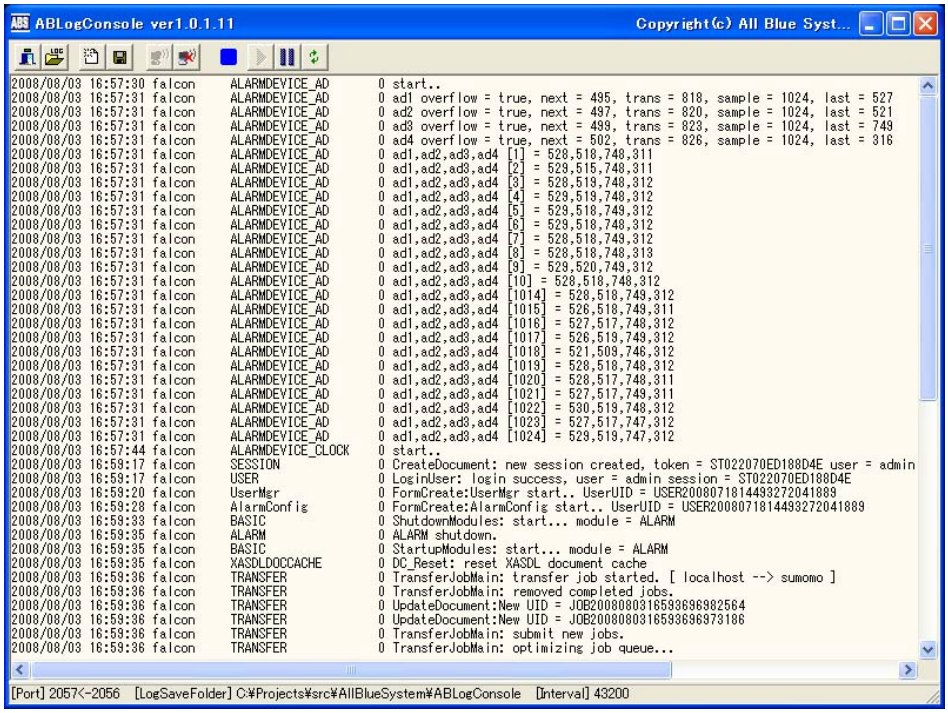
DeviceServer にはログサーバー機能を内蔵しています。サーバーで検出したエラーやイベント、クライアントアクセスの記録、ユーザーが作成したスクリプトからのログ出力を集中管理して、記録しています。

DeviceServer のログサーバーは、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

[11] Internet Information Server マイクロソフト社の Web サーバーソフトウエア
[12] http://www.apache.org/

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)
ログサーバーに記録する内容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)

ログ管理機能の詳細は”サーバーソフトウエア” の章の ”ログサービス(ABLogService)” や ”ログ管理コンソール(ABLogConsole)”の項を参照してください。

3.10 複数の DeviceServer を使用した分散処理システムの構築

複数の PC DeviceServer を設置して、それぞれの DeviceServer 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 PC 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 PC 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 DeviceServerを平行して動作させておいて、１つの PC が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。

１つの DeviceServer では、XBeeデバイスや XBee-ZB デバイスを使用したワイヤレスネットワークを構築する場合に、それぞれ(XBee, XBee-ZB)１つずつの PAN(Personal Area Network) しか構築できませんが、複数の DeviceServer を同時に使用することで多数の PAN に属しているワイヤレスデバイスを統合してコントロールすることが可能になります。

リモート PC の DeviceServer 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。

DeviceServer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 DeviceServer 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 DeviceServer 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 DeviceServer 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。

- リモートに設置した DeviceServer のパーマネントデータ(データベース領域)を直接参照または更新

リモートに設置した DeviceServer のデータベース領域を参照したり、更新することができます。詳しくは、データベース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 DeviceServer に限定して、リモートアクセスを許可します。詳しくは、"デスクトップ"クライアントソフトウエアの項を参照してください。

- DeviceServer 間の通信データを暗号化することにより高いセキュリティを維持したシステムを構築

4 インストール

DeviceServer のインストールは、インストールキット内のセットアッププログラムを実行することで行います。DeviceServerはインストール時に ABAppService (ABS-9000 DeviceServer本体) と ABLogService(ログサーバー)の2つの Windows サービスプログラムを登録します。データベース機能を内部で利用するために、Firebird DBMS[13]も同時にインストールします。全てのセットアップはウィザード形式で行われます。サービスの登録や Firebird のセッ

[13] Firebird DBMS 参照 http://www.firebirdsql.org/

トアップ等の設定項目は、全てデフォルト値で設定できますので、手動で設定内容を変更してインストールする必要はありません。

セットアッププログラム終了後、サーバー設定プログラムが起動します。正規の DeviceServer のライセンスまたはデモライセンスを入力する場合や、サーバー機能の詳細設定を行いたい場合は続けて作業を行うことができます。サーバー機能の設定やライセンス入力を後から再設定することもできます。この場合は、”終了” ボタンを押して、サーバー設定プログラムを終了して下さい。

ライセンスを入力しない場合は、DeviceServer起動後 5 時間経つと自動的にDeviceServer は停止します。Windows のコントロールパネル等から DeviceServerのサービスを手動で起動しなおすと、引き続き 5 時間使用できます。デモライセンスもしくは、正規に購入いただいたライセンスをサーバー設定プログラムから入力することで、この制限は無くなり続けて使用することができます。詳しい説明は後述の ”DeviceServerライセンスについて”を参照してください。

この章で説明するインストール手順の他に、インストールキットまたはダウンロードサイトに含まれる ″DeviceServer セットアップガイド″ マニュアルに、サービスモジュール毎の詳しいインストール手順を説明しています、併せてご覧ください。

4.1 動作環境

DeviceServer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 以外にも Windows7(64bit版)を含む最新 OS で正常に動作します。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。

Oracle データベース接続機能を使用する場合は、下記の動作環境の他にDeviceServer の動作 PC に Oracleクライアントの導入が必要です。詳しくは “Oracle接続機能” の章を参照してください。

Firebird DBMS はDeviceServerのセットアッププログラムから自動的にインストールしますので、事前にインストールしておく必要はありません。すでに Firebird DBMS が PC にインストール済みの場合はバージョンが動作環境のバージョンと一致していることを確認してください。もし、バージョンが一致しない場合は既存の Firebird をアンインストールした後に、DeviceServer のセットアッププログラムを実行してください。

DeviceServerの動作環境項目	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
USB2.0 ポート	1つ以上（UIOUSB デバイスを接続する場合のみ）
Firebird DBMS	2.1.1（セットアッププログラム実行中に、自動でインストールします）

DeviceServer を動作させるために、下記のネットワークリソース(プロトコル、ポート)を使用します。インストールする PC で、他の目的に使用されていないことを確認してください。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、下記のネットワークリソースを使用可能な様に設定変更が必要な場合があります。詳しくはご使用中のファイアウォールプログラムまたはセキュリティソフトのマニュアル等を参照ください。

SigSensor, NetUIO デバイスを使用して、DeviceServer へのイベント送信機能を利用する場合は、DeviceServer が動作する PC には必ず固定 IPアドレスを使用してください。SigSensor, NetUIOデバイスにはホスト名解決の機能が無いため、送信先アドレスには固定した IP アドレスが必要となるためです。

DeviceServer がサーバーPC で使用するネットワークリソースは以下になります。

DeviceServerプログラム・機能	プロトコル	ポート番号(名称)
DeviceServer コマンド処理	TCP	27101
DeviceServer イベント受信・応答	TCP	27102
DeviceServer アラームデバイス操作 AlarmSignal.exe 使用時は下記（注意1)を参照の事	TCP	27103
DeviceServer WebProxy機能（注意2)を参照の事	TCP	80（http)
ログメッセージ送受信	UDP	2056 2057
Firebird DBMS	TCP	3050（gds_db)

DeviceServer のクライアントプログラムがPC で使用するネットワークリソースは以下になります。

プログラム・機能	プロトコル	ポート番号(名称)
DeviceServer コマンド処理	TCP	27101
DeviceServer イベント受信・応答	TCP	27102
Web ブラウザからDeviceServer にアクセスする	TCP	80（http)

DeviceServer にネットワーク接続したアラームデバイス(SigSensor, NetUIO, アラームシグナルプログラム）が使用するネットワークリソースは以下になります。

プログラム・機能	プロトコル	ポート番号(名称)
DeviceServer イベント送信	TCP	27102
DeviceServer アラームデバイス操作 AlarmSignal.exe 使用時は下記（注意１)を参照の事	TCP	27103

（注意1）アラームシグナル使用時の注意

AlarmSignal.exe はソフトウエアで実現したアラームデバイスです。これは複数のインスタンスを同一CPU で同時に、別ポート番号で起動することができます（デフォルトは TCP/27103）。この場合は、起動しているポート番号分のネットワークポートリソースを使用していることになります。

（注意2）WebProxy使用時の注意

既存のHTTPサーバー(IIS, Apache) を使用したい場合、例えば外部のホスティングサービス利用時にHTTPサーバーとDeviceServer の動作 PC を分離して設置することもできます。詳しくは "WebProxy(Webブラウザからの操作)"の章を参照してください。

4.2 セットアッププログラム実行

Windows のシステム管理者権限のユーザー(Administrator等)でセットアップを行います。Windows の一般ユーザーではセットアップできませんので注意してください。

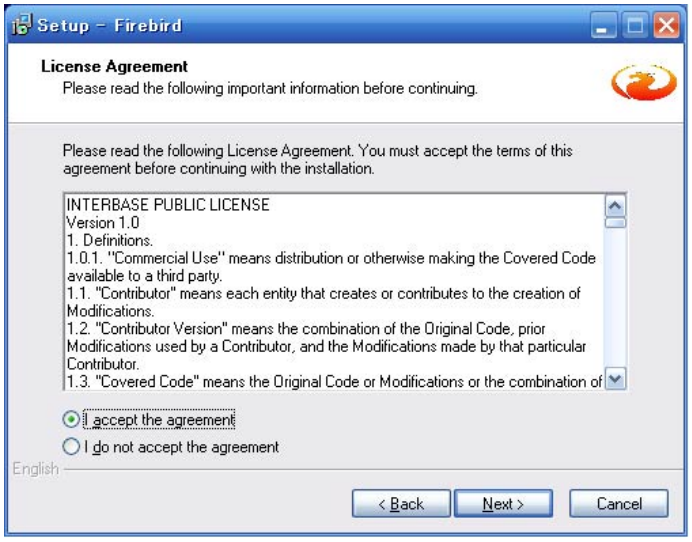
インストールキットは ZIP 形式で１つのファイルにまとめられていますので、適当なディレクトリにファイルを展開してください。WindowsXP, Windows2003, Windows7 の場合にはファイルを右クリックして “全て展開” メニューでファイルを展開することができます。ファイルを展開したら、"setup.exe" をダブルクリックしてインストーラプログラムを起動します。

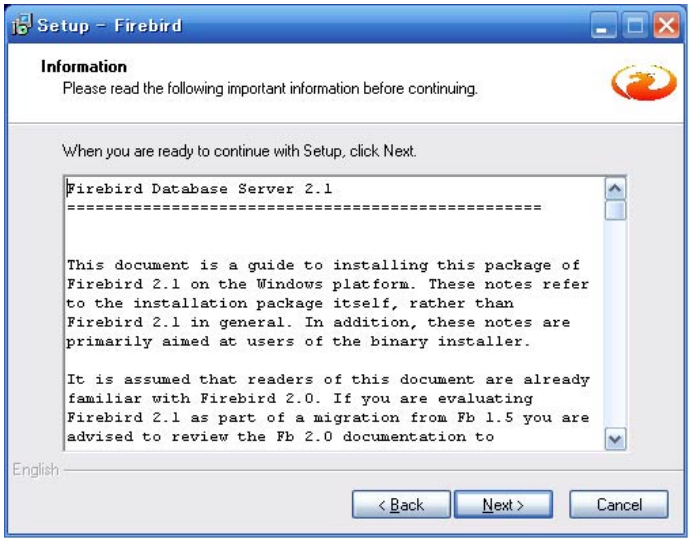
セットアップを行う PC に Firebird DBMS がインストールされていない場合は、最初にインストールを促す画面を表示します。"OK" を押して Firebird DBMS のインストールを行います。既に Firebird DBMS がインストール済みの場合はこの画面は表示されないで、DeviceServer のインストール画面表示までスキップします。その場合は次の Firebird DBMS のインストールの説明部分をスキップしてください。

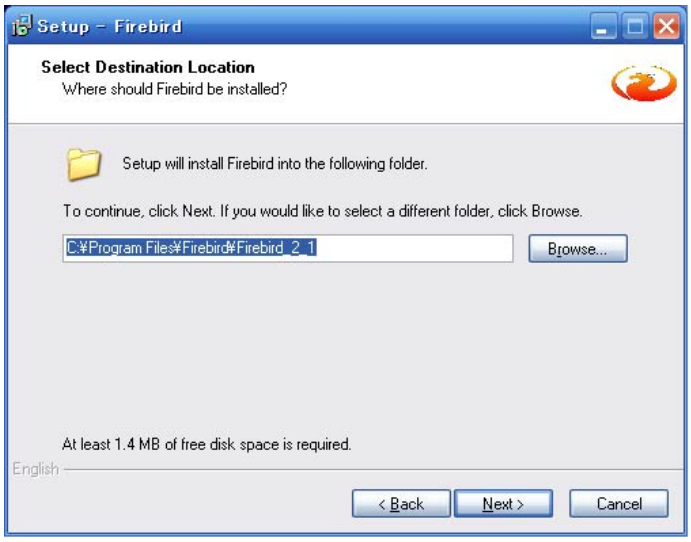
4.2.1 Firebird DBMS インストール

最初に言語選択画面を表示しますので “English” を選択して “OK” を押してください。

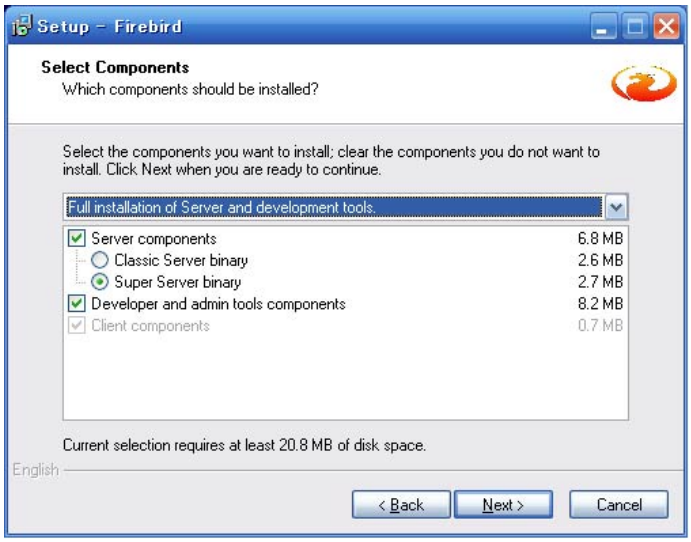
以降、全ての設定項目はデフォルト項目のままでインストールしますのでそのまま”OK”を押すことでセットアップは終了します。



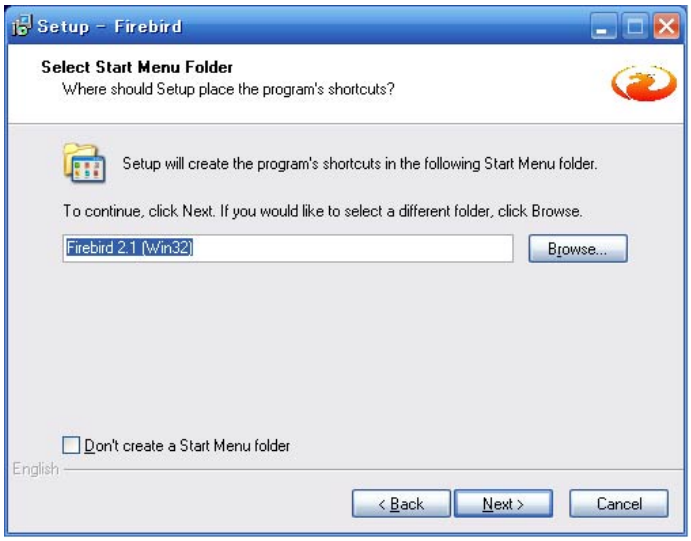


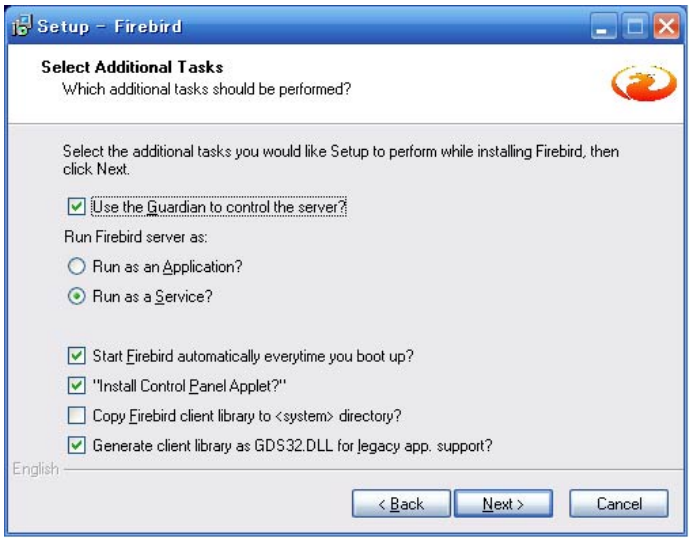


インストール先のフォルダはデフォルトのままにしてください

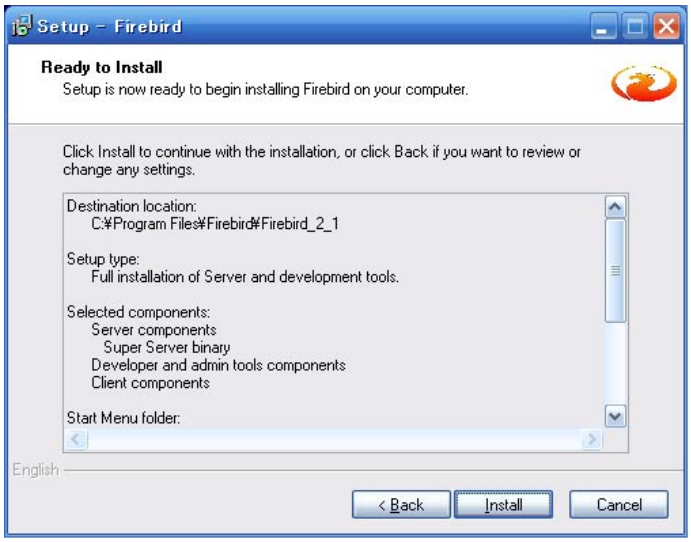


コンポーネント選択はデフォルトの “Full installation of Server and development tools.” を選択してください

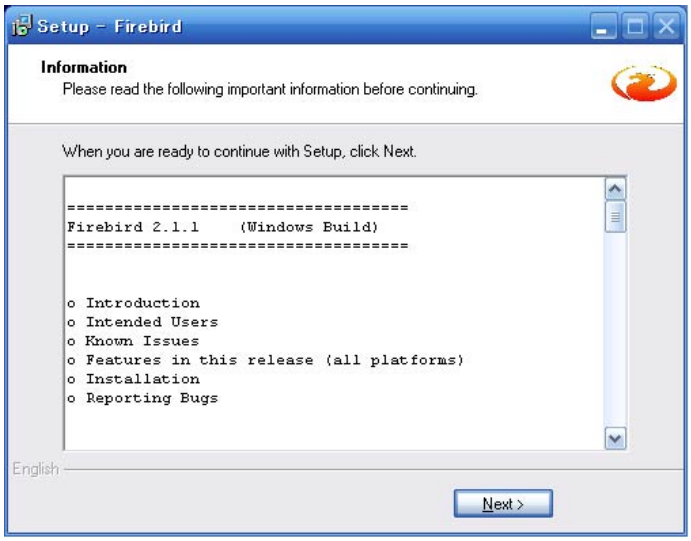




その他の設定項目画面もデフォルトのままにして、”Next” を押してください



インストール準備が完了したので、”Install” を押します



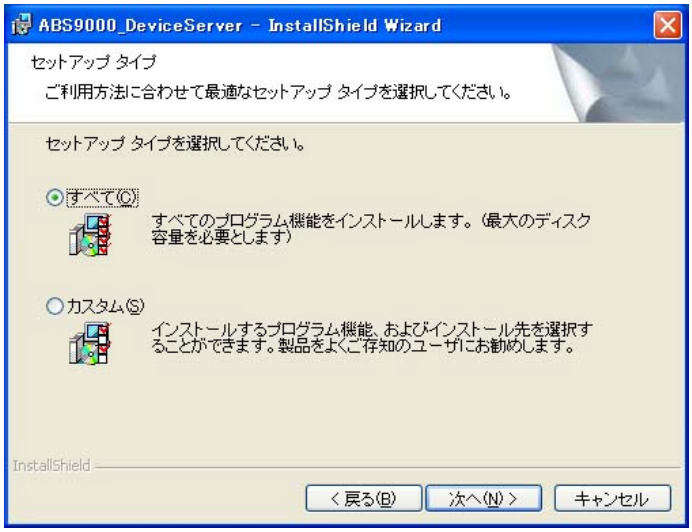
インストールが終了の画面です。”Start Firebird now?” だけにチェックをして “Finish” を押してください

4.2.2 DeviceServer インストール

Firebird のインストールが終了すると、DeviceServer のインストール画面を表示します。表示内容を確認の上セットアップを続行してください。

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 DeviceServer を使用することはできません。

DeviceServer 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。

DeviceServer のセットアッププログラムが正常に終了すると、続けてサーバー設定プログラムが起動します。

ライセンス入力や詳細なサーバー機能設定を行う場合には続けて設定できます。” 初期設定”の章と”サーバーソフトウエア”の章中の “サーバー設定”の項を参照してください。

設定を後で行いたい場合は、 “終了” ボタンを押してください。サーバー設定は何時でも実行することができます。

4.3 クライアントインストール(別のPCから利用する場合のみ)

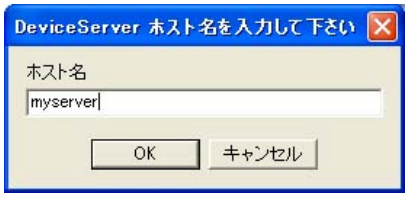
DeviceServer のセットアッププログラムが終了すると、サーバーPC では常にクライアントソフトも使用可能な状態になっています。サーバーPC 以外の PC からクライアントアプリケーションを利用する場合は予め必要なソフトをコピーしておく必要があります。Webブラウザ経由でアクセスする場合には、必要なプログラムなどはありませんので直ぐにアクセスできます。

4.3.1 DeviceServerのクライアントソフトウエア(デスクトップ、ユーザー管理、アラーム管理など)を使う場合

サーバー PC にある以下の１つのクライアントプログラムを任意の PC のローカルディスクにコピーを行ってください。

コピーを行うファイル（サーバーPC のファイル名）	クライアント PC のコピーする場所
C:¥Program Files¥AllBlueSystem¥ABLauncher.exe または C:¥Program Files (x86)¥AllBlueSystem¥ABLauncher.exe  ABLauncher....	（任意の場所にコピーしてください）

ABLauncher.exe プログラムをクライアントPC で最初に実行するとDeviceServerホスト名の入力ダイアログを表示します。ABLauncher.exe は最後に接続したサーバーに接続を試みて、もしエラーが発生した場合は同様にホスト名を入力する画面になります。



ホスト名に DeviceServer をセットアップしたサーバーのホスト名もしくは IP アドレスを入れて “OK” を押してください。接続に成功すると、ABLauncher.exe プログラムが必要に応じて最新版のクライアントプログラムを DeviceServer から自動的にダウンロードして実行します。

このように、クライアントPC では必要に応じてサーバーPC から自動的にダウンロードして実行する仕組みがありますので、ABLauncher.exe ファイルにみを最初に一度手動でコピーしておくだけで、全てのクライアントプログラムを利用できます。

ABLauncher プログラムとその他クライアントプログラムの詳しい説明は “クライアントソフトウエア” の章を参照してください。

4.3.2 ユーザーアプリケーションを利用する場合(APIライブラリを使用)

DeviceServer のに付属するAPIライブラリ（XASDLCMD.DLL）を使用して作成されるユーザープログラムを、開発もしくは実行する場合は、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。

コピーを行うファイル（サーバーPC のファイル名）	クライアント PC のコピーする場所
C:¥Program Files¥AllBlueSystem¥Demo¥Excel¥SystemFolder¥XASDLCMD.dll または C:¥Program Files (x86)¥AllBlueSystem¥Demo¥Excel¥SystemFolder¥XASDLCMD.dl  	Windows のシステムディレクトリ WindowsXP,Windows2003 の場合は ”C:¥Windows¥System32¥” になります Windows7(64bit) の場合には ”C:¥Windows¥SysWOW64” になります

4.3.3 Webブラウザ経由で使用する場合

DeviceServer が動作しているPC とネットワーク接続されたクライアントPCの Webブラウザを使用して、デバイスの操作を行う場合には、Webブラウザで Adobe Flash Player（バージョン9 以上)が使用可能になっている必要があります。使用可能になっていない場合には、下記の Adobe ホームページから最新版の Flash Player をダウンロードしてインストールしてください。

Adobe 社ホームページ　http://www.adobe.com/jp/

DeviceServer インストール直後は、WebProxy 機能は無効になっていますので、”サーバー設定”プログラムをプログラムメニューから選択・実行して、WebProxy 機能を有効にしてください。

WebProxy 機能の詳細は “WebProxy(Webブラウザからの操作)” の章を参照してください。

5 初期設定

DeviceServer をインストール直後は、初期設定状態で起動しています。サーバー設定プログラムを使用してライセンスの設定や、その他の機能を設定できます。ここでは、ライセンスと最低限の設定について説明します。詳しいサーバー機能の設定項目と内容については “サーバーソフトウエア” の章内 ”サーバー設定” の項を参照してください。

5.1 DeviceServerライセンスについて

DeviceServer のライセンスには下記の種類があります。

DeviceServerライセンス種別	説明
スタンダードライセンス	●最大同時ログイン数 2 セッション ●ネットワーク接続可能なアラームデバイス(SigSensor, NetUIO, アラームシグナル)は最大２台 ●シリアルポート経由で接続する Arduinoや計測器等は最大２台 ●XBee 802.15.4 シリーズ1 デバイス管理・操作機能(同一PAN 内で運用可能なリモートデバイスは制限なし、ただし処理スピードはイベントハンドラ同時実行数に依存します) ●XBee-ZB シリーズ2 デバイス管理・操作機能(同一PAN 内で運用可能なリモートデバイスは制限なし、ただし処理スピードはイベントハンドラ同時実行数に依存します) ●TWEワイヤレスデバイス操作機能 ●MQTT クライアント接続機能。設定可能なエンドポイントは最大２個 ●スクリプト・イベントハンドラ同時実行数：16 ●内部データベースセッションプール接続数：10
エンハンスライセンス	●最大同時ログイン数 10 セッション ●ネットワーク接続可能なアラームデバイス(SigSensor, NetUIO,アラームシグナル)は最大１０台 ●シリアルポート経由で接続する Arduinoや計測器等は最大１０台 XBee 802.15.4 シリーズ1 デバイス管理・操作機能(同一PAN 内で運用可能なリモートデバイスは制限なし) ●XBee-ZB シリーズ2 デバイス管理・操作機能(同一PAN 内で運用可能なリモートデバイスは制限なし) ●TWEワイヤレスデバイス操作機能 ●MQTT クライアント接続機能。設定可能なエンドポイントは制限なし ●Oracleデータベース接続機能(Oracle10g) ●スクリプト・イベントハンドラ同時実行数：任意の値に設定可能 ●内部データベースセッションプール接続数：任意の値に設定可能

ライセンスまたはデモライセンス（後述）を入力しないでDeviceServer を利用することもできますが、この場合は、

同時ログイン数が 1 で連続動作は 5 時間までに制限されていますのでご注意下さい。後述のデモライセンスにはこれらの制限事項はありませんので、試用や評価にはこちらを利用してください。

正規ライセンスをご購入される方は、メールにて注文をお受けしております。
後述の”デモライセンスを利用する” で説明してあります “デモライセンス.txt” ファイル内にライセンス購入手続きについての詳しい説明が記載していますので参照下さい。もしライセンスについて不明な点がございましたら、下記までメールにてご質問下さい。

Oracleデータベース接続機能使用時には DeviceServer のエンハンスライセンスが必要になります。
また、DeviceServer の動作する PC に Oracleクライアントを導入する必要があります。そのときには Oracleデータベースのクライアントライセンスが必要となります。オールブルーシステムの DeviceServer エンハンスライセンスには Oracle データベースのクライアントライセンスは含まれません。Oracleクライアントライセンスについては使用中の Oracle データベースのサービスプロバイダやデータベース管理者にお問い合わせ下さい。

ライセンスについての問合せ先： contact@allbluesystem.com

5.2 デモライセンスを利用する

DeviceServer をインストールするとインストールフォルダのEtcフォルダ “C:¥Program Files¥AllBlueSystem¥Etc” または “C:¥Program Files (x86)¥AllBlueSystem¥Etc”に “デモライセンス.txt” ファイルが保管されています。この中に記載されているライセンスコードを使用すると、有効期限付きで全機能が使用可能になりますので、試用や評価などにご利用下さい。

また、デモライセンスの有効期限が切れている古いインストールキットを使用した場合や、評価の為に有効期限を延長したい場合などは下記までメールでご連絡下さい。

ライセンスについての問合せ先： contact@allbluesystem.com

5.3 サーバーの初期設定

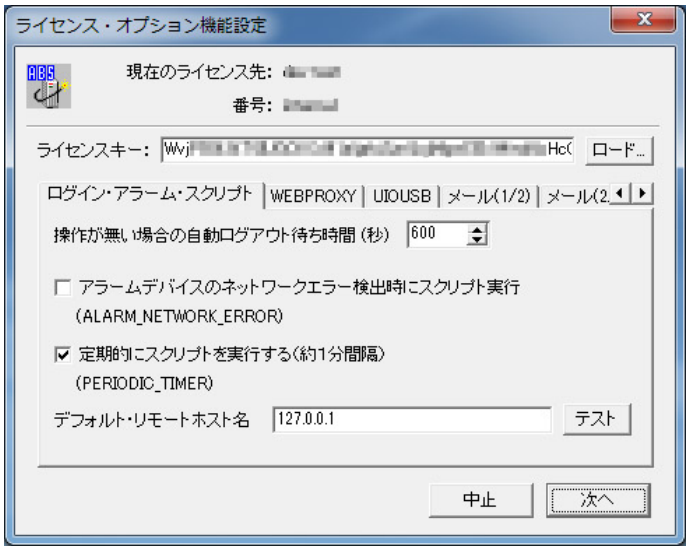
DeviceServer セットアップ終了後や、プログラムメニューから “ALL BLUE SYSYEM” -> “サーバー設定”を選択・実行して、最初の画面で”次へ”を選択するとサーバーの詳細設定を行う画面を表示します。ここでは、動作確認に必要な最低限の機能とライセンスの入力についてのみ説明します。全ての機能についての説明は、“サーバーソフトウエア” 章の ”サーバー設定” の項を参照してください。

5.3.1 ライセンス入力

デモライセンスもしくは、正規ライセンスをライセンスキー入力欄に入力します。テキストエリアは右にスクロールしますので、必ず全文字列を正確にいれてください。間違いを防ぐためにコピー＆ペーストで入力することをお勧めします。

キット製品などで DeviceServer のライセンスも同時に提供されている場合には、キット製品に付属するインストー

ルメディア中にあるライセンスファイル(license.xml) を読み込んでライセンスキーを設定します。”ロード”ボタンを押して、メディア中のライセンスファイル(license.xml) を開いて下さい。ライセンスキーがテキストエリアに入ります。



ライセンスキーを入力したら、”次へ”を押してください。サービスが再起動されて、入力したライセンスが有効になります。

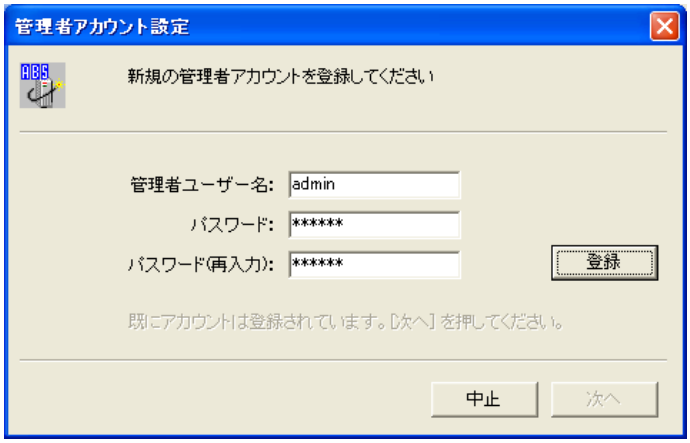
設定後のライセンス情報と現在実行中の DeviceServerの機能（サービスモジュール）一覧を表示します。

5.3.2 管理者アカウント設定

DeviceServer の管理者アカウント（管理者特権を持ったユーザー）を作成します。DeviceServer にユーザーが１つ以上既に登録済みの場合はこの画面は表示しません。インストール直後にはユーザーは一つも登録されていない状態なのでこの画面から最初のユーザーを登録します。ここで登録するユーザーは管理者として登録されて、他のユーザー登録や削除等を行うことができます。

管理者パスワードを忘れないように注意してください

ここで登録した管理者パスワードは忘れないように注意してください。他の管理者特権(AdminTask)を持ったユーザーを作成する前に忘れてしまうと、ユーザ管理が一切できなくなります。この場合には手動でアプリケーションデータを削除した後、DeviceServer を再インストールする必要があります。



管理者ユーザー名(ログイン名)とパスワードを入力して、”登録”ボタンを押してください。ここで入力したパスワードはクライアントソフトウエア(ABDesktop)にログイン後にいつでも変更できます。

サーバーの基本的な設定はこれまでの操作で終了です。もし、UIOUSB デバイスやメールコマンドの機能を使用する場合は、”サーバー設定” プログラムを起動して、設定を行う必要があります。詳しい内容は、“サーバーソフトウエア” 章の ”サーバー設定” の項を参照してください。

5.4 スクリプト編集用エディタを用意する

サーバー側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。

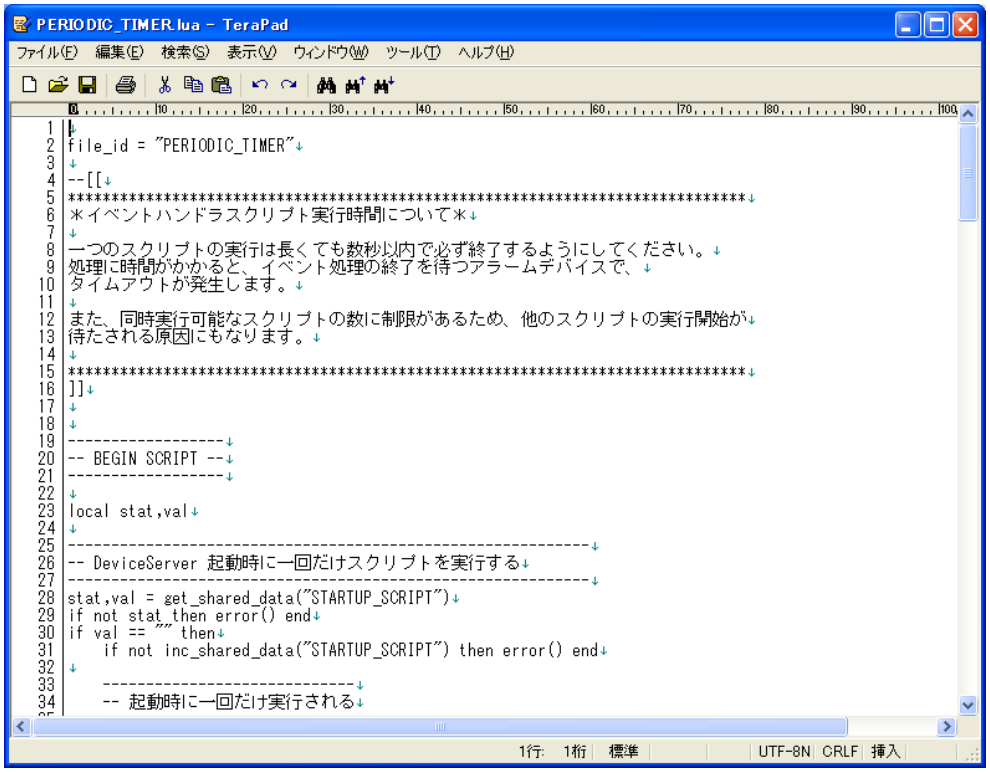
スクリプト中で日本語を使用する

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。

このため、UTF-8 文字コードで編集ができるエディタソフトを準備してください。

Windows付属のワードパッドやメモ帳ではこの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/



TeraPad で DeviceServer 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。

アプリケーション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。

6 アンインストール・再インストール(アップデート)

DeviceServer のアンインストールはWindows コントロールパネルの”プログラムの追加と削除”から行います。また、DeviceServer で使用した Firebird DBMS の削除も同様に ”プログラムの追加と削除”から行います。

アンインストーラでは、DeviceServer のサービスプログラムやサーバー側にあるクライアントプログラム、ライブラリ等を自動的に削除しますが、インストール後に更新された設定ファイルやアプリケーションデータ（ライセンス情報、登録されたユーザーデータ、アラーム設定等） は削除しません。DeviceServer の再インストールを行う場合には、アンインストール直後に、インストールを行うことで、全ての設定を引き継いでセットアップが完了しますので、サーバー設定を省略することができます。全てのデータを PC から削除したい場合や、初期化を行いたい場合は手動でファイルを削除します。

6.1 DeviceServerアンインストール

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_DeviceServer” を選択して “削除” を押します。

サービスが自動的に停止されアンインストールが行われます。

6.2 Firebirdアンインストール

Firebird DBMS を別のプログラムで使用中の場合や、DeviceServer の再インストールを行う予定の場合には、Firebird のアンインストールの必要はありません。Firebird DBMS を削除する場合は、システム管理者で 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 “Firebird xxxx” を選択して “削除” を押します。(“xxxx” にはFirebird のバージョン番号が入ります)

サービスが自動的に停止されアンインストールが行われます。

6.3 アプリケーションデータ削除

DeviceServer を再インストールする場合は、アプリケーションデータを削除する必要はありません。

DeviceServer で作成・更新されたアプリケーションデータを全て削除するには、下記のフォルダごと手動でゴミ箱に入れてください。

削除対象のDeviceServer のアプリケーションデータフォルダ
C:¥Program Files¥AllBlueSystem (”C:¥Program Files” に移動した後、AllBlueSystem フォルダごと削除する) Windows7(64bit)の場合、 C:¥Program Files (x86)¥AllBlueSystem (”C:¥Program Files (x86)” に移動した後、AllBlueSystem フォルダごと削除する)

6.4 クライアントプログラム削除(クライアントPC)

クライアントプログラムが動作したPC からファイルを削除するために、手動でセットアップを行った下記の２つの

ファイルと自動でダウンロードされたキャッシュファイルを手動で削除します。(サーバーPC 上のABLauncher.exe、XASDLCMD.DLLファイルは DeviceServer のアンインストーラで自動削除しますので、削除の必要はありません)

クライアントPC上の削除対象のファイル・フォルダ
ABLauncher.exe (インストール時にクライアントPCにコピーを行ったファイルを削除してください)
C:¥Windows¥System32¥XASDLCMD.DLL (C:¥Windows¥System32に移動した後、XASDLCMD.DLLファイルを削除する) Windows7(64bit)版の場合には、 C:¥Windows¥SysWOW64¥XASDLCMD.DLL (C:¥Windows¥SysWOW64に移動した後、XASDLCMD.DLLファイルを削除する)
C:¥Documents and Settings¥All Users¥Application Data¥AllBlueSystem (C:¥Documents and Settings¥All Users¥Application Data に移動した後、AllBlueSystem フォルダごと削除する) Windows7(64bit)版の場合には、 C:¥Users¥All Users¥AllBlueSystem (C:¥Users¥All Users に移動した後、AllBlueSystem フォルダごと削除する)

7 動作確認と簡単な使い方

この章では、ステップバイステップで簡単な使用方法を説明します。SigSensor, NetUIO, UIOUSB 等の外部デバイスを使用せずにアラーム機能を試せるようになっていますので、DeviceServerのインストール後の動作確認にも使用できます。 この章で説明している操作を行う前に、DeviceServer のセットアップが完了していることが必要です。また、DeviceServerをインストールしたPCをクライアントPC として使用しますので、別途クライアントPC を準備する必要はありません。

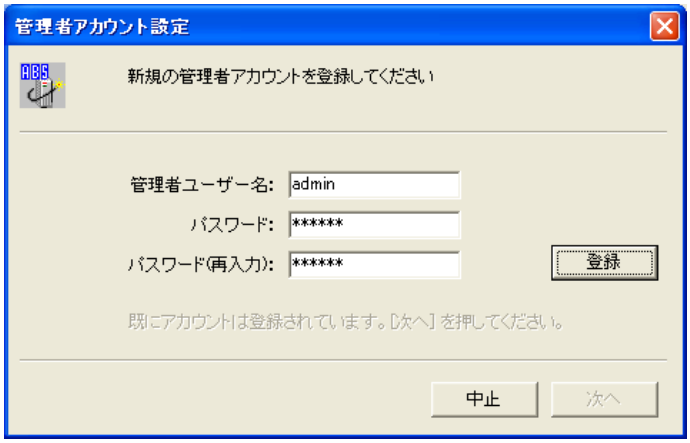
7.1 管理者アカウント登録

DeviceServer のインストール時に管理者アカウントを既に登録済みの場合は、次のステップに進んでください。

管理者アカウント登録は、”サーバー設定”プログラムから行います。プログラムメニューから “ALL BLUE SYSYEM” -> ”サーバー設定” を選択して “次へ” を押していくと、”管理者アカウント設定” 画面を表示します。

ユーザー名とパスワードを設定して “登録”ボタンを押します。(ユーザー名とパスワードに使用可能な文字はアルファベットと数字で64 文字以内にしてください)

既に、管理者アカウントや、他のユーザーアカウントが DeviceServer に登録済みの場合は、画面はインアクティブな表示になって登録できなくなっています。



7.2 一般ユーザーアカウント登録

アラームデバイスやスクリプト操作を行う、一般ユーザーアカウントを登録します。

プログラムメニューから、 “ALL BLUE SYSTEM” -> “クライアント起動” を選択・実行します。

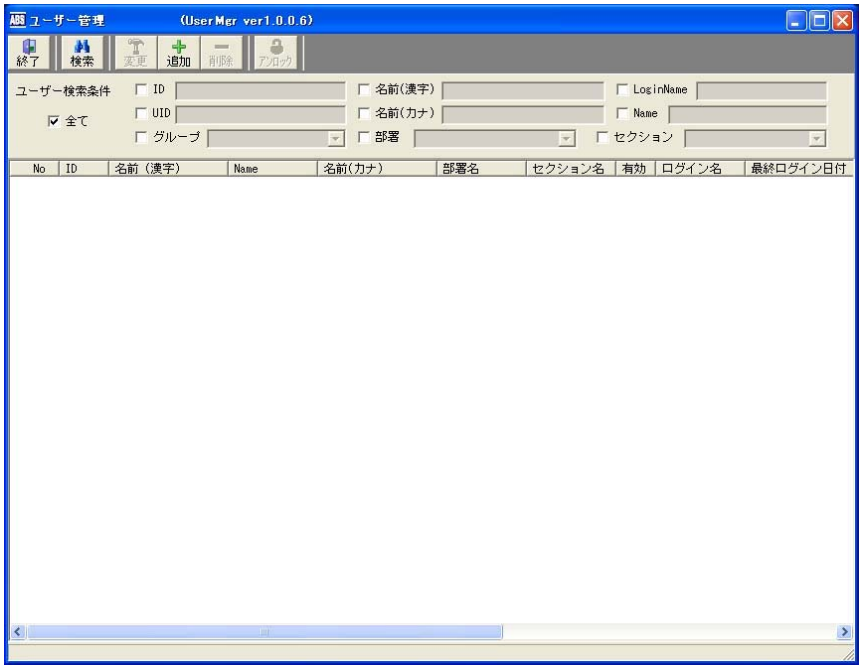
もし、最初にクライアントを起動した場合には、DeviceServer ホスト名の入力ダイアログを表示します。その場合は、ホスト名に “localhost” を入れて “OK” を押してください。

デスクトッププログラムが自動的に起動されてログイン画面を表示します。ここで、先に登録した管理者アカウントのユーザー名とパスワードを入力します。ホスト名は “localhost” のままにしてください。

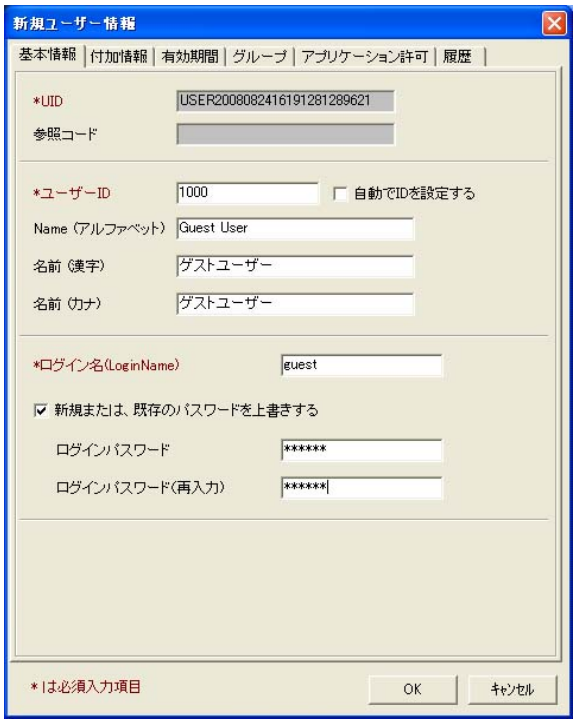
ログインが成功して、デスクトッププログラムが起動すると、ログインを行ったユーザー（ここでは管理者アカウント）で使用可能なツールボタンを表示しています。

ユーザー登録を行いますので、”ユーザー” ツールボタンを押します。

ユーザー管理プログラムが起動すると、デスクトッププログラムの下部にユーザー管理プログラムのフォームを表示します。ユーザー管理プログラム内にも幾つかのツールボタンがあり、ユーザー登録や変更操作を行います。ここでは新規の登録をするために、”追加”ボタンを押します。



確認ダイアログを表示しますので、”OK” を押します。



新規に作成するユーザーアカウントの情報を入力します。ここでは、下記の値で入力を行います。

- ユーザーID　1000
- Name　Guest User
- 名前(漢字)　ゲストユーザー
- 名前(カナ)　ゲストユーザー
- ログイン名　guest
- ログインパスワード　〈任意の文字列でアルファベットまたは数字で64 文字以内〉

また、登録フォームの “アプリケーション許可”タブを選択して下記の値もチェックして有効にします。

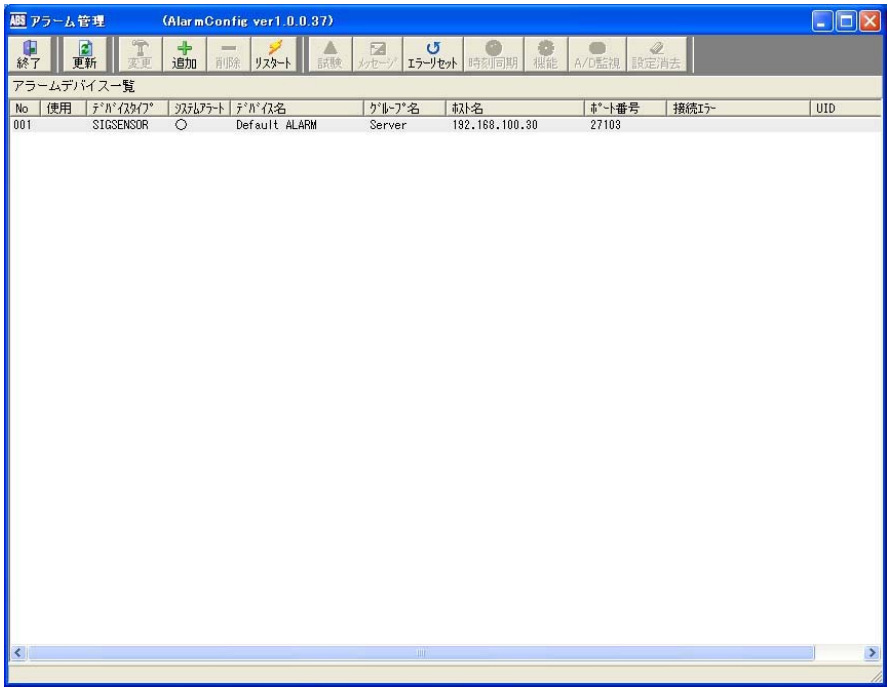
- AllowLogin　有効にする
- WebLogin　有効にする

ここで、”OK” を押すと DeviceServer にユーザーアカウントを登録します。新規ユーザーを登録することにより、このユーザーアカウントでログインして、アラーム操作やスクリプト実行ができるようになりました。次に、アラームデバイスを登録しますが、アラームデバイスの追加などサーバー構成を変更するためには引き続き管理者アカウントのまま行う必要があります。

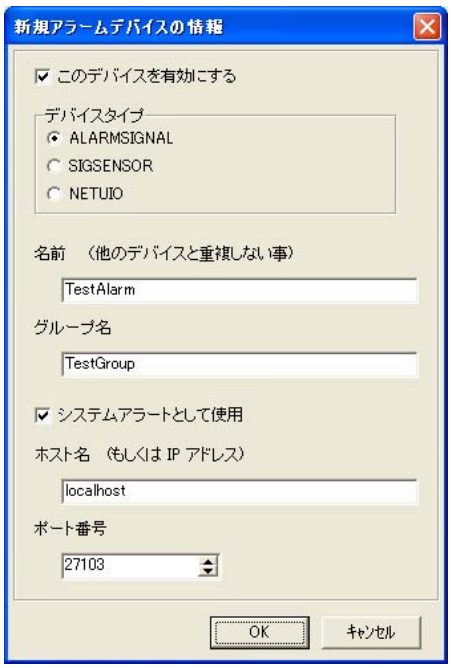
7.3　アラームデバイス登録

ユーザー登録に続いて、アラームデバイスを登録します。デスクトッププログラムの “アラーム” ツールボタンを押します。ユーザー管理プログラムが起動すると、デスクトッププログラムの下にメインフォームを表示します。アラ

ーム管理プログラム内にもユーザー管理プログラムと同様に幾つかのツールボタンがあり、アラームデバイス登録や変更操作を行います。このとき、既に “Default Alarm” が登録されています。これは SigSensor デバイスを新規に登録するときのデフォルトデバイスですが、今回は使用しません。新規のデバイス登録をしますので、”追加”ボタンを押します。



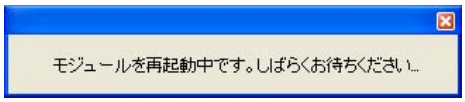
確認ダイアログを表示しますので、”OK” を押します。



新規に作成するアラームデバイスの情報を入力します。ここでは、下記の値で入力を行います。

- デバイスを有効にする　チェックを有効にする
- デバイスタイプ　ALARMSIGNAL
- 名前　TestAlarm
- グループ　TestGroup
- システムアラートとして使用　チェックを有効にする
- ホスト名　localhost
- ポート番号　27103

“OK” を押して、登録を行います。アラームデバイスの構成を変更すると、DeviceServer はアラーム関連の設定を初期化するために、アラームモジュール機能をリスタートします。再起動中のダイアログが表示された後にアラーム管理プログラムのデバイス一覧を更新します。



次に、アラームシグナルプログラムを起動します。起動を簡単にするために、ショートカットをWindows のデスクトップに作成してから起動します。下記のプログラムのショートカットをデスクトップに作成してください。
C:¥Program Files¥AllBlueSystem または C:¥Program Files (x86)¥AllBlueSystem に移動した後、マウスの右ボタンで下記のファイルを選択したまま、デスクトップにドラッグすることで、簡単にショートカットが作成できます。

アラームシグナルプログラム(AlarmSignal.exe)のインストール先
C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe または C:¥Program Files (x86)¥AllBlueSystem¥AlarmSignal.exe 

次に、作成したショートカットの下記の項目を変更します。ショートカットファイルを右クリックして”プロパティ”を選択して内容を変更してください。起動パラメータを 2つ追加しています。パラメータの間はスペースで区切ってください。

- リンク先(変更前)　"C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe"
  または "C:¥Program Files (x86)¥AllBlueSystem¥AlarmSignal.exe"
- リンク先(変更後)　"C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe" 27103 サンプルアラーム
  または
  "C:¥Program Files (x86)¥AllBlueSystem¥AlarmSignal.exe" 27103 サンプルアラーム

ショートカットの変更が終了したら、ダブルクリックして起動してください。下記の画面が表示され、先にアラーム

管理プログラムで登録したアラーム(TestAlarm) に対応するデバイスが操作可能な状態になっています。

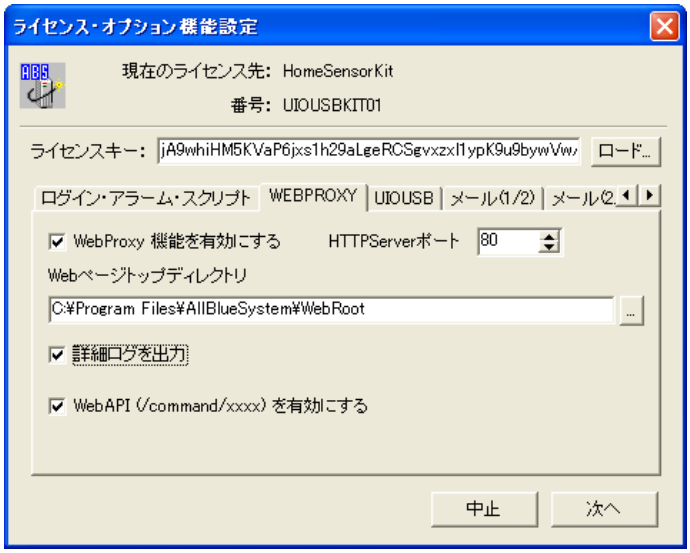
起動パラメータ(ポート番号とデバイス名称) を付加したショートカットからアラームシグナルプログラムを起動した場合には下記の様に自動的に ONLINE 状態になります。画面左上の丸印が白くなっているランプ部分が ONLINE 状態を示しています。また、このランプ部分をマウスでクリックすると ONLINE <-> OFFLINE の切り替えができます。

DeviceServer のアラーム管理プログラムのデバイス一覧で、アラーム(TestAlarm) を選択して “試験”ツールボタンを押すと、アラームシグナルプログラムのランプ点灯とブザー音出力を確認することができます。

ここまでの操作でユーザーとアラームの登録が終了しましたので、DeviceServer のクライアントプログラムを終了します。デスクトッププログラムの “終了” ツールボタンを押すと、起動中のユーザー管理プログラムとアラーム管理プログラムと共に全て終了してログアウト状態になります。アラーム管理プログラムの”終了” を押すとデスクトッププログラムとユーザー管理プログラムは起動したままで、アラーム管理プログラムのみが終了します。ユーザー管理プログラムの場合も同様です。

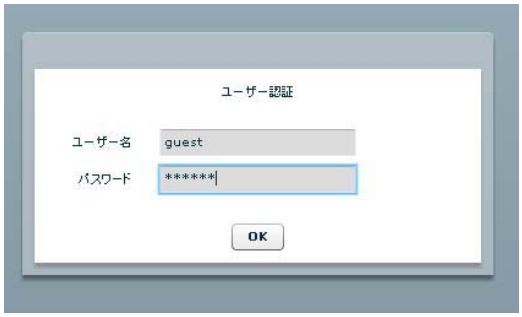
7.4 Webブラウザからのアラーム操作

Webブラウザから、先ほど登録したアラームを操作します。予め、DeviceServer のサーバー設定で”WebProxy”機能を有効にしておきます。プログラムメニューから “ALL BLUE SYSTEM” -> “サーバー設定” を選択して、ライセンスオプション機能設定画面まで、”次へ” を押して進めます。”WEB PROXY” タブを選択して、”WebProxy機能を有効にする” にチェックをつけます。サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。



Webブラウザ(インターネットエキスプローラ等)を起動して、http://localhost/remote/AlarmControl.html をアクセスするか、プログラムメニューから “ALL BLUE SYSTEM” -> “アラーム操作”を選択・実行します。

ログイン画面が表示されますので、先に登録したユーザー（guest）のログイン名とパスワードを入力してログインします。(Webブラウザには、予め Adobe Flash Player（バージョンver9 以上)がセットアップされている必要があります)



ログインに成功するとアラームデバイス(先に登録した"TestAlarm")が表示されているのが確認できます。
ここで、チェックボタンを操作するとアラームシグナルプログラムのランプを点灯・点滅させることや、ブザー出力を行えます。アラームシグナルプログラムの右にある “RESET” ボタンエリアをクリックすると全てのシグナル出力が強制的に OFF になります。このとき、Webブラウザ上のアラーム管理画面の"更新"ボタンを押すことでチェックボタンもクリアされるのが確認できます。

Webブラウザからのアラーム管理を終了する場合は、"終了"ボタンを押してログアウトします。ログアウトを行わずにWebブラウザを終了すると DeviceServer にログインセッションが一時的に残ってしまいます。この場合、次にログインを行うときに、同一ユーザーの同時ログイン制限(ユーザー管理プログラムで設定した場合)やライセンスで決められた同時ログイン数を超えた場合に、ログインできなくなりますので注意してください。初期設定では 10 分以上操作が行われていないセッションを自動的に強制削除しますので、それ以降はログインできるようになります。または、プログラムメニューから “ALL BLUE SYSTEM” -> “セッション管理” を選択・実行して、強制的にログインセッションを終了させることもできます。

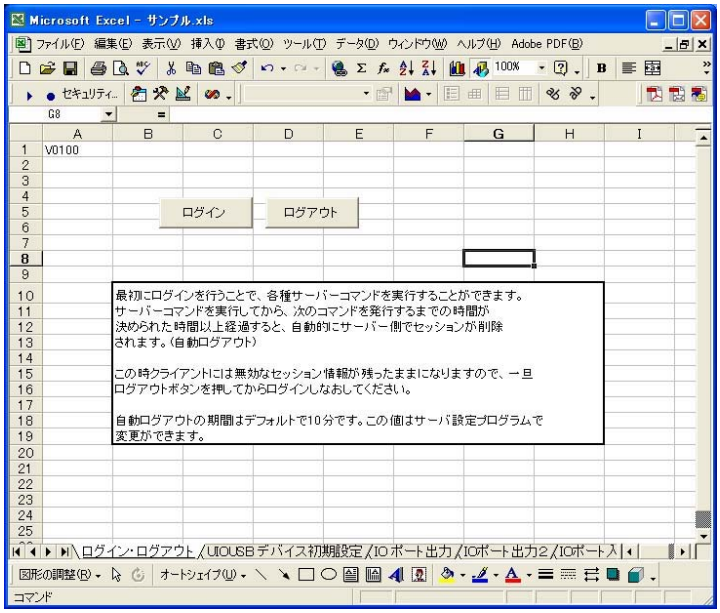
7.5 EXCEL(VBA)からのアラーム操作

PC にMicrosoft Excel(バージョン Excel2000 以降) がセットアップされている場合には、API 経由のアラーム操作を試すことができます。"C:¥Program Files¥AllBlueSystem¥Demo¥Excel¥サンプル.xls" または "C:¥Program Files (x86)¥AllBlueSystem¥Demo¥Excel¥サンプル.xls" を Excel で開いてください。

マクロを有効にするかどうかを聞いてきますので、必ず”マクロを有効にする”を選択してください。

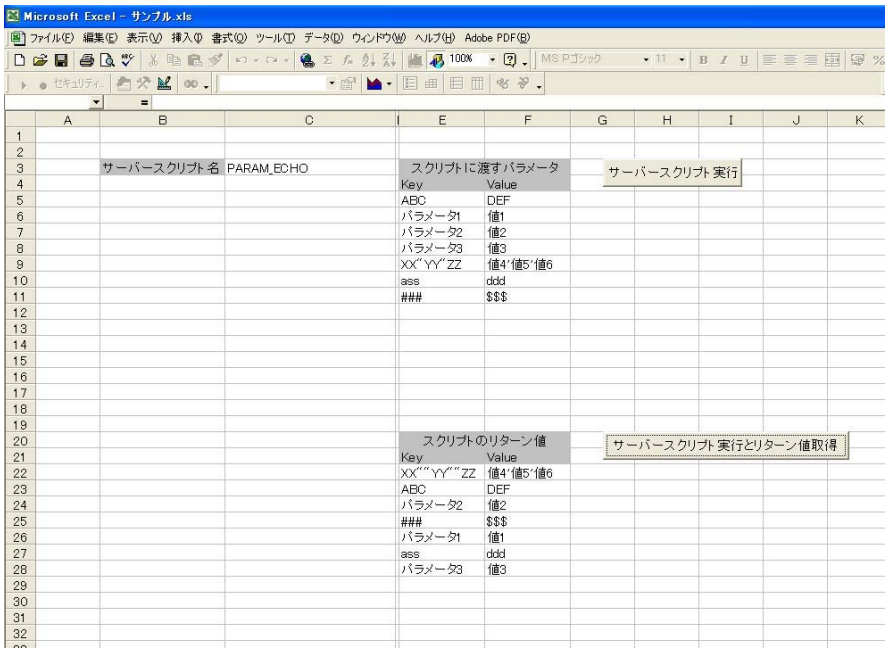
もし、DeviceServer をインストールしたPC 以外のPC にセットアップされた Excel から操作したい場合には、エクセルファイル（サンプル.xls）と DLLライブラリ(XASDLCMD.DLL) をコピーしておく必要があります。DLLライブラリコピーの方法は “インストール”の章内の ” API を使用したユーザー作成のアプリケーションを利用する場合”の項を参照してください。

サンプル.xls エクセルファイル中に、機能デモの為のワークシートが入っています。APIで操作するDeviceServerの機能に対応してワークシートに分かれていますので、タブで切り替えて実行してください。Excel をデザインモードにしてVisual Basic Editor を使用することで、ワークシート内のスクリプト内容を参照することや、カスタマイズすることができます。エクセルから API(XASDLCMD.DLL) を呼び出すときの定義ファイルも、スクリプト中に記述されていますので参考にしてください。



“ログイン・ログアウト” ワークシートタブを選択して “ログイン” ボタンを押します。ログインダイアログが表示されますので、先に登録したユーザー（guest）のユーザー名とパスワードを入力してログインします。もし、DeviceServer の動作している PC 以外から操作している場合は、”ホスト名” の内容を “localhost” から、DeviceServer の動作 PC のホスト名もしくは IPアドレスをセットして下さい。

ログインに成功したら､ワークシートタブの “SCRIPT 実行” を選択してください｡ボタン “サーバースクリプト実行” を押すとサーバースクリプト名セルで指定したスクリプトを実行することができます。その時に、セル上に入力したパラメータも同時にスクリプトに渡されます。ボタン “サーバースクリプト実行とリターン値取得”を押すと、同様にスクリプトを実行してスクリプト中で設定した複数のリターンパラメータを取得してセル上に設定します。



API 経由で操作する場合には、DeviceServer に対して 10 分のタイムアウト値（初期設定の場合）以上、何も操作がないと自動的にログインセッションが削除されてしまいます。これを防ぐために、定期的に SX_session_update() API などをコールしてユーザーのアクションが無い場合にセッション更新のみをDeviceServer に実行させることができます。

DeviceServer のクライアントプログラムやWebProxy で提供しているクライアントプログラムは上記のコールを行っています。サンプル.xls では定期的なセッション更新を利用していませんので、EXCEL 操作を行わない時間がタイムアウト値を超えると(10分)勝手にログアウトされた状態になります。この場合には、”ログアウト” ボタンを押してから、続けて “ログイン” ボタンを押してログインしなおしてください。

操作が終わったら､必ず “ログイン･ログアウト” ワークシートを選択して “ログアウト” ボタンを押してください。ログアウトを行わずにWebブラウザを終了すると DeviceServer にログインセッションが一時的に残ってしまいます。この場合、次にログインを行うときに、同一ユーザーの同時ログイン制限(ユーザー管理プログラムで設定した場合)やライセンスで決められた同時ログイン数を超えた場合に、ログインできなくなりますので注意してください。初期設定では 10 分のタイムアウト値以上操作が行われていないセッションを自動的に強制削除しますので、それ以降はログインできるようになります。または、プログラムメニューから “ALL BLUE SYSTEM” -> “セッション管理” を選択・実行して、強制的にログインセッションを終了させることもできます。

7.6 スクリプト作成

先に登録したアラームデバイス(“TestAlarm”)をスクリプトから操作する方法を説明します。スクリプトは、DeviceServer をインストールした PC の下記のフォルダに保存する必要があります。

スクリプト保存フォルダ(DeviceServer 動作 PC)
C:¥Program Files¥AllBlueSystem¥Scripts Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥Script

インストール時に予めサンプルのスクリプト（イベントハンドラを含む）が幾つか保管されています。スクリプトはテキストエディタで作成・修正して、拡張子（.lua）のファイル名でフォルダに保存するだけで直ぐに使用可能になります。

注意

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。

ここでは、下記の内容のスクリプトを作成して動作を確認します。

```
log_msg("start..","SAMPLE_ALARMSIGNAL")


local signal = g_params["Signal"]
if (not signal)  then
    if not alarm_signal_set("TestAlarm","Red",true) then error() end
    if not alarm_signal_set("TestAlarm","Beep1",true) then error() end
else
    if not alarm_signal_set("TestAlarm",signal,true) then error() end
end
```

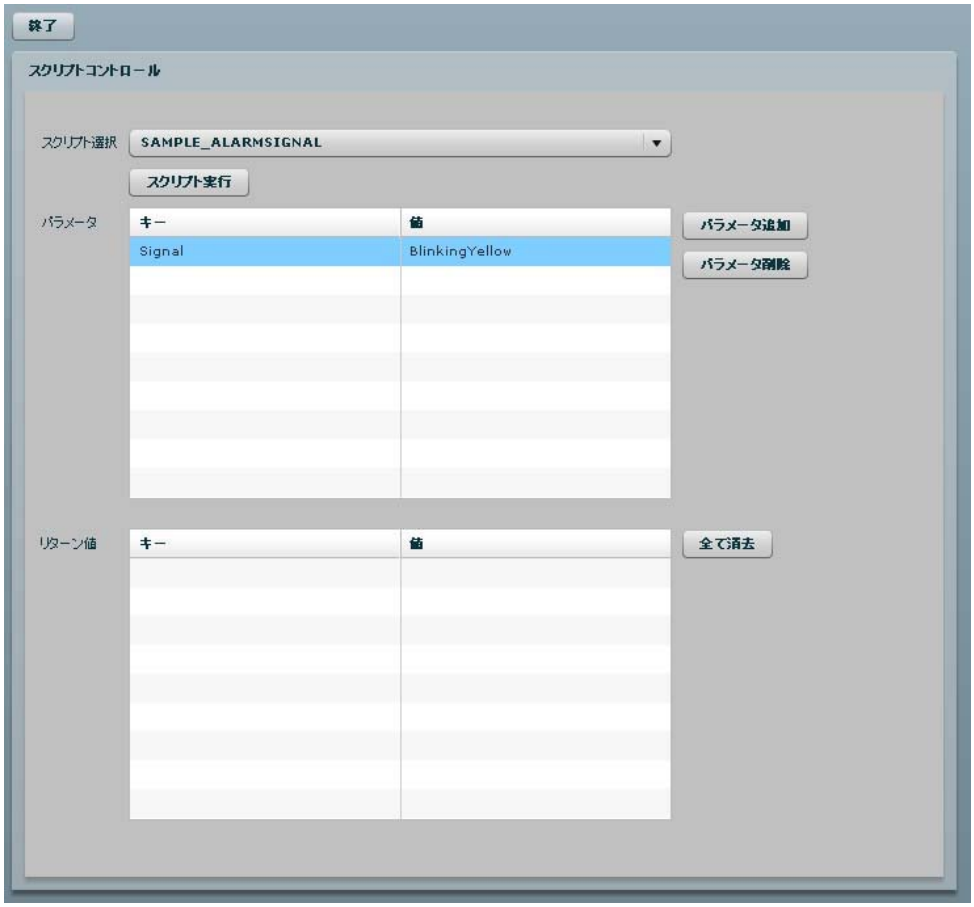
スクリプトは以下の動作を行います。

- スクリプト起動時に “start..” をログに記録
- スクリプトパラメータが指定されていないか、スクリプトパラメータ “Signal” に値が指定されていない場合は、アラームデバイス名 “TestAlarm”の赤ランプ点灯とブザー音を出力する
- スクリプトパラメータ “Signal” に値が指定された場合は、アラームデバイス名 “TestAlarm”に、パラメータで指定された値のシグナルをON にする。有効なSignal 値は、Red, Yellow, Green, BlinkingRed, BlinkingYellow, BlinkingGreen, Beep1, Beep2 のいずれか一つ。

作成したスクリプトをファイル名 “SAMPLE_ALARMSIGNAL.lua”で保存します。

スクリプトの動作を試すために、最初にWebブラウザから実行します。プログラムメニューから “ALL BLUE SYSTEM” -> “スクリプト操作” を選択・実行します。Webブラウザからアラーム管理を実行した場合と同様に、ログイン画面になりますので、先に作成したユーザー名とパスワードを入れてログインしてください。

スクリプトコントロール画面が表示されます。スクリプト選択プルダウンメニューから、”SAMPLE_ALARMSIGNAL”を選択して、”スクリプト実行” ボタンを押すとアラームシグナルプログラムのランプ点灯とブザー音を出力します。
一旦、アラームシグナルプログラムの “RESET” ボタン（右側グレイ色の四角い部分）を押してリセットした後、スクリプトコントロール画面下の ”パラメータ追加” ボタンを押して、パラメータを追加します。”キー” と ”値” のフィールドはエディットできますので、キーを “Signal”, 値を “BlinkingYellow” に変更して、”スクリプト実行” ボタンを押すと今度は黄色ランプの点滅が確認できます。



Webブラウザからのスクリプトコントロールを終了する場合は、”終了”ボタンを押してログアウトします。

次に、スクリプト実行を DLLライブラリ(XASDLCMD.DLL)経由で行います。コマンドライン用プログラム (ScriptExecCmd.exe) を利用してコマンドプロンプトからスクリプトを実行します。ScriptExecCmd.exe ファイルはインストールしたフォルダ ″C:¥Program Files¥AllBlueSystem¥″ または ″C:¥Program Files (x86)¥AllBlueSystem¥″ に格納されています。

ScriptExecCmd.exe プログラムはDLLライブラリ(XASDLCMD.DLL)を利用したプログラムで、パラメータとしてスクリプト名とサーバーホスト名、ユーザー名とパスワードを受け取って、サーバーログイン後にスクリプトを実行するプ

ログラムです。スクリプト実行後は自動的にログアウトします。(DEMO フォルダ内には ZIP ファイル中に Delphi ソースファイルも同梱してありますので参考にしてください) 詳細は、"その他プログラム"章内の" スクリプト実行プログラム・コマンドライン版 (ScriptExecCmd)" の項を参照してください。

Windows のコマンドプロンプトを起動して、下記のように入力するとスクリプトを実行することができます。
(コマンド実行は途中に改行を入れずに入力してください)

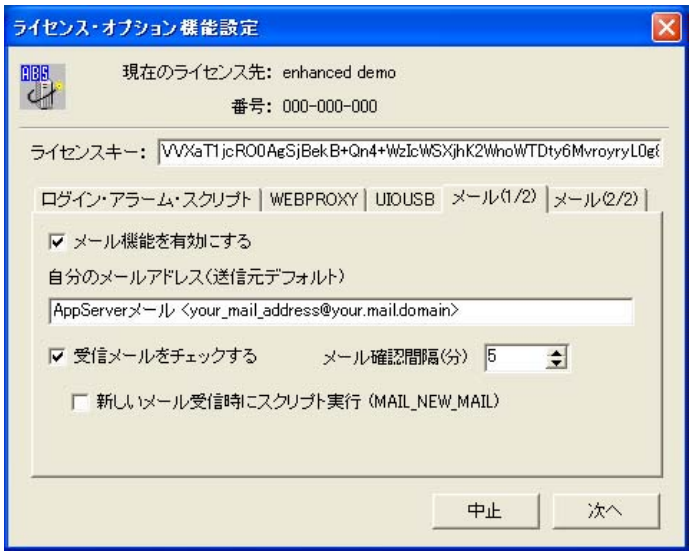
```
C:¥Projects>"C:¥Program Files¥AllBlueSystem¥ScriptExecCmd.exe" SAMPLE_ALARMSIGNAL localhost guest <guest ユーザーのパスワード>
C:¥Projects>
```

先ほどの Webブラウザからスクリプトを実行した時と同様に、アラームシグナルプログラムのランプ点灯とブザー音を出力します。

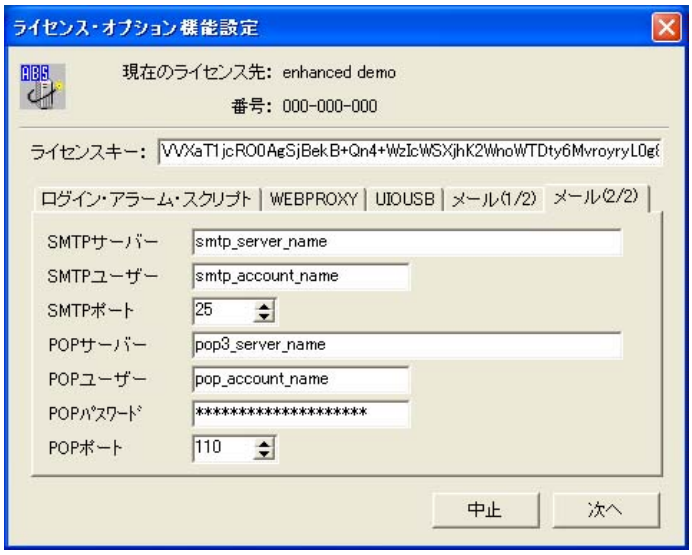
7.7 メールからのアラーム操作

メールからスクリプト実行を行う方法を説明します。インターネットメールアカウント(POP3, SMTPサーバーアカウント)が必要です。通常使用中のメールアカウントも使用できますが、その場合は使用中のメールソフト（メールクライアントプログラム）が自動受信になっていると、DeviceServerがメール受信する前にメールサーバーからメールを取り出してしまうことがあるため、手動で受信するようにするか、できればメールコマンド受信用に専用アカウントを準備してください。

DeviceServer がメールを受信して応答内容を送信するためのメールサーバーの設定を行います。プログラムメニューから “ALL BLUE SYESTEM” -> “サーバー設定” を選択・実行します。ライセンスオプション機能設定画面まで、”次へ” を押して進めます。”メール(1/2)” タブと”メール(2/2)” タブを順に選択して以下の設定項目を入力してください。



- メール機能を有効にする　チェックを有効にする
- 自分のメールアドレス　　DeviceServer から送信されたメールの送信元アドレスになります
  例：DeviceServerメール <your_mail_address@your.mail.domain>
- 受信メールをチェックする　　チェックを有効にする
- メール確認間隔　　5 分
- 新しいメール受信時にスクリプト実行　　<どちらでも構わない>



- SMTPサーバー関連の設定　DeviceServer からメールを送信するときに使用する SMTP サーバーの設定
- POPサーバー関連の設定　DeviceServer がメールコマンドを受信するときに使用する POP サーバーの設定

項目の入力が終わりましたら、サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。

次に、メール送信可能な PC や携帯電話から、下記の内容でログインリクエストのメールを送信します。

- 宛先　　サーバー設定で POP サーバー設定を行ったメールアカウント
- メール件名　$LOGIN$
- メール内容

guest <guestユーザーのパスワード *1>

(*1) パスワードはユーザー管理プログラムで設定したものを <> を除いて入力してください

暫くすると、DeviceServer から下記の様な応答メールが返信されてきます。2行目にある文字列がセッショントークンと呼ばれるもので、次に送信するスクリプト実行をリクエストするメール認証のために使用します。

ログイン成功 ST02202B28F03E32

下記の内容でスクリプト実行リクエストのメールを送信します。

- 宛先　　サーバー設定で POP サーバー設定を行ったメールアカウント
- メール件名　$SCRIPT$
- メール内容

<セッショントークン *2> SAMPLE_ALARMSIGNAL Signal　Beep2

(*2) セッショントークンは、ログインリクエストの応答メールに記載されたセッショントークン文字列を <> を除いて入力してください

メール本文の 2行目には実行するスクリプト名を入れます(スクリプトファイルの拡張子 .lua は除いて下さい)
3 行名以降には、スクリプトパラメータをキー名、値をスペースで区切って記入します。

暫くすると、DeviceServer から下記の様な応答メールが返信されてきます。またアラームシグナルプログラムでは、パラメータで指定したブザー音出力が確認できます。

スクリプト実行成功

メールでスクリプト実行をする場合は、最初にログインを行ってからスクリプト実行のメールが DeviceServerで処理されるまでに、10 分以上経過した場合はエラーになる場合があります。これは、ログインセッションのタイムアウトを検出したためです。この場合は “サーバー設定” プログラムで “操作が無い場合の自動ログアウト待ち時間” の設定値を大きくするか、メールの確認間隔を短くしてください。

メールでスクリプトが実行されると自動的にログアウト状態になります。スクリプト実行のメールを連続して送信したい場合は、メール本文の行に NO_LOGOUT を入れると自動ログアウトしなくなります。

7.8　他の機能や設定を試したい

様々なケースについて、設定方法を設定ガイドマニュアルで解説しています。具体的な設定方法を、ステップバイステップ方式で説明していますので、機能を使いこなすために、設定ガイドマニュアル（別マニュアル）を参照してください。（順次ホームページにて公開していく予定です http://www.allbluesystem.com）

8　デバイス・シリアルポート・エンドデバイス管理

DeviceServerでは、外部 I/F として複数のデバイスを接続できます。現在サポートしているのは、UIOUSB, SigSensor, NetUIO, XBeeデバイス,XBee-ZB デバイス, アラームシグナルデバイスです。その他の外部プログラムや装置と接続する場合に利用可能なシリアルポートデバイス、MQTT エンドポイント、Web APIコマンド送信、任意のUDP/TCP データを送信する機能が DeviceServerスクリプトライブラリに用意されています。これを使用することでデータやイベントを送信することもできます。

ここでは、それぞれのデバイスの機能や DeviceServer への登録方法や設定変更方法について説明します。

8.1 UIOUSBデバイス

DeviceServer の USB ポートに接続できる汎用 I/O デバイスで、以下の機能があります。

- 8 ビットI/O ポート(RB) 出力
- 8 ビットI/O ポート(RB) 入力
- 4 チャンネル 10ビットA/D 変換入力 (RA0, RA1, RA2, RA3)
- 2 チャンネル PWM信号出力 (CCP1, CCP2)
- 8 チャンネル 簡易サーボ信号出力(RB)
- 8 ビットI/O ポート(RB) 入出力切り替え設定
- 入力ポートのプルアップ切り替え設定
- 設定値をPIC内部 EEPROM に保存（デバイスリセット時に自動で読み込まれます）

DeviceServer からは、これらの機能をスクリプトライブラリや API (DLLライブラリ)から操作します。簡単なデバイスの操作であればWebブラウザ経由で UIOUSB コントロール画面からマウスで操作することもできます。

詳しい UIOUSBデバイスの機能やファームウエアのインストール方法については “UIOUSB ユーザーマニュアル” を参照してください。

8.1.1 UIOUSB 接続方法

UIOUSB デバイスは DeviceServer の動作している PC に１つだけ接続することができます。最初に UIOUSB デバイスをPC に接続するとドライバインストールが開始されますので、“UIOUSB ユーザーマニュアル” に従ってドライバをインストールしてください。デバイスが使用可能になると、PC からは 仮想 COM ポートを経由でUIOUSB デバイスを操作可能になります。

デバイスが PC に接続されたら、次に DeviceServer から UIOUSB デバイスを使用可能にするために COM ポートの設定を行います。サーバー設定プログラムを起動して、UIOUSB タブを選択して COM ポート番号を設定して “UIOUSB 機能を有効にする” にチェックをつけてください。サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。

8.1.2 UIOUSB設定内容変更

UIOUSB デバイスは最初に使用する時は、初期状態で全ポートが入力モード、PWM, SERVO が OFF の状態になっています。ポートを出力に設定する場合は、UIOUSB の仮想COMポート経由で 設定変更用のコマンドを実行する必要があります。ただし、UIOUSB が DeviceServer に接続した状態では、仮想COMポートはDeviceServer が排他的に利用中なので直接コマンドをシリアルポートに送る方法ではなく、スクリプトまたは API 経由でコマンドを送信します。

ここでは、スクリプト経由でUIOUSB デバイスの設定を変更する方法を説明します。
設定変更を行うための UIOUSB のコマンドが記述された、下記の様なスクリプトを作成します。この例では、全ポートを出力にして、プルアップ無効、PWM の２つの channel を ON にして、出力 duty 値を 0 にします。その後、設定内容をデバイス内部の EEPROM に保存します。これで次からは、デバイスリセット後に保存した設定値が自動で読み込まれて設定が完了します。**UIOUSB デバイスは設定変更後に、直ぐに設定変更内容が有効になりますので、リセットを行う必要はありません。**

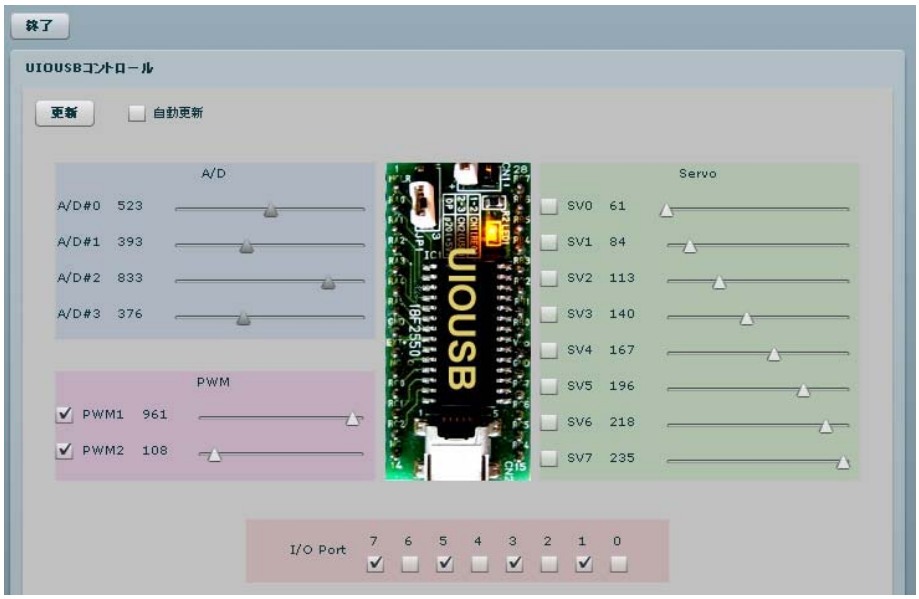
```
if not uio_command("dcfg 0") then error() end
if not uio_command("pullup 0") then error() end
if not uio_command("duty1 0") then error() end
if not uio_command("duty2 0") then error() end
if not uio_command("pwm1 1") then error() end
if not uio_command("pwm2 1") then error() end
if not uio_command("save") then error() end
```

(インストール時に UIOUSB_INIT.lua のファイル名で “C:¥Program Files¥AllBlueSystem¥Scripts” フォルダに保管されています)

上記のスクリプトを Web ブラウザのスクリプトコントール画面から実行するか、コマンドプロンプトから ScriptExecCmd.exe プログラムを使用して実行してください。”動作確認と簡単な使い方” の章に、スクリプト実行方法について詳しい説明がありますのでそちらも参照してください。

8.1.3 UIOUSB動作確認

UIOUSB を前述のスクリプトを使用して設定した場合は、Webブラウザから UIOUSB コントロール画面を使用して簡単に I/O ポートの値を出力することや、PWM 出力の duty の変更をすることができます。プログラムメニューから “ALL BLUE SYSTEM” -> “UIOUSB操作 (WebProxy)”を選択・実行してください。



また、スクリプトから UIOUSB ポートの内容を操作するサンプルが多数インストールされていますので “C:¥Program Files¥AllBlueSystem¥Scripts” フォルダも参考にしてください。

8.2 SigSensor, NetUIOデバイス

DeviceServer とネットワーク接続する汎用 I/O やアラーム機能を持ったデバイスです。ネットワークに複数台接続して、DeviceServerから操作することができます。デバイス上で発生したイベントを DeviceServer に送信する機能があり、イベントハンドラをスクリプトで記述することができます。 SigSensor, NetUIOデバイスはアラーム管理プログラム(AlarmConfig) で管理され、デバイスのネットワーク設定やI/O 詳細機能の設定は、このプログラムから行います。

SigSensor, NetUIO デバイスには、以下の機能があります。

(NetUIO)

- 4 ビットポート入力
- 4 ビットポート入力値に変化があった場合のイベント送信
- 8 ビットポート出力
- 4 チャンネル A/D 変換
- 4 チャンネル A/D 自動サンプリングと変換バッファ格納 (1024 ポイント)
- 4 チャンネル A/D 自動サンプリング時に閾値を越えた場合のイベント送信

- 4 チャンネルプッシュスイッチ入力時のイベント送信

(SigSensor)

- 4 ビットポート入力
- 4 ビットポート入力値に変化があった場合のイベント送信
- 4 ビットポート出力
- 4 チャンネル A/D 変換
- 4 チャンネル A/D 自動サンプリングと変換バッファ格納 (1024 ポイント)
- 4 チャンネル A/D 自動サンプリング時に閾値を越えた場合のイベント送信
- 4 チャンネルプッシュスイッチ入力時のイベント送信
- LCD 表示モジュールへの文字列表示
- 3 種類の警告表示(LED)とブザー出力

DeviceServer からは、これらの機能をスクリプトライブラリや API (DLLライブラリ)から操作します。複数のデバイスの各種機能を同時にコントロールすることができます。簡単なデバイスの操作であればWebブラウザ経由で アラームコントロール画面からマウスで操作することもできます。API (DLLライブラリ)を使用して独自のユーザーアプリケーションを作成することや、既存のシステムに組み込むことができます。

詳しい SigSensor, NetUIOデバイスの機能やファームウエアのインストール方法については “SigSensor_NetUIOユーザーマニュアル” を参照してください。

8.2.1 SigSensor, NetUIO 接続方法

SigSensor,NetUIO デバイスはネットワーク上に複数設置して DeviceServer に接続することができます。
SigSensor,NetUIO デバイスは、ファームウエア書き込み後の最初に起動した時に初期設定状態になります。そのときのセットワーク設定は下記になっています。

SigSensor, NetUIO設定項目	初期設定値
IP アドレス	192.168.100.30
ポート番号	27103
IP ネットワークマスク	255.255.255.0
デフォルトゲートウエイ	192.168.100.1
CSVIF ホストアドレス(DeviceServerのホストアドレス)	192.168.100.40
CSVIF ポート番号	27102

デバイスの設定変更を行うときに、一時的に DeviceServer が動作している PC は、上記の初期設定状態のアドレスに接続可能な、同一ネットワークに属している必要があります。具体的には下記のIPアドレスのいずれかとサブネットマスクを DeviceServer の動作している PC が持っている必要があります。ユーザーが現在使用中のネットワークアドレスと違う状態で接続するため、初期状態のデバイスのネットワーク設定を変更する時は本来の運用環境とは切

り離したネットワーク上で行ってください。

初期状態のSigSendor, NetUIO デバイスにアクセスするために必要な設定項目	設定値(例)
IP アドレス (xxxx は 30 以外の任意のアドレス)	192.168.100.xxxx
IP ネットワークマスク	255.255.255.0

SigSensor, NetUIO デバイスのネットワーク設定は、**アラーム管理プログラム**から行います。この方法については次の項で詳しく説明します。

複数のデバイスを接続する場合に、**初期状態の SigSensor, NetUIOデバイスは IP アドレスが重複するのを防ぐため必ず一つずつネットワークに接続してください**。デバイスのネットワーク設定を変更した後は、デバイスの電源を切ってネットワークから切り離してください。その後、次の初期状態のデバイスを接続して同様にネットワーク設定を変更します。全てのデバイスのネットワーク設定が終了したら、DeviceServer の動作している PC のネットワーク設定を通常のユーザー環境で使用している値に戻します。その後、ネットワーク設定変更済みの SigSensor, NetUIO デバイスを全てネットワークに接続します。

デバイスのネットワークアドレス(上記のデバイス初期設定時だと 192.168.100.0) を変更するとき以外は、アラーム管理プログラムから設定内容を自由に変更できます。その時に上記のような DeviceServer の動作 PC のネットワークを変更する必要はありません。(IP アドレスも変更できます)

一時的にサーバーとして動作可能 PC がもう一台使用可能でしたら、本来運用予定の PC のネットワーク設定を変更しないでアラーム管理プログラムが実行できますので、作業が簡単になります。一時的にデバイス設定目的に使用するDeviceServer にはデモライセンスが使用できます。(デモライセンスには有効期限がありますので、最新版をオールブルーシステムのサイトからダウンロードして下さい)

8.2.2 SigSensor, NetUIO 設定内容変更

アラーム管理プログラムを使用して、SigSensor, NetUIO デバイスを DeviceServer に登録することや、デバイス自身の設定内容を変更することができます。プログラムメニューから “ALL BLUE SYSTEM” -> “クライアント起動”を選択・実行します。ログインするときは、管理者特権をもったユーザー (例えば DeviceServer セットアップ時に管理者アカウントとして登録したユーザー等) でログインしてください。デスクトッププログラムが起動したら、 “アラーム” ツールボタンを選択してアラーム管理プログラムを起動します。

DeviceServer では SigSensor, NetUIO, アラームシグナルなどのデバイス管理用のデータベースを保管しています。また SigSensor, NetUIO デバイスにも自分のIPアドレスやイベント送信先の DeviceServerの IP アドレス情報等を、CPU ボード上の24C256(EEPROM) チップに保管しています。アラーム管理プログラムではこれら両方の設定情報を変

更することができます。

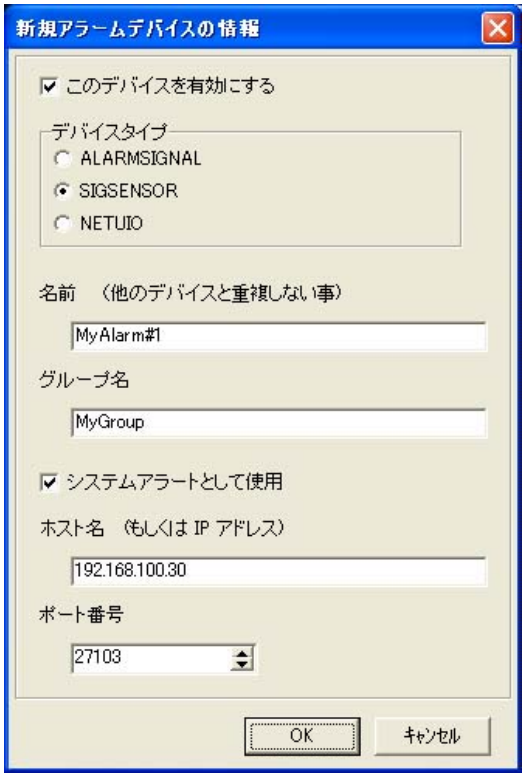
デバイス登録時は、最初に DeviceServer のデバイス管理用データベースにデバイスを登録します。このときに、SigSensor, NetUIO デバイスの現在のネットワーク設定に合わせてデバイス管理用のデータベースに登録します。この状態で、デバイスと通信できるようになりますので、アラーム管理プログラムからデバイスのネットワーク設定やその他の詳細設定を変更することができます。その後デバイスをリセットするとデバイスは新しいネットワーク設定で起動しますので、再度アラーム管理プログラムからデバイス管理用データベースを新しいネットワーク設定に合わせて変更します。

以下に、登録するステップを順に説明します。

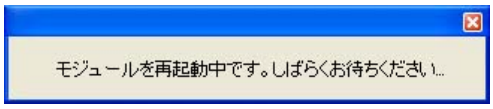
- **[Step1/4] 新しいデバイス登録を行う**

アラーム管理プログラムの“追加”ツールボタンを押します。

デバイス情報を入力するダイアログが表示されますのでデバイスの詳細情報を入力します。このとき、デバイスの登録画面の IP アドレス設定は、現在または初期設定状態のアラームデバイスに合わせてください。デバイスタイプを、接続するデバイスに合わせて設定します。システムアラートとして使用するにチェックを付けると、スクリプトライブラリから alarm_sysalert_list()を使用して、このデバイスの名前を取得することができます。これは警報出力を行う対象デバイスを自動選択するときに利用します。グループ名はアラーム管理画面のデバイス一覧でソートを行って、見やすくするために利用します。



入力が終わったら、“OK” を押して、登録を行います。アラームデバイスを追加すると、DeviceServer はデバイス管理用データベースをロードして初期化するために、アラームモジュール機能をリスタートします。再起動中のダイアログが表示された後にアラーム管理プログラムのデバイス一覧を更新します。

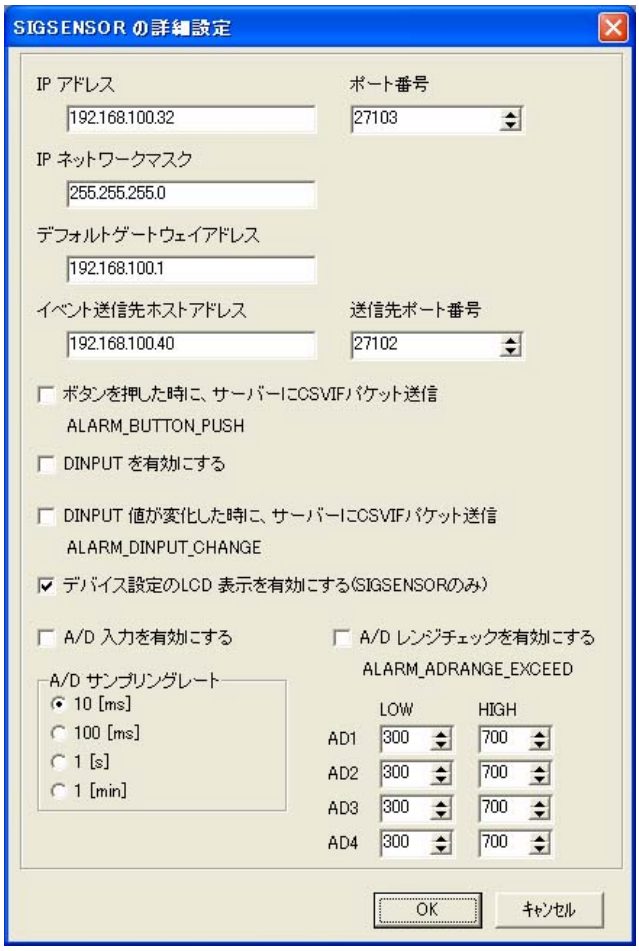


- **[Step2/4] デバイス詳細設定を変更(デバイス自身が持つ設定)**

デバイスの詳細設定を変更するために、アラーム管理プログラムのデバイス一覧から対象デバイスを選択して、”機能” ツールボタンを押します。

デバイスと通信を行って、現在のデバイス情報を取り込みます。もし、この時点でエラーが発生した場合は [Step1/4] で設定したデバイス情報(IP アドレスまたはポート番号)に間違いがあります。その場合は、アラーム管理プログラムのデバイス一覧から対象デバイスをダブルクリックするか、選択した後に “変更” ボタンを押して設定を変更してください。もしどうしても接続できない場合は、後述のデバイスの設定を強制的に初期状態に戻してやり直してください。

デバイスの現在の設定値を取り込んだ後、詳細設定変更用のダイアログを表示します。
ここでは、デバイスのネットワークと詳細機能を設定しますが、最低限ネットワーク関連の設定だけは必ず行って下さい。IP アドレスとネットワークマスク、デフォルトゲートウェイアドレスは運用時に使用するこのデバイスのアドレスを入力してください。イベント送信先ホストアドレスは運用時に使用する DeviceServer のホストアドレスを入力してください。デバイスにはホスト名解決(DNS検索)の機能がありませんので、DeviceServer のホストアドレスには IP アドレスを入力してください。その他の詳細設定項目については “クライアントソフトウエア”の章の “アラーム管理(AlarmConfig)”の項を参照してください。



入力が終わったら、“OK” を押して、新しい設定内容をデバイスのEEPROMに書き込みます。

- **[Step3/4] デバイスのリセットを行う**

SigSensor, NetUIO デバイスの詳細設定(EEPROM の設定値)を変更した場合は、手動でSigSensor, NetUIO のデバイスリセット(CPU ボードのリセットボタンを押すか、電源の抜き差しを行う)することで始めて、新しい設定値が有効になります。アラーム管理プログラムの詳細設定画面を “OK” を押して変更を有効にした後、SigSensor デバイスは LCD に “*WARNING* xxxx” の表示がされてデバイスリセットを行う様にユーザーに知らせます。NetUIO の場合は LCD 表示機能がありませんので、忘れずに必ずデバイスリセットまたは電源の抜き差しを行ってください。

- **[Step4/4] デバイスの新しいネットワークに合わせて設定内容を変更する**

[Step2/4] でデバイスの IP アドレスを変更した場合は、アラーム管理プログラムでデバイスを選択した後、”変更” ボタンを押して、デバイスの新しいネットワークに合わせて設定内容を変更します。

デバイス情報を入力するダイアログを表示しますので、[Step2/4]でデバイスに設定した値を入力します。
入力が終わったら、“OK” を押して、変更を行います。アラームデバイスを変更すると、DeviceServer はデバイス管理用データベースをロードして初期化するために、アラームモジュール機能をリスタートします。再起動中のダイアログが表示された後にアラーム管理プログラムのデバイス一覧を更新します。

現在接続中のデバイスの設定変更を行う場合は、最初に[Step1] で”追加”ツールボタンではなく、変更対象のデバイスを選択した後に “変更” ツールボタンを押します。これ以降の操作はデバイス追加の時と同じです。

デバイスの削除を行う場合は、アラーム管理プログラムで削除対象のデバイスを選択した後、”削除”ボタンを押してください。

8.2.3 リセット直後の通信エラーについて

DeviceServer の動作している PC の ファイアーウォールの設定や OS の設定によっては、デバイスリセット直後にデバイスからのイベント送信時がエラーになることがあります。これは TCP 通信時に DeviceServer の動作している OS からの応答が一時的にブロックされているために発生します。デバイスのプッシュボタンをおしたときに、DeviceServer の動作している PC でログコンソールプログラムを起動しておくと、イベント送信のログが確認できますが(デフォルトのイベントハンドラスクリプトの場合)、通信エラーが発生しているとイベントが記録できません。

SigSensor デバイスの場合には、通信エラーが発生すると LCD に “*ERROR*”表示が行われるので直ぐに確認することができます。通信エラーが発生した場合は、しばらく(数分)待ってから再度送信することで通常は復帰できます。一旦デバイスから DeviceServer への通信が成功すると、デバイスがリセットされるまではこの様な現象は発生しません。

8.2.4 デバイスの設定値を強制的に初期値に戻す

アラーム管理プログラムからSigSensor, NetUIOデバイスに接続可能な場合は、対象デバイスを選択した後 “設定消去”ボタンを押した後、SigSensor, NetUIOデバイスをリセットするとデバイス自身の全設定項目は初期値に戻って起動されます。

アラーム管理プログラムからSigSensor, NetUIOデバイスに接続できない状態になって、デバイスの IP アドレスも不明な場合は、SigSensor, NetUIOデバイスのプッシュボタンの A, B, C を同時にしばらく押してください(数秒)。SigSensor デバイスの場合のみ LCD に “Factory reset” が表示されます。その後、デバイスをリセットするとデバイス自身の全設定項目は初期値に戻って起動されます。

SigSensor, NetUIOデバイスの CPU ボードに接続した 24C256(EEPROM) チップを取り出してEEPROM ライタで、アドレス 0x000 にデータ 0x00または 0xFF の１バイトを書き込むことで、同様に初期化することができます。その後、24C256(EEPROM)を CPU ボードに戻してデバイスを起動すると全設定項目は初期値に戻っています。

8.2.5 SigSensor, NetUIO 動作確認

デバイスの設定が終了したら、Webブラウザから アラームコントロール画面を使用して簡単にアラームのシグナル値や出力ポートの値を変更できます。プログラムメニューから “ALL BLUE SYSTEM” -> “アラーム操作 (WebProxy)”を選択・実行してください。

アラーム管理プログラムで登録済みのデバイスを一覧で表示します。4つ以上のデバイスを登録している場合にはスクロールバーを操作して残りのデバイスを確認できます。最新のデバイスのグナル情報(ランプ、ブザー出力)、入力ポート値、出力ポート値、A/D 変換値(最も直近にサンプリングされたもの)を取り込んで表示します。デバイスタイプによって、サポートされる I/O 機能に違いがありますので表示される項目は違います。”更新”ボタンを押すと、デバイスから最新の値を取り込んで表示内容を更新します。”自動更新”にチェックを付けると約 1 分毎に更新を行います。

デバイスシグナル情報と出力ポート項目は、マウスでチェックボックスを操作することで、デバイスのシグナルやポート値を直接変更することができます。

デバイスのネットワーク障害で、DeviceServer からアクセスできない場合は、エラー表示され該当デバイスの表示項目は空白になります。エラーが発生した後に、”更新”ボタンを押すとエラーが発生したデバイスは一時的に DeviceServer から切り離された状態になって、表示されるデバイス一覧からは除かれます。デバイスのネットワーク接続を復旧した後に、”エラーリセット”ボタンを押すと、一時的に切り離された状態は元に戻り、次に”更新”ボタンを押すとデバイスを表示します。

8.3 シリアルデバイス(Arduino、TWEワイヤレスデバイス、計測器、I/O装置、GPS 等)

シリアルポートでコントロール可能なデバイスを DeviceServer に接続してデータの受信や送信をすることができます。シリアルポートで通信が可能な CPU ボード(Arduino など)、シリアルデータを出力可能な計測器、シリアル

ポート経由のコマンドで制御可能な I/O 装置などをコントロールできます。

TWEワイヤレスデバイスをシリアルポートに接続して、複数のリモート側 TWEワイヤレスデバイスをコントロールすることもできます。

DeviceServer では複数のシリアルポートを同時にコントロールすることができます。各シリアルポートでデータを受信すると、イベントが発生して各々のデバイス毎にデータの処理を行うことができます。

シリアルポートをバッファ入力モードに設定することで、スクリプト中の serial_readln() ライブラリ関数を使用してデータを順番に読み出して処理を行うことも可能です。

DeviceServer のシリアルデバイスを登録する時には、下記のデバイスタイプのいずれかを選択します。

- STRING デバイスタイプ
  主に文字列形式でデータをやり取りする場合に設定します。文字列の終端を自動で判別して文字列毎にイベントの発生やバッファからの取り出しを行います。
- FIRMATA デバイスタイプ
  Arduino デバイス上で実行する FIRMATA 標準ライブラリで定義されたプロトコルを扱います。
  FIRMATA プロトコルを自動で判別して、１パケットデータ毎にイベントの発生や、バッファからの取り出しを行います。
- RAW デバイスタイプ
  シリアルポートから受信したデータの固まりをバイナリデータとしてそのまま扱います。
- TWE デバイスタイプ
  TWEワイヤレスデバイスで使用しているアスキー形式のプロトコルを扱います。主に、デフォルトで提供されている TWE-Zeroアプリで扱っているアスキー形式に対応します。扱えるアスキー形式は、":" から始まるHEXデータ列、"::" から始まるキーと値を表したデータ列、";" から始まる SimpleTagV3形式、"!" または "*" から始まるコメントデータです。ユーザーが独自に作成したアスキー形式を使用する時には、イベントハンドラ SERIAL_TWE.lua 中に対応する処理を記述して自由に拡張することもできます。

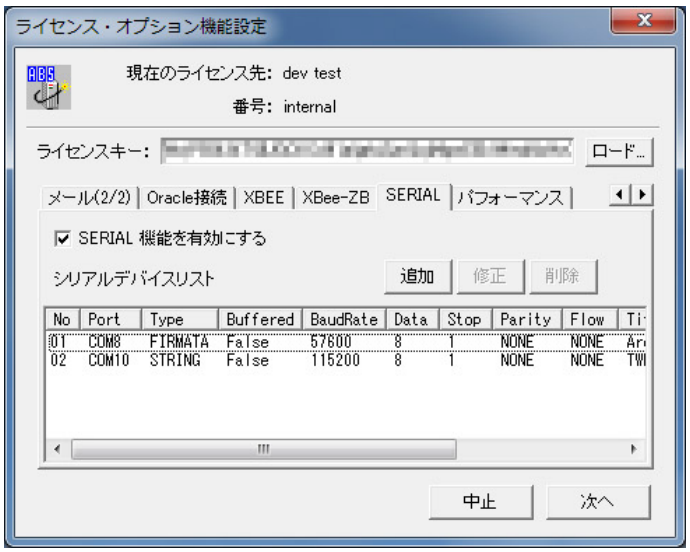
FIRMATA プロトコルを自動で判別して、１パケットデータ毎にイベントの発生や、バッファからの取り出しを
送信時には、serial_print(), serial_write() ライブラリ関数を使用して、文字列データやバイナリデータの送信を行います。また、FIRMATA デバイスタイプのシリアルデバイスに対して、SYSEX フォーマットの Query-Reply 形式のデータをやり取りすることもできます。このときには、serial_write() ライブラリ関数で、Reply パケットのバイナリコマンドデータを指定することで、一致するデータを受信するまで待機させることもできます。

TWE ワイヤレスデバイスにデータを送信する場合には、専用の twe_print() ライブラリ関数を使用できます。パラメータで指定した任意の文字列データをデバイスに送信できます。この関数を使用すると、TWEワイヤレスデバイスのデフォルトアプリケーション "超簡単！TWEアプリ" で使用できる Query-Reply タイプのコマンドも簡単に使用できます。コマンドデータ送信後にリプライデータで受信したいステータスバイト値を指定すると、一致するデータ

を受信するまで待機させることができます。独自のファームウエアを作成する場合でも、TWEワイヤレスデバイスで定義されたアスキー形式のフォーマットに合わせておくと、簡単にワイヤレスデバイスを使用したアプリケーションを構築することができます。

8.3.1 シリアルデバイスの追加・修正

シリアルデバイスを DeviceServer で使用するために、シリアルデバイスの設定を行います。サーバー設定プログラムを起動して、SERIAL タブを選択してデバイスの登録や修正、削除を行います。



デバイスの追加や修正を行う場合には、シリアルデバイスの設定画面で通信パラメータを設定します。

個々の設定項目については、”サーバーソフトウエア” の章の “サーバー設定(ServerInit)” プログラムの説明を参照して下さい。

シリアルデバイスの設定が終了したら、サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。シリアルデバイスの設定を変更した場合には、必ず DeviceServer を再起動する必要があります。サーバー設定プログラムでシリアルデバイスの設定を変更した場合には、自動的に DeviceServer サービスプログラムを再起動します。

8.4 XBee 802.15.4 シリーズ 1 デバイス

Digi international Inc.社製の IEEE 802.15.4 無線通信デバイスです。デジタル入力と出力、PWM 出力等をデバイス単体で可能です。また、A/D 変換入力機能を使用することもできます。

DeviceServer には COM ポート経由で一つの XBee デバイスを接続します。この XBee デバイス経由で同一 PAN(Personal Area Network) 内の複数の リモートXBee デバイスを操作することが可能です。

リモート XBee デバイス上で発生したイベントを DeviceServer に送信する機能があり、イベントハンドラをスクリプトで記述することができます。 XBee デバイス（リモートデバイスを含む）は、XBee デバイス管理プログラム (XBeeConfig) で管理され、デバイスの設定やI/O 詳細機能の設定は、このプログラムから行います。

XBee IEEE 802.15.4 OEM RF デバイスには、以下の機能があります。

- DIO0-DIO4　　ポート入力、出力、A/D 変換入力
- DIO5　　ポート入力、出力、A/D 変換入力
- DIO6　　ポート入力、出力
- DIO7　　ポート入力、出力
- DIO8　　ポート入力
- ポート入力値に変化があった場合のI/Oデータフレーム送信
- ポート入力のプルアップ設定

DeviceServer では、これらの XBee デバイス機能をスクリプトライブラリを使用して操作できます。詳しい XBee デバイスの機能詳細については Digi international Inc. のドキュメントを参照してください。

注意

XBee デバイスは XBee IEEE 802.15.4 OEM RF モジュールのファームウエアバージョン "10CD" で動作確認しています。これ以前のバージョンの XBee デバイスは、DeviceServer では使用できませんので注意してください。 XBee デバイスのファームウエア更新は Digi international Inc.社の X-CTU プログラムを使用することで可能です。詳しいファームウエアの仕様と更新方法については XBee デバイスの購入元もしくは Digi international Inc.社のドキュメントを参照して下さい。

8.4.1　　XBee デバイス初期設定

XBee デバイスを DeviceServer 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、DeviceServer に直接接続する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以外のXBeeデバイスの設定項目は、後からXBee管理プログラムで操作することができます。

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。

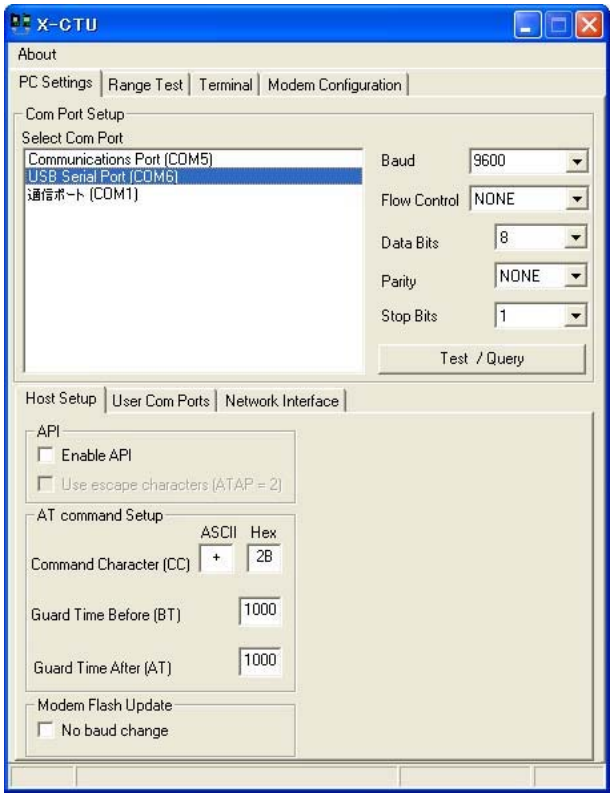
XBee デバイス設定項目	設定値
API モード	1 (default は 0)
PAN(Personal Area Network) ID	任意の値 (default はOx3332) デフォルトの値のままだと、予期しないデバイスからのフレームを受信したり、間違ってデバイスを操作する恐れがありますので、適当な任意の値を設定するようにしてください。このマニュアルでは 0xAB90 を使用しています。
16bit Source Address	同一PAN ID 内でユニークな値 (default は 0x0000)

	この値は、ここで設定しなくても後から XBee 管理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。

初期設定のコマンドをXBee に送信するために、XBee デバイスを PC のCOM ポートに接続して Digi international Inc. 社製の X-CTU プログラム、または汎用のターミナルエミュレータプログラム等を使用します。XBee と COM ポートのボーレートは初期値の 9600bps を使用します。(ターミナルエミュレータの場合は、ローカルエコー ON, 受信時の改行 CR + LF にするとコマンド実行の結果が見やすくなります)

このマニュアルでは、Sparkfun Electoronics 社製の XBee Explorer USB を使用して、仮想 USB ポート経由で接続した例で説明します。これ以外の方法で COM ポート接続する場合も手順は同じです。
既に、XBee Explorer USBのドライバ等がインストール済みで仮想 USB ポートが作成されているものとします。(例では COM6)

X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。

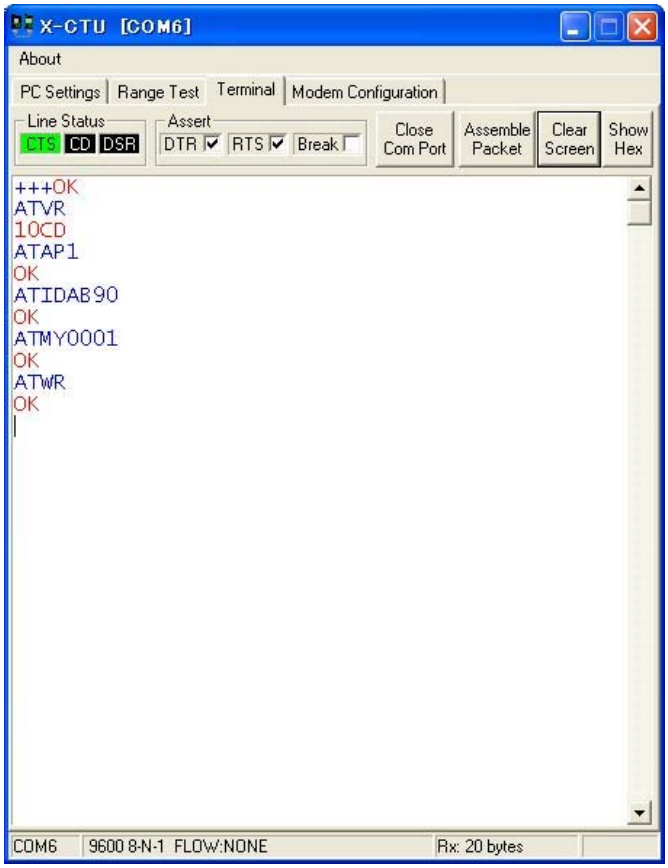


Terminal タブを選択してターミナル画面を表示します。キーボードから、"+++" を入力して、コマンドモードに入

ります。コマンドモードに入ると “OK” が表示されますので、続けて以下のコマンド文字列を入力してください。
コマンド入力の時間がかかりすぎると、自動的にコマンドモードから抜けてしまいますので、その場合は、”+++” を入力して最初からコマンドを入力し直して下さい。

```
ATVR
ATAP1
ATIDAB90
ATMY0001
ATWR
```

最初に ATVR でファームウエアバージョンを表示しています。”10CD” 以降になっていることを確認してください。
ATAP1 は、API モードを “1” に設定しています。ATIDAB90 は PAN_ID を 0xAB90 に設定しています。もし別の PAN_ID を使用する場合は適宜変更してください。次に、ATMY0001 で、デバイスの16 bit Source Address を “0x0001” に設定しています。この部分は、デバイスごとにユニークな値になるように変更して下さい。
最後に、ATWR で、設定値を不揮発メモリに書き込みます。実際に入力した時の画面表示は以下の様になります。



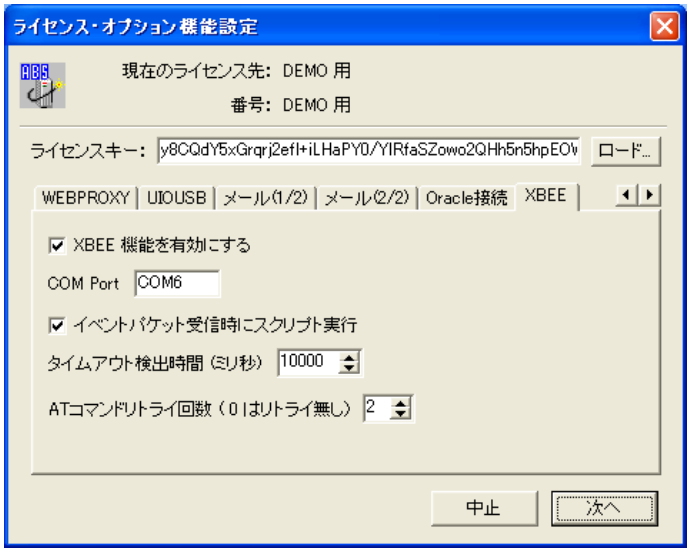
X-CTU プログラムを終了します。 その後、XBee Explorer USB に接続する XBee デバイスを切り替えて、DeviceServer で使用する全ての XBee デバイスについて同様に初期設定を行って下さい。
このときに、設定した16bit Source Address の値をデバイス機器にマーキングしておくと、後で XBee デバイス管理プログラムでデバイスを選択するときに、識別し易くなります。

8.4.2 XBee デバイスのDeviceServer接続方法

XBee デバイスは DeviceServer の動作している PC に１つだけ COM ポート経由で接続することができます。
その他の リモートXBee デバイスは、DeviceServer の COM ポートに接続したXBee デバイス経由でワイヤレス制御します。

このマニュアルでは、Sparkfun Electoronics 社製の XBee Explorer USB を使用して、仮想 USB ポート経由で接続した例で説明します。これ以外の方法で COM ポート接続する場合も手順は同じです。
既に、XBee Explorer USBのドライバ等がインストール済みで仮想 USB ポートが作成されているものとします。(例では COM6)

デバイスが PC に接続されたら、DeviceServer から XBee デバイスを使用可能にするために COM ポートの設定を行います。サーバー設定プログラムを起動して、XBEE タブを選択して COM ポート番号を設定して “XBEE 機能を有効にする” にチェックをつけてください。サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。



8.4.3 XBee デバイス設定内容変更

XBee デバイス管理プログラム(XBeeConfig)を使用して、同一 PAN ID に設定された XBeeデバイスを DeviceServer に登録したり、XBee デバイスの詳細設定をリモートから変更することができます。

プログラムメニューから “ALL BLUE SYSTEM” -> “クライアント起動”を選択・実行します。ログインするときは、管理者特権をもったユーザー（例えば DeviceServer セットアップ時に管理者アカウントとして登録したユーザー等）でログインしてください。デスクトッププログラムが起動したら、 “XBee” ツールボタンを選択してXBee デバイス管理プログラムを起動します。

DeviceServer では 登録済みの XBee デバイスをマスターファイルに記録しています。
XBee デバイス登録は、”探索＆登録” ボタンを押すことで、同一 PAN ID のリモートデバイスを見つけて、自動的にマスターファイルに登録します。既に、登録済みのXBee デバイスの場合は最新の情報でマスターファイルの内容を更新します。DeviceServer に COMポートで直接接続された XBee デバイスについても同様に自動登録します。

一度、マスターファイルに登録された XBee デバイスは、”削除” ボタンを押して、デバイス登録を削除しない限りマスターファイルに保存されたままで、デバイスリストにも常に表示されます。

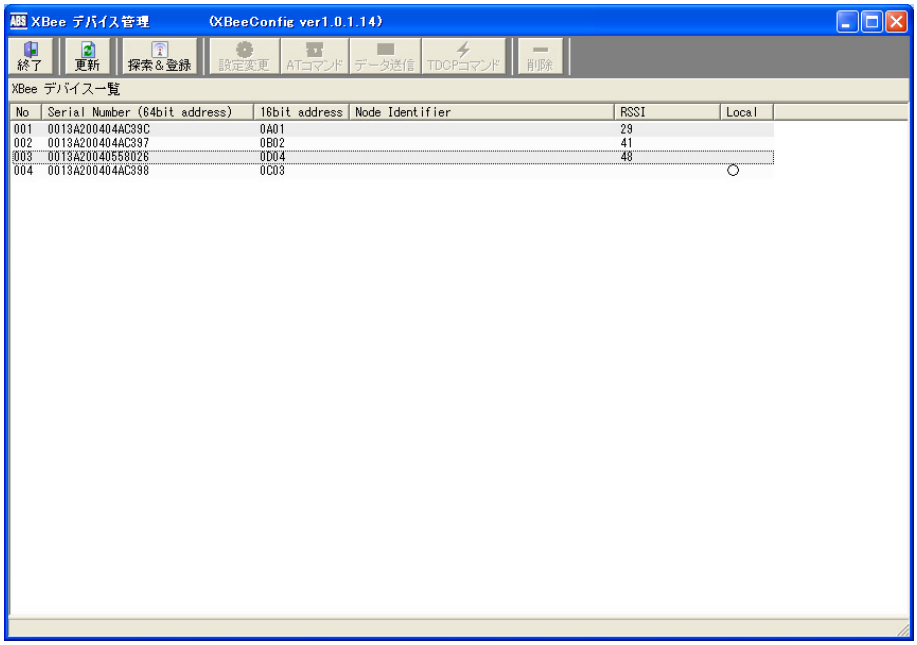
デバイスが登録されたら、デバイスを一覧から選択して”設定変更”ボタンを押すことで、XBee デバイスの詳細設定を変更することができます。ここでは最低限、XBee デバイス詳細設定項目中の Node Identifier、16bit Source Address について設定を行ってください。Node Identifier は、デバイス毎にわかり易い名前を設定することができ、DeviceServer のスクリプト中から XBee デバイスを特定する場合に、この Node Identifier を使うことができます。

以下に、XBee デバイスの登録・詳細設定ステップを順に説明します。

- **[Step1/3] XBeeデバイス探索＆登録を行う**

XBee デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

登録確認ダイアログが表示されますので、”OK” を押します。
DeviceServer の COM ポートに直接接続された 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。XBee デバイス管理プログラムのデバイス一覧には、登録済みのXBee デバイスを表示します。



（デバイスの探索＆登録が完了したときの画面）

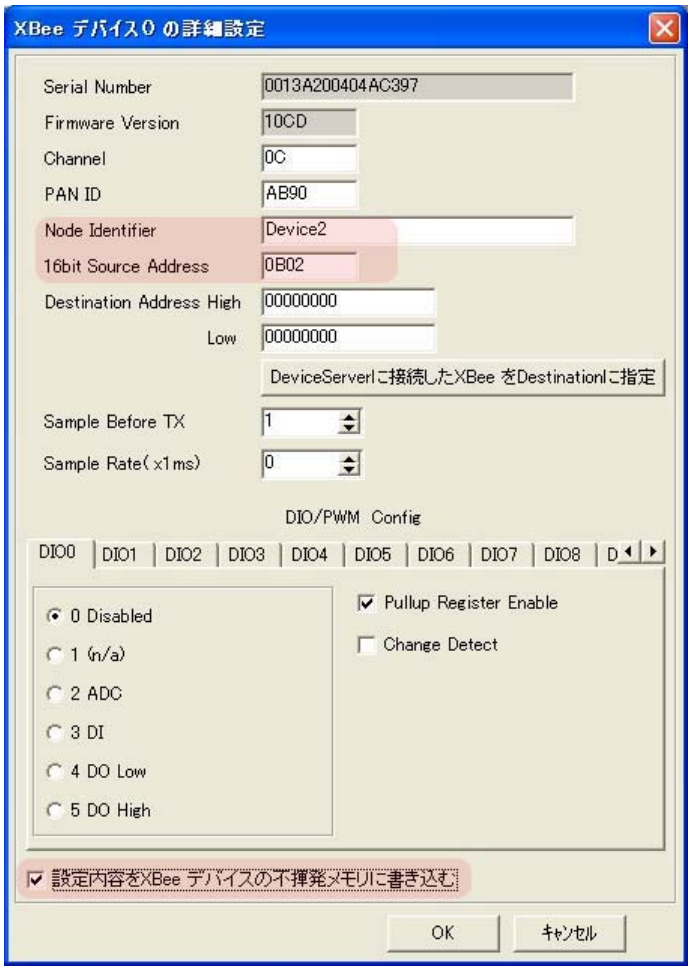
- **[Step2/3] XBeeデバイス詳細設定を変更**

XBee デバイスの詳細設定を変更するために、XBee デバイス管理プログラムのデバイス一覧から対象デバイスを選択して、”設定変更” ツールボタンを押します。初期設定時に 16 bit Source Address を設定した場合は、その値がデバイス一覧に表示されていますので、変更対象の XBee デバイスを確認することができます。

選択したXBee デバイスと通信を行って、現在のデバイス情報を取り込みます。
もしエラーが発生した場合は、選択したXBee デバイスとの間で通信ができない状態になっていますので、通信経路や電源等を確認してください。

XBee デバイスの現在の設定値を取り込んだ後、詳細設定変更ダイアログを表示します。
ここでは、XBee デバイスの Node Identifier の設定を行って下さい。Node Identifier に指定可能な文字は ASCII で 20文字までです。また、ダイアログに表示されている 16 bit Source Address が、対象のデバイスであるかどうかの確認も行ってください。ここで 16 bit Source アドレスを任意の値に変更することも可能です。

最後に、”設定内容を XBee デバイスの不揮発メモリに書き込む” にチェックを付けて “OK” を押してください。



その他の設定項目については “クライアントソフトウエア”の章の “XBee デバイス管理(XBeeConfig)”の項を参照してください。

- **[Step3/3] XBeeデバイス新しい設定に合わせて、マスタを更新する**

[Step2/3] でデバイスの Node Identifier または 16bit Source Address を変更した場合は、XBee デバイス管理プログラムのデバイス一覧に表示されている、マスターファイルも更新しておく必要があります。

[Step1/3] と同様に XBee デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

デバイスのNode Identifier または 16bit Source Address以外の詳細設定を変更する場合には、マスターファイルの更新は必要ありません。

XBee デバイスの削除を行う場合は、XBee デバイス管理プログラムで削除対象のデバイスを選択した後、”削除”ボタンを押してください。マスターファイルから XBee デバイスが削除されます。

8.5 XBee-ZB シリーズ 2 デバイス

Digi international Inc.社製の ZigBee 対応無線通信デバイスです。デジタル入力と出力、A/D 変換入力ができます。DeviceServer の COM ポートに XBee-ZB (coordinator)デバイスを接続します。この 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(router,end device)を操作することが可能です。

リモート XBee-ZB デバイス上で発生したイベントを DeviceServer に送信する機能があり、イベントハンドラをスクリプトで記述することができます。同一PAN 内の全ての XBee-ZB デバイス(リモートデバイスを含む)は XBee-ZB デバイス管理プログラム(ZBConfig) で管理することができます。XBee-ZBデバイスの詳細機能はこのプログラムから GUI で設定することができます。

XBee-ZB デバイスには、以下の機能があります。

- DIO0-DIO4　　ポート入力、出力、A/D 変換入力
- DIO5,DIO10-12　ポート入力、出力
- ポート入力値に変化があった場合のI/Oデータフレーム送信
- ポート入力のプルアップ設定

DeviceServer では、これらの XBee-ZB デバイス機能をスクリプトライブラリを使用して操作できます。詳しい XBee-ZB デバイスの機能詳細については Digi international Inc. のドキュメントを参照してください。

注意

XBee-ZB デバイスは下記のファームウエアバージョンで動作確認しています。XBee-ZB で使用するファームウエアを書き込むときには、このバージョン以降のものを使用してください。XBee-ZB デバイスのファームウエア更新は Digi international Inc.社の X-CTU プログラムを使用することで可能です。詳しいファームウエアの仕様と更新方法については XBee デバイスの購入元もしくは Digi international Inc.社のドキュメントを参照して下さい。

DeviceServer接続用 XBee-ZB ZIGBEE COORDINATOR API firmware version "21A7"

リモート用 XBee-ZB ZIGBEE ROUTER API firmware version "23A7"

リモート用 XBee-ZB ZIGBEE END DEVICE API firmware version "29A7"

8.5.1 XBee-ZB デバイスの初期設定

XBee デバイスを DeviceServer に接続する前に、XBee デバイス自身の初期設定を行ってください。設定を行うデバイスは、DeviceServer に直接接続する XBee-ZB デバイスとリモート側の全ての XBee-ZB 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以外のXBeeデバイスの設定項目は、後からXBee管理プログラムで操作することができます。ここで説明する XBee-ZB デバイスの初期設定は XBee-ZB デバイスを最初に使用するときに一度だけ必要です。初期設定が終了して ZigBee ネットワークに接続できるようになると、後は DeviceServer の 管理プログラム(ZBConfig) からリモート操作で XBee-ZB の詳細パラメータを変更できます。

XBee-ZB には ZigBee デバイスタイプに対応して、Coordinator, Router, End device の３種類のファームウエアが

あります。DeviceServer の COM ポートには Coordinator タイプのファームエアを書き込んだ XBee-ZB デバイスを接続します。リモート側の XBee-ZB デバイスは Router または End Device タイプを選択して書き込みます。

XBee-ZB デバイスのデフォルト値から変更が必要な項目は以下になります。

XBee-ZB 設定項目	設定値
ファームウエアの書き換え	DeviceServer のCOMポートに直接接続するデバイスは、ZIGBEE COORDINATOR API を使用します。 その他リモート側のデバイスは ZIGBEE ROUTER API または ZIGBEE END DEVICE API を使用してください。
API モード	1（ ZIGBEE ****** API タイプのファームエア書き換え直後は 1 に設定されていますが、念のため確認してください）
PAN(Personal Area Network) ID	任意の値 (default は0) 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 Identifier	同一PAN ID 内でユニークな文字列 (default は "") この値は、後から ZB管理プログラムで設定・変更することも可能ですが、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。デバイスを選択するときにわかり易いように全てのデバイスで異なった名前を設定してください。例えば、"NODE01", "NODE02" 等。

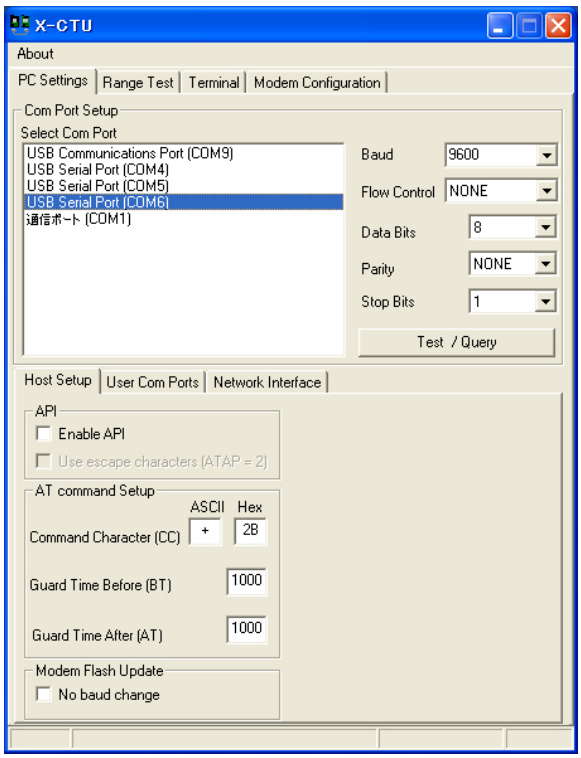
初期設定のコマンドをXBee に送信するために、XBee-ZB デバイスを PC のCOM ポートに接続して Digi international Inc. 社製の X-CTU プログラムを使用します。XBee-ZB と COM ポートのボーレートは初期値の 9600bps を使用します。最初にリモート側のXBee-ZB デバイスを全て設定した後に、サーバーに接続するXBee-ZB を設定します。

このマニュアルでは、Sparkfun Electoronics 社製の XBee Explorer USB を使用して、仮想 USB ポート経由で接続した例で説明します。これ以外の方法で COM ポート接続する場合も手順は同じです。
既に、XBee Explorer USBのドライバ等がインストール済みで仮想 USB ポートが作成されているものとします。(例では COM6)

8.5.2 リモート側 XBee-ZB デバイス初期設定 (ZigBee ROUTER)

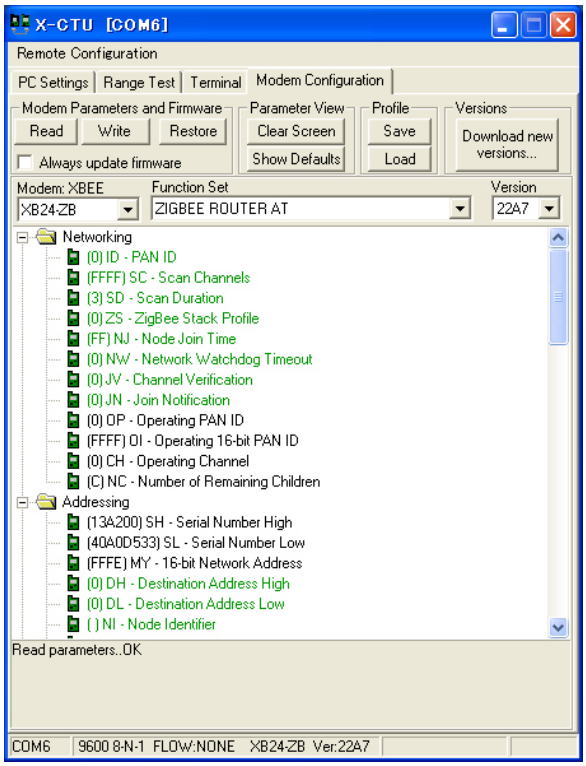
ここでは最初に、リモート側デバイス用に ZigBee デバイスタイプ "router" のXBee-ZB デバイスを最初に設定する例を示します。

X-CTU プログラムを起動して、COM ポートを選択します。ここでは、USB Serial Port(COM6) を選択しています。"Host Setup"タブ中の "API 設定" のチェックボックスは、最初にデバイスを接続するときにはチェックを外してください。XBee-ZB 購入直後には `ZIGBEE ******* AT` タイプのファームウエアがロードされています。後で `ZIGBEE ****** API` タイプのファームウエアを書き込んだ後には、 "API 設定" のチェックを入れて設定値の読み込みと書き込み操作をします。



(X-CTU プログラムで XBee-ZB デバイスへの接続条件を設定している画面)

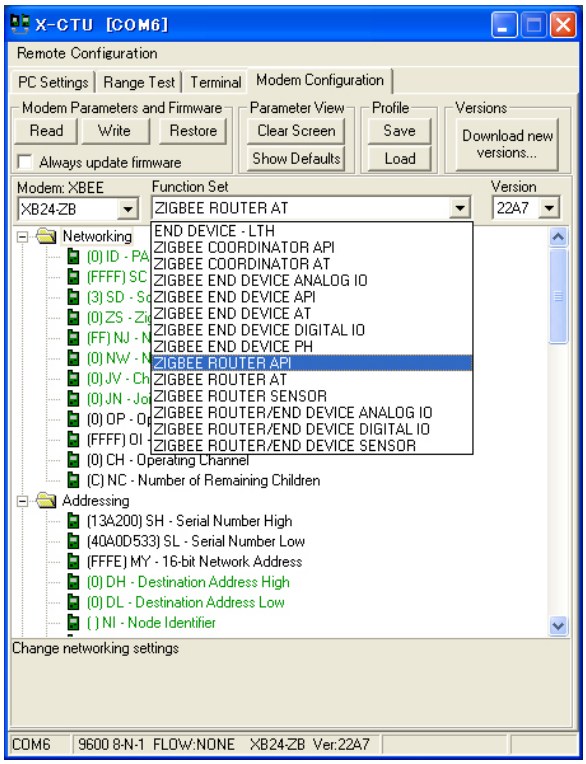
"Modem Configuration" タブを選択して、現在の XBee-ZB デバイスの設定をリードします。これは COM ポートの通信が正常に行えるかの確認も兼ねています。



(XBee-ZB デバイスの工場出荷時のデフォルト設定を読み込んだ様子)

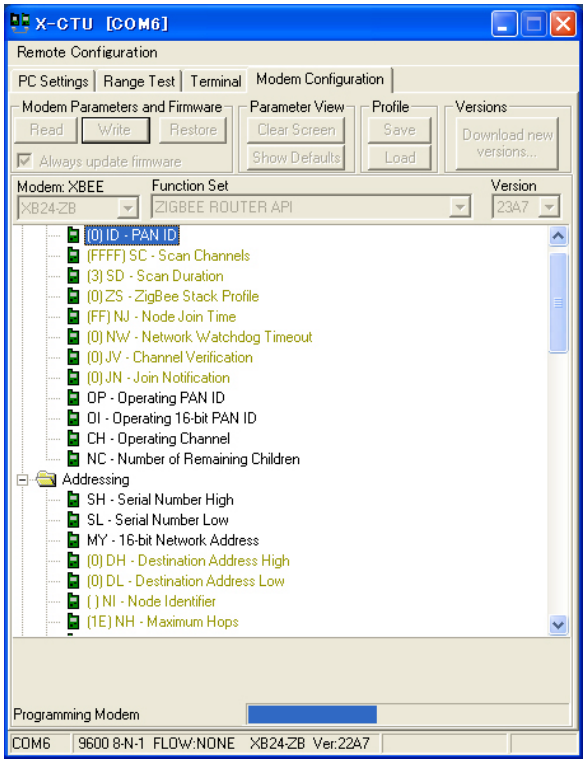
XBee-ZB デバイスの初期設定は AT モードのファームエアがロードされていますのでこれを、書き換えます。

"Function Set" プルダウンメニューから `ZIGBEE ROUTER API` を選択します。



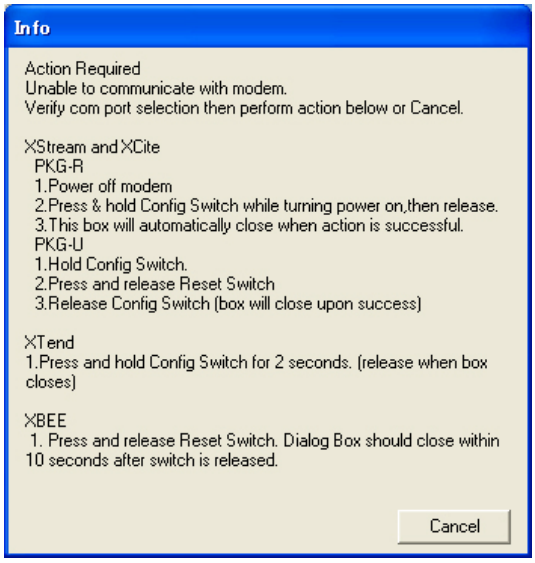
"Always update firmware" チェックボックスにチェックをつけて、"Write" ボタンを押してファームウエアの書き

込みをします。

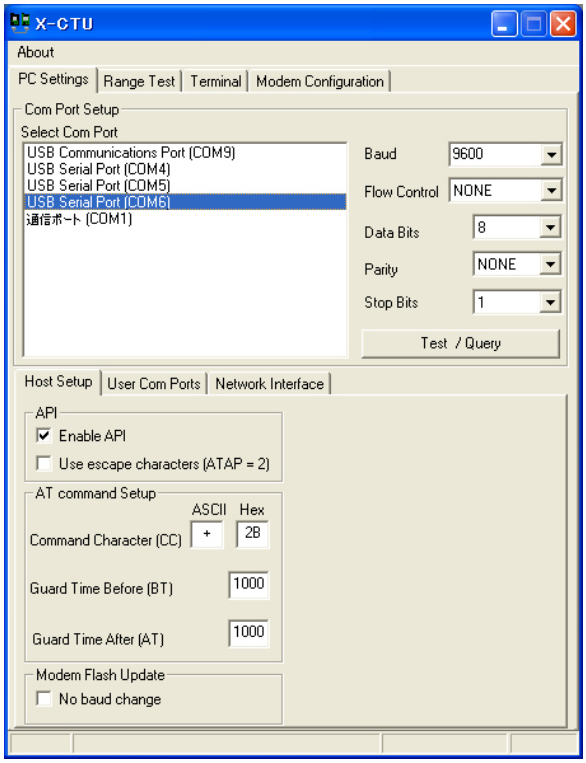


（ファームエア書き込み中の X-CTU の画面）

ここでファームエア書き込み終了後に、下記の様な画面が表示される場合があります。

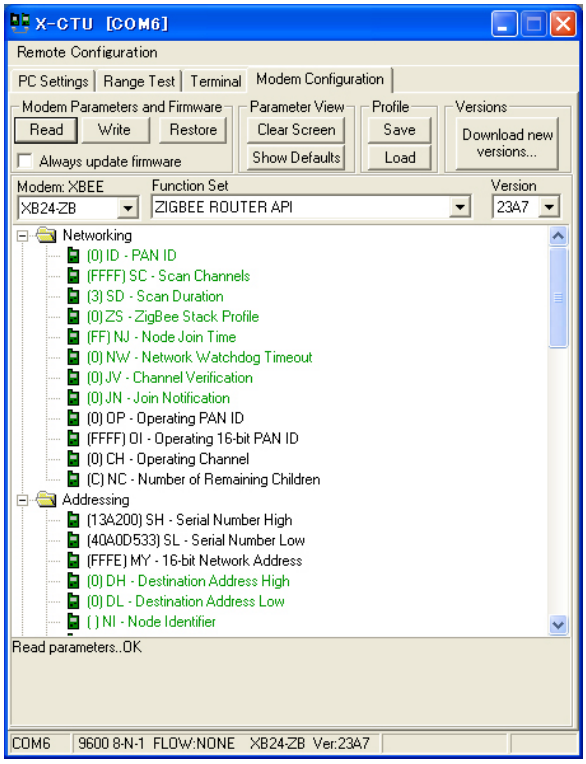


これは、ファームエアが API モード用のものに変わったために、最初にX-CTU プログラムの通信条件設定画面で "Enable API" のチェックを外していた状態と矛盾するようになったのが原因ですので問題ありません。ここでは "Cancel" ボタンを押してダイアログを消してください。ここで 一旦 X-CTU プログラムを終了して、もう一度同じ X-CTU プログラムを起動してください。



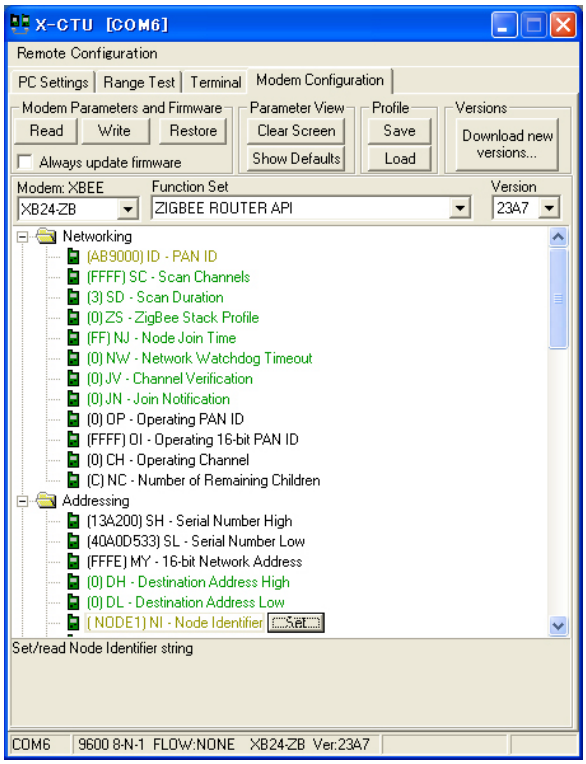
XBee-ZB が接続されたCOM ポートを選択して、今度は "Enable API" にチェックを付けます。

"Modem Configuration" タブを選択して、"Read" ボタンを押してファームウエア書き込み後の XBee-ZB デバイスの設定値をリードします。



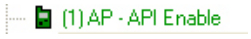
ここから、DeviceServer でこの XBee-ZB デバイスを使用するためのパラメータを設定します。最初に "PAN ID" 項目を選択して、任意の 最大64bit幅の 16進数値を設定します。ここでは "0xAB9000" を設定しています。PAN ID は全ての XBee-ZB デバイスで同一の値を設定してください。次に "Node Identifier" に任意のアルファベット文字列

を設定します。ここでは "NODE1" を設定しています。XBee-ZB デバイス毎に異なった名前をつけてください。

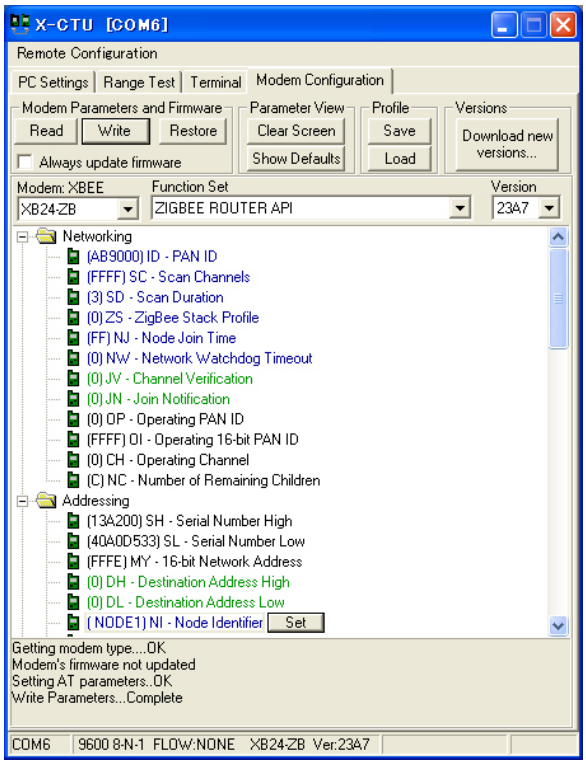


(PAN ID と Node Identifier を設定している様子)

また、API モード値がデフォルト値の 1 になっていることも確認してください。設定項目をスクロールして設定値を表示できます。



設定値を XBee-ZB デバイスに書き込むために、"Write" ボタンを押します。



(設定値を XBee-ZB デバイスに書き込んでいる様子)

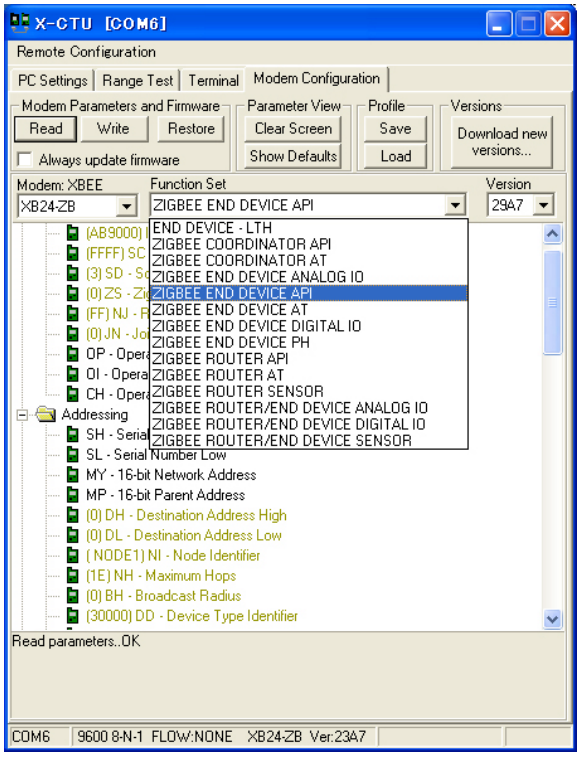
これでリモート側の XBee-ZB デバイスの初期設定が終了しました。

X-CTU プログラムを終了させてから XBee Explorer USB を PC から取り外します。XBee Explorer USB の XBee ソケットから XBee-ZB デバイスを取り外します。

ZigBee Routerタイプで使用する XBee-ZB デバイスが他にもある場合には、XBee Explorer USB の XBee ソケットに接続して同様の手順で初期設定してください。このとき各々の "Node Identifier" の設定値に、区別しやすい別々の名前を付けてこれをメモした紙をデバイスに添付しておくことをお勧めします。

8.5.3 リモート側 XBee-ZB デバイス初期設定 (ZigBee END DEVICE)

リモート側デバイス用に ZigBee デバイスタイプ "end device" のXBee-ZB デバイスを最初に設定する場合も、前の項で説明した "Router" の手順とファームウエアの選択以外は同じです。ファームエアを最初に書き込むときに、"Function Set" プルダウンメニューから `ZIGBEE END DEVICE API` を選択します。



(ZigBee End Device 対応のファームウエアを選択している様子)

ファームウエア書き換え後に、PAN ID と Node Identifier パラメータを前項の "Router" で説明している手順で設定してください。

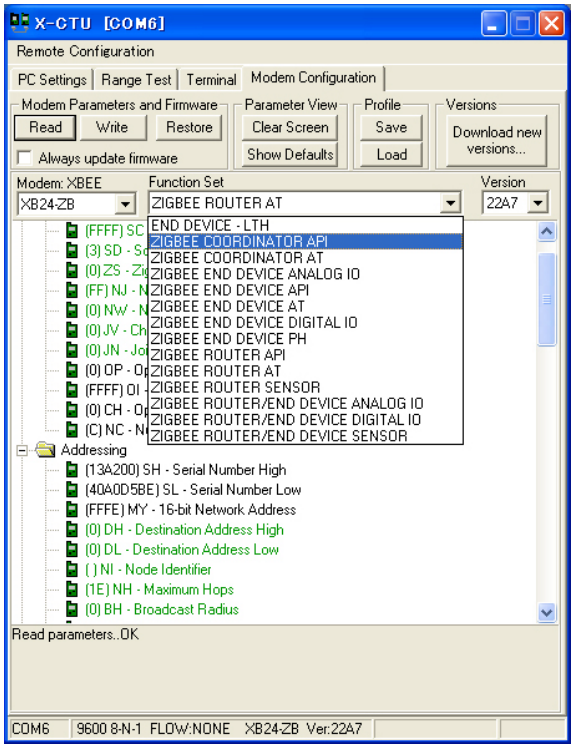
ファームエアを `ZIGBEE END DEVICE API` に設定した場合には、XBee-ZB デバイスはスリープモードが "Cyclic sleep enabled" に設定されます。このときXBee-ZB は定期的にスリープ状態に入って、シリアル信号やRF データの到着時にのみ一定期間ウェイクアップして消費電力をおさえることができます。

ファームウエアのデフォルトのスリープ設定値のまま使用する場合には、DeviceServer からリモート操作でパラメータを変更したり、X-CTU を使用した XBee-ZB のシリアルポート経由のパラメータの変更も問題なく行えます。

スリープパラメータを変更してスリープに入るまでの期間やポーリングのタイミング等を変更すると、リモート操作やシリアルポート経由の通信が難しくなる場合があります。スリープパラメータを変更するときには十分注意してください。

8.5.4 サーバー側 XBee-ZB デバイス初期設定 (ZigBee COORDINATOR)

リモート側 XBee-ZB デバイスの初期設定が終了したら、最後にサーバー PC に接続する XBee-ZB デバイスの初期設定を行います。サーバーに接続する XBee-ZB は ZigBee coordinator として動作させます。設定方法は前の項で説明した "Router" の手順とファームウエアの選択以外は同じです。ファームエアを最初に書き込むときに、"Function Set" プルダウンメニューから `ZIGBEE COORDINATOR API` を選択します。



(ZigBee coordinator 対応のファームウエアを選択している様子)

ファームウエア書き換え後に、PAN ID と Node Identifier パラメータを前項の "Router" で説明している手順で設定してください。

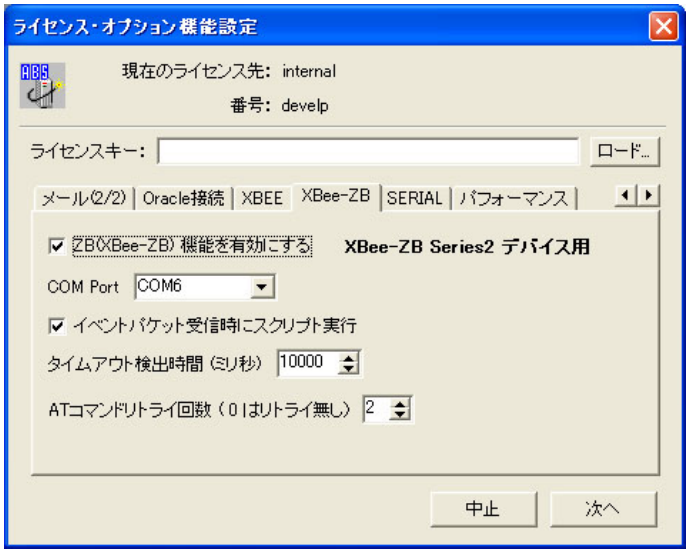
設定が終了したら、X-CTU プログラムを終了します。サーバーPC に XBee Explorer USB 経由で接続した XBee-ZB デバイス(Coordinator) は、接続した状態のまま引き続きDeviceServer で使用します。

8.5.5 XBee デバイスのDeviceServer接続方法

初期設定が終了した XBee-ZB デバイス(Coordinatorファームエア) が PC に接続されているのを確認したら、

DeviceServer から XBee デバイスを使用可能にするために COM ポートの設定を行います。

サーバー設定プログラムを起動して、サービスモジュール設定画面の "ZB" タブ(下の図を参照) を選択します。この画面で、"ZB 機能を有効にする" にチェックをつけてください。"COM Port" に XBee Explorer USB で作成された COM ポートを選択します。"イベントパケット受信時にスクリプト実行" にもチェックを付けて、その他の設定値についてはデフォルト値のままにしておきます。



(DeviceServer の XBee-ZB サービスモジュール設定画面)

サーバー設定プログラムの "次へ" を押して"完了"ボタンが表示されるまで進めて設定を完了して下さい。

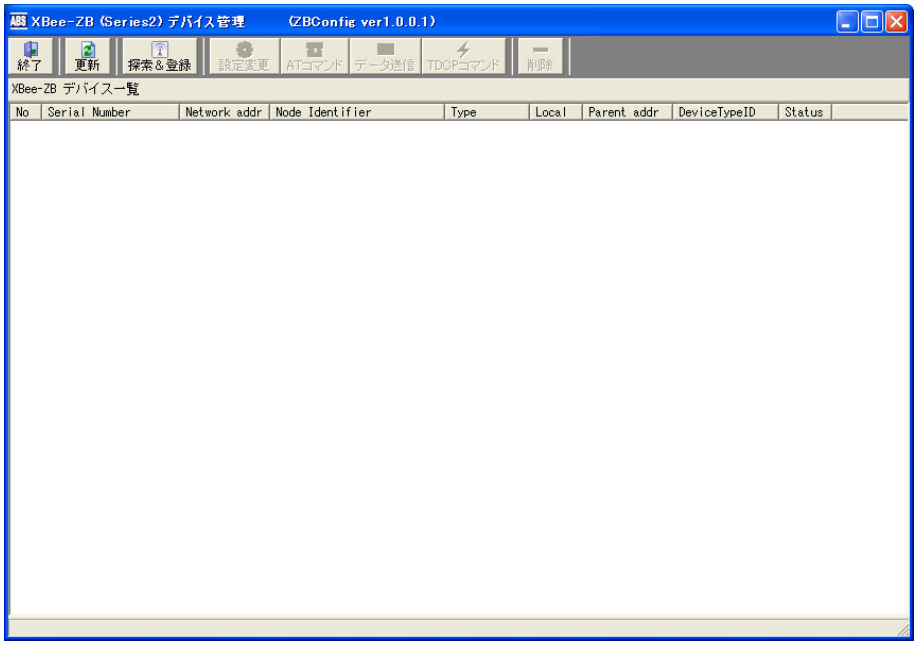
8.5.6 XBee-ZB デバイス設定内容変更

XBee-ZB デバイス管理プログラム(ZBConfig)を使用して、同一 PAN ID に設定された XBee-ZB デバイスを DeviceServer に登録したり、XBee-ZB デバイスの詳細設定をリモートから変更することができます。

プログラムメニューから "ALL BLUE SYSTEM" -> "クライアント起動"を選択・実行します。ログインするときは、管理者特権をもったユーザー (例えば DeviceServer セットアップ時に管理者アカウントとして登録したユーザー等) でログインしてください。

(DeviceServerにログインしてデスクトッププログラムを起動した様子)

デスクトッププログラムが起動したら、 "XBee-ZB" ツールボタンを選択して XBee-ZB デバイス管理プログラムを起動します。



DeviceServer では 登録済みの XBee-ZB デバイスをマスターファイルに記録しています。

XBee-ZB デバイス登録は、”探索＆登録” ボタンを押すことで、同一 PAN ID のリモートデバイスを見つけて、自動的にマスターファイルに登録します。既に、登録済みのXBee-ZB デバイスの場合は最新の情報でマスターファイルの内容を更新します。DeviceServer に COMポートで直接接続された XBee-ZB デバイスについても同様に自動登録します。

一度、マスターファイルに登録された XBee-ZB デバイスは、”削除” ボタンを押して、デバイス登録を削除しない限りマスターファイルに保存されたままで、デバイスリストにも常に表示します。

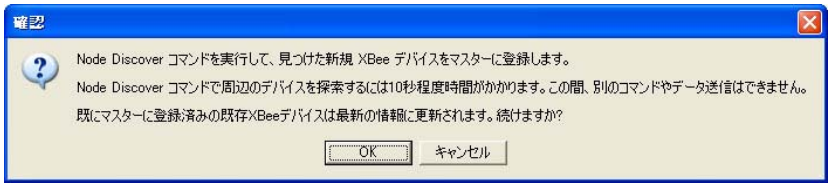
デバイスが登録されたら、デバイスを一覧から選択して”設定変更”ボタンを押すことで、XBee-ZB デバイスの詳細設定を変更することができます。ここで、XBee-ZB デバイスの初期設定時に設定した内容を含む、全てのデバイス設定を自由に変更することができます。Node Identifier も新しい名前に変更することができます。

以下に、XBee-ZB デバイスの登録・詳細設定ステップを順に説明します。

- **[Step1/3] XBee-ZBデバイス探索＆登録を行う**

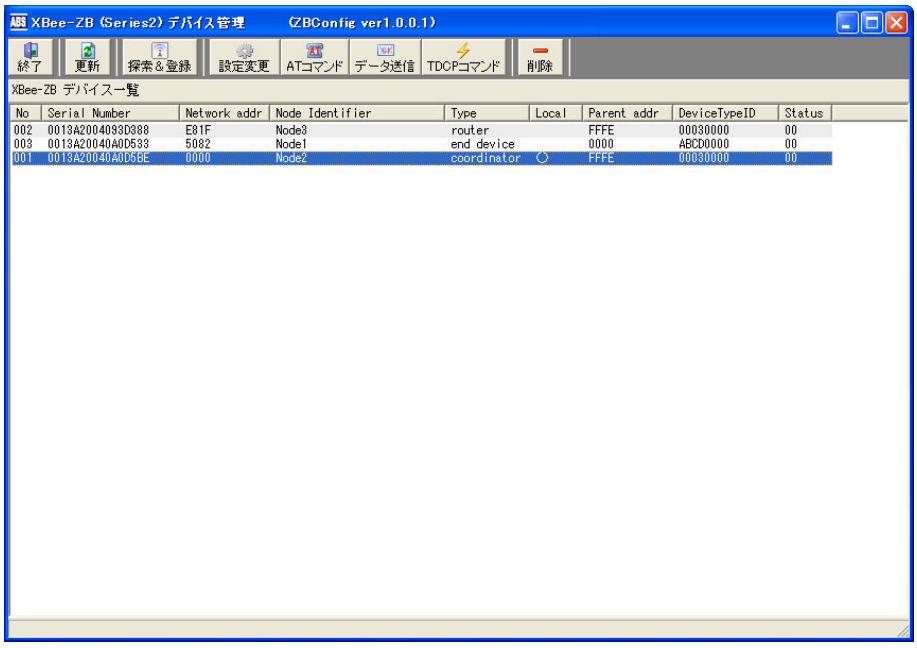
XBee-ZB デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

以下の登録確認ダイアログが表示されますので、”OK” を押します。



(XBee-ZB デバイスを探索・登録するときに表示される確認画面)

DeviceServer の COM ポートに直接接続された XBee-ZB デバイス(Coordinator) で “Node discover” 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。XBee-ZB デバイス管理プログラムのデバイス一覧には、登録済みのXBee-ZB デバイスを表示します。



(デバイスの探索＆登録が完了したときの画面)

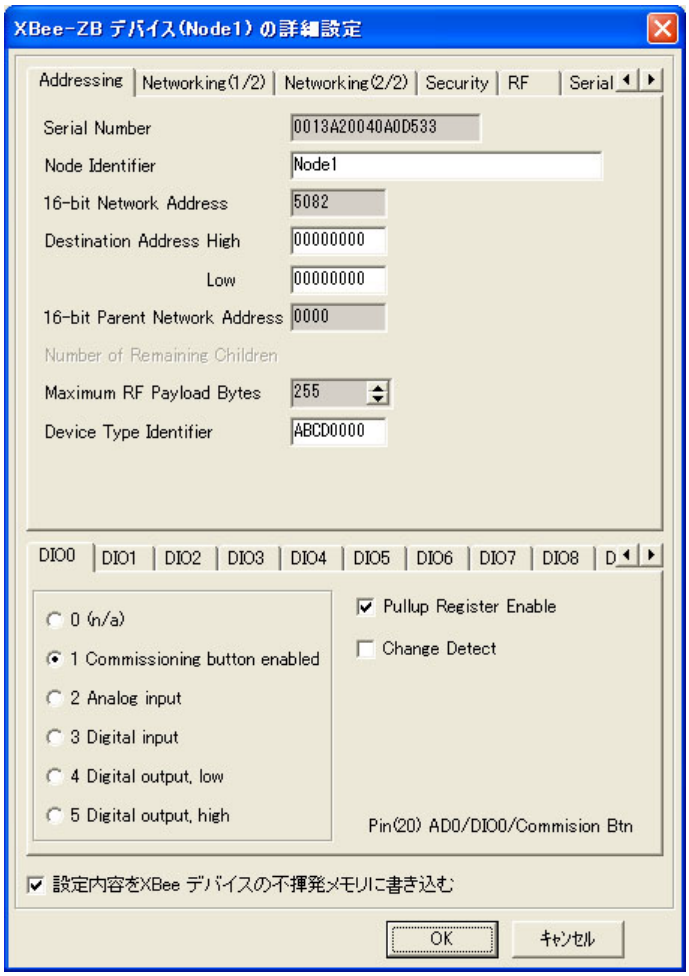
- **[Step2/3] XBee-ZBデバイス詳細設定を変更**

XBee-ZB デバイスの詳細設定を変更するために、XBee-ZB デバイス管理プログラムのデバイス一覧から対象デバイスを選択して、”設定変更” ツールボタンを押します。初期設定時に Node Identifier を設定した場合は、その値がデバイス一覧に表示されていますので、変更対象の XBee デバイスを確認することができます。

選択したXBee-ZB デバイスと通信を行って、現在のデバイス情報を取り込みます。
もしエラーが発生した場合は、選択したXBee-ZB デバイスとの間で通信ができない状態になっていますので、通信経路や電源等を確認してください。

XBee-ZB デバイスの現在の設定値を取り込んだ後、詳細設定変更ダイアログを表示します。
ここでデバイスのパラメータを変更することができます。初期設定時に書き込んだ Node Identifier の値もここで

変更することができます。画面からパラメータを変更したときは、”設定内容を XBee デバイスの不揮発メモリに書き込む” にチェックを付けて “OK” を押してください。



(XBee-ZB デバイスのパラメータ変更画面)

詳しい設定項目の内容については “クライアントソフトウエア”の章の “XBee-ZB デバイス管理(ZBConfig)”の項を参照してください。

- **[Step3/3] XBee-ZBデバイス新しい設定に合わせて、マスタを更新する**

[Step2/3] でデバイスの Node Identifier または DeviceTypeID を変更した場合は、XBee-ZB デバイス管理プログラムのデバイス一覧に表示されている、マスターファイルも更新しておく必要があります。

[Step1/3] と同様に XBee デバイス管理プログラムの “探索＆登録” ツールボタンを押します。

デバイスのNode Identifier または DeviceTypeID以外の詳細設定を変更する場合には、マスターファイルの更新は必要ありません。

XBee-ZB デバイスの削除を行う場合は、XBee-ZB デバイス管理プログラムで削除対象のデバイスを選択した後、”削除”ボタンを押してください。マスターファイルから XBee-ZB デバイスが削除されます。

8.6 アラームシグナルプログラム(AlarmSignal)

SigSensor と同様のアラーム機能(ランプ出力とブザー音出力のみで I/O は持たない)のソフトウエアです。ネットワークに接続された PC 上で起動して DeviceServerから操作することができます。

DeviceServer が動作しているとは別のPC で起動することもできます。ネットワーク接続された任意の Windows PC(Windows 2000,Windows XP,Windows 7) に AlarmSignal.exe プログラムを任意のフォルダにコピーしてから実行してください。

同一 PC 上に複数の AlarmSignal プログラムを同時に起動するときには、起動パラメータでポート番号を変更して起動してください。このときには、アラーム管理プログラムからは各々別のアラーム(ポート番号が違った)として登録します。

アラームシグナルプログラムは前述の SigSensor, NetUIO と同様にアラーム管理プログラム(AlarmConfig) で管理します。

アラームシグナルプログラムには、以下の機能があります。アラーム機能のみのシンプルなプログラムで、簡単に DeviceServer のアラーム機能を使用することができます。

- 3 種類の警告表示とブザー出力

8.6.1 アラームシグナル 接続方法

アラームシグナルプログラムは、下記の場所にインストールされています。アラームシグナルプログラムの起動を簡単にするために、ショートカットを予め作成しておきます。ショートカットファイルを任意のフォルダやデスクトップ、スタートアップメニュー等に作成してください。

アラームシグナルプログラム(AlarmSignal.exe)のインストール先
C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥AlarmSignal.exe 

次に、作成したショートカットのリンク先の項目を変更して起動パラメータを 2 つ追加します。パラメータの間はスペースで区切ってください。最初のパラメータはアラームデバイスが DeviceServer と通信するときに使用するポート番号です。通常は 27103 を使用します。次の文字列は、アラームシグナルプログラム下部のステータスパネル

に表示する文字列を指定します。

- ショートカットのリンク先 "C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe" 27103 サンプルアラーム

ショートカットをダブルクリックして起動します。パラメータを追加した場合には自動的にオンライン状態になり、DeviceServer からのコントロールを受け付ける状態になっています。左側の上部ランプがオンライン状態を示すボタンでクリックするとオンライン、オフラインの切り替えができます。

同一の PC に複数のアラームシグナルプログラムを同時に起動することができます。このときは必ず起動パラメータのポート番号をそれぞれ違う値にしてください。

詳しいアラームシグナルプログラムの説明は “その他のプログラム”の章中の “アラームシグナル(AlarmSignal)” の項目を参照してください。

8.6.2 アラームシグナル 設定内容変更

DeviceServer 側の設定は SigSensor, NetUIO デバイスの登録時同様に、アラーム管理プログラムから行います。デバイスタイプとして “ALARMSIGNAL” を選択する部分が違うだけで、後は同様の手順で登録できます。

アラームシグナルデバイスの場合は、DeviceServer から IP アドレス以外にもホスト名で設定することができます。また、デバイスの詳細設定はありませんので “機能” ボタンは使用しません。デバイスの IP アドレス（ホスト名）は起動したPC と同一で、ポート番号はショートカットに指定するパラメータで決まります。

同一 PC に複数のアラームシグナルプログラムを起動している場合には、それぞれのインスタンス毎に DeviceServer に対応するデバイスを登録します。そのときに、同一 PC 上ではアラームシグナルプログラムの起動しているポート番号がそれぞれ違う筈ですので、デバイス登録時に指定するポート番号も、アラームシグナルプログラムの起動パラメータで指定したポート番号と値をそれぞれ一致させてください。

8.6.3 アラームシグナル 動作確認

Webブラウザを使用して動作確認ができます。前述の”SigSensor, NetUIO 動作確認” の項目を参照してください。

8.7 MQTT ブローカ接続(MQTTクライアント・エンドポイント)

DeviceServer には、外部に設置された MQTT ブローカに接続するための MQTT クライアント接続機能が用意されています。MQTT クライアント機能を使用すると MQTT ブローカに購読リクエストや任意のメッセージの送信を行ったり、購読中のトピックを受信することができます。サーバー起動時に自動で MQTT ブローカに接続して、Willトピックや Willメッセージ、自動で購読対象にするトピックなどを定義できます。

MQTT クライアント接続は DeviceServer では MQTTクライアント・エンドポイント(以降 エンドポイント)として管理します。ユーザーは好きなだけ(スタンダードライセンスでは最大２つ、エンハンスライセンスでは無制限)エンドポイントを定義して MQTT ブローカとの接続を管理できます。

エンドポイントにはブローカとの接続に必要なホスト名やポート番号の他、MQTT プロトコルのコネクション時に必要な情報を予めサーバー設定プログラムで定義しておきます。サーバー起動時には自動的に全てのエンドポイントの接続が開始されて、DeviceServer 中のスクリプトやイベントハンドラから常にブローカに対して任意のメッセージを送信(PUBLISH) したりブローカで購読中のトピックをエンドポイントを経由して送受信します。

複数の MQTT ブローカに同時に接続するために、それぞれのホストに対するエンドポイントを定義することもできます。また、同一のブローカに複数のエンドポイントを定義して、アプリケーションの機能毎にエンドポイントを分けて通信することで通信経路を分散してパフォーマンスを向上させたり、冗長性を持ったネットワークを構築することができます。

エンドポイントはサーバー起動時に自動的に全ての接続が開始されます。これらの動作はユーザーがカストマイズ可能な様にスクリプトで記述されています。後述の MQTT_CONNECT_ALLスクリプトの項を参照して下さい。同様に、サーバー終了時には全てのエンドポイントで MQTT ブローカに対して DISCONNECT 処理が行われます。これもユーザーがカストマイズ可能な様にスクリプトで記述されています。後述の MQTT_DISCONNECT_ALLスクリプトの項を参照して下さい。

MQTTブローカに対して、サーバー起動時に自動的に購読を開始しするトピックをエンドポイントに定義することができます。ライブラリ関数 mqtt_subscribe() を使用することで、任意のタイミングでリクエストすることもできます。

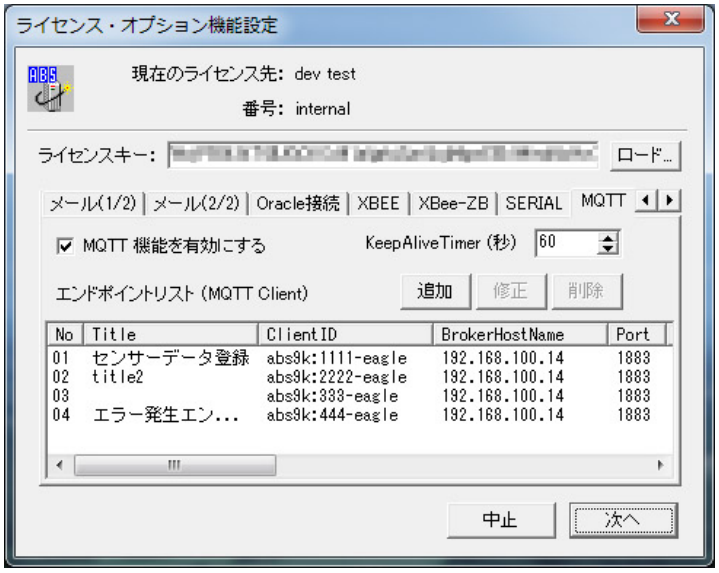
ブローカで購読したトピックのメッセージがエンドポイントに配信された場合には、MQTT_PUBLISH イベントハンドラがメッセージ毎に起動されます。デフォルトではメッセージ内容を文字列(UTF-8)としてデコードしてログに出力するようになっています。ユーザーはこのイベントハンドラをカストマイズすることで配信されたメッセージを処理することができます。詳しくはイベントハンドラの章中の MQTT_PUBLISH の項を参照してください。

MQTT ブローカに接続中の全てのエンドポイントでは、サーバー設定プログラムで設定した KeepAliveTimer 間隔(default 60秒)ごとに MQTT PING メッセージが自動送信されてブローカとのコネクションを維持するようになっています。また、ソケット接続中にエラーを検出した場合にはブローカとの再接続を自動で行うようになっています。これらの動作はユーザーがカストマイズ可能な様にスクリプトで記述されています。詳しくはイベントハンドラの章中の MQTT_KEEP_ALIVE_TIMER の項を参照してください。

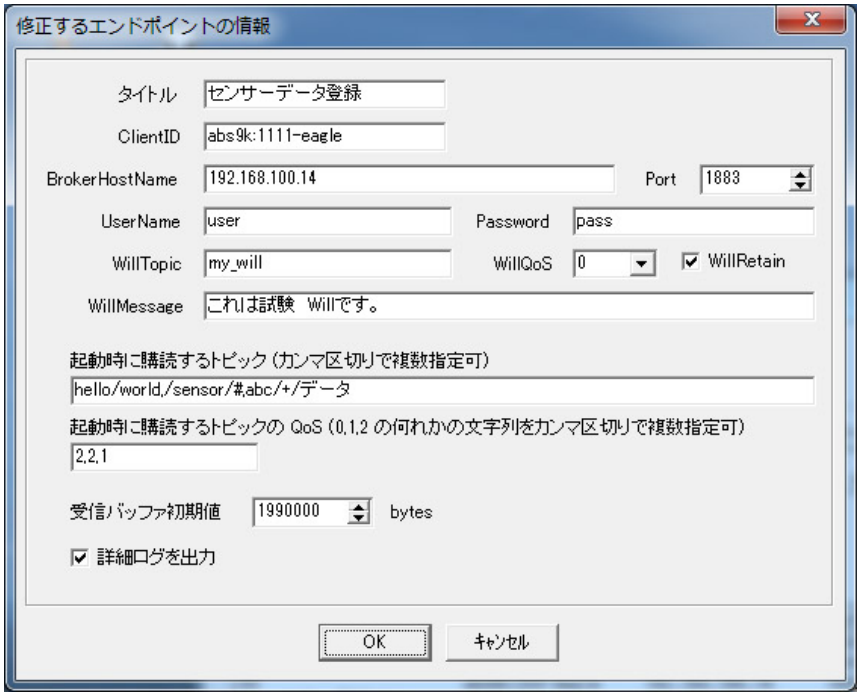
DeviceServerのライブラリ関数には MQTT ブローカに対してトピックメッセージの送信や購読リクエストなどの関数が予め用意されています。また、MQTT プロトコルで定義されたコントロールメッセージ(CONNECT,DISCONNECT,PINGREQ など) を任意のタイミングで送信することもできます。これらのライブラリ関数についてライブラリ・リファレンスの "MQTT クライアントAPI(Lua)" の章を参照してください。

8.7.1 MQTT クライアント・エンドポイントの追加・修正

MQTT ブローカとの接続を行うために予め DeviceServer にエンドポイントを設定します。サーバー設定プログラムを起動して、MQTT タブを選択してエンドポイントの登録や修正、削除を行います。



エンドポイントの追加や修正を行う場合には、エンドポイント設定画面で MQTTブローカとの通信パラメータや MQTT CONNECT リクエストパラメータ、デフォルトで購読を開始するトピックなどを設定します。



個々の設定項目については、”サーバーソフトウエア” の章の “サーバー設定(ServerInit)” プログラムの説明を参照して下さい。

MQTT エンドポイントの設定が終了したら、サーバー設定プログラムの “次へ” を押して”完了”ボタンが表示されるまで進めて設定を完了して下さい。MQTT エンドポイントの設定を変更した場合には、必ず DeviceServer を再起動する必要があります。サーバー設定プログラムでMQTT エンドポイントの設定を変更した場合には、自動的に DeviceServer サービスプログラムは再起動します。

8.7.2 起動時に自動エンドポイント接続 MQTT_CONNECT_ALL

サーバー設定プログラムで作成した MQTT エンドポイントは、DeviceServer 起動時に自動的にブローカに接続を行う様に設定されています。

起動時に DeviceServer 各サービスモジュールが準備できた時点で SERVER_START 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内では MQTT エンドポイントを開始するためのスクリプトをコールしています。下記のスクリプト部分が MQTT_CONNECT_ALL スクリプトをコールしている箇所です。

もし、DeviceServer 起動時に MQTT エンドポイントをブローカに自動接続したくない場合には、下記のスクリプト部分を全てコメントアウトしてください。

```
--[[
*********************************************************************************
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
```

SERVER_START 中の上記スクリプトから別スレッドで実行される MQTT_CONNECT_ALL スクリプトは以下の内容になっています。

```
file_id = "MQTT_CONNECT_ALL"
--[[
●機能概要
登録された全ての MQTT エンドポイントリストの接続を開始する。
●備考
サーバー設定プログラムで登録された MQTT エンドポイントに対して以下の動作を行います。
mqtt_open() を使用してソケット接続
mqtt_connect() を使用してブローカに接続
mqtt_subscribe() を使用して自動購読に指定したトピックを購読
```

```
●変更履歴
2015/05/13          初版作成
ABS-9000 DeviceServer          copyright(c) All Blue System
]]
-----------------------------------------------
-- 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
-----------------------------------------------
-- エンドポイントをブローカに接続
-----------------------------------------------
for key,val in ipairs(id) do
  ---------------------------------------------------------
  -- 現在ソケット接続していないエンドポイントを対象にする
  ---------------------------------------------------------
  if (not enable[key]) then
    ------------------------------
    -- 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
```

8.7.3 終了時に自動エンドポイント切断 MQTT_DISCONNECT_ALL

DeviceServer 終了時には、現在 MQTT ブローカと接続中の全てのエンドポイントに対して切断処理を自動で行う様に設定されています。

DeviceServer 終了直前には SERVER_STOP 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内では接続中の MQTT エンドポイントを切断するためのスクリプトをコールしています。下記のスクリプト部分が MQTT_DISCONNECT_ALL スクリプトをコールしている箇所です。

もし、DeviceServer 終了時に MQTT エンドポイントで接続中のブローカに DISCONNECT メッセージを送信したくない場合には、下記のスクリプト部分を全てコメントアウトしてください。コメントアウトした場合でも DeviceServer 終了時には、接続中の全ての MQTT ブローカへのソケットは正常に削除されます。

```
--[[
*******************************************************************************
MQTT モジュールが有効な場合にエンドポイントの接続を終了させる
*******************************************************************************
]]
local mstat,module_stat = service_module_status()
if not mstat then error() end
if module_stat["MQTT"] then
  script_exec("MQTT_DISCONNECT_ALL","","")
end
```

SERVER_STOP 中の上記スクリプトから実行される MQTT_DISCONNECT_ALL スクリプトは以下の内容になっています。

```
file_id = "MQTT_DISCONNECT_ALL"
--[[
●機能概要
接続中の MQTT エンドポイントリストの接続を終了する。
●備考
サーバー設定プログラムで登録された MQTT エンドポイントで現在 Broker に接続中のもの
```

```
に対して以下の動作を行います。
mqtt_disconnect() を使用して ブローカに切断リクエストを送信
mqtt_clse() を使用してソケット削除
●変更履歴
2015/05/13          初版作成
ABS-9000 DeviceServer        copyright(c) All Blue System
]]
-----------------------------------------------
-- 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
-----------------------------------------------
-- エンドポイントに確立済みのソケット削除
-----------------------------------------------
for key,val in ipairs(id) do
  ---------------------------------------------------
  -- 現在ソケット接続しているエンドポイントを対象にする
  ---------------------------------------------------
  if enable[key] then
    if ready[key] then
      ---------------------------------------------------------------------------
      -- Broker への通信が正常なエンドポイントには DISCONNECT メッセージを送信
      ---------------------------------------------------------------------------
      mqtt_disconnect(id[key])
    end
    ---------------------------------
    -- エンドポイントのソケット削除
    ---------------------------------
    mqtt_close(id[key])
  end
end
```

9 サーバーソフトウエア

DeviceServer のサーバープログラムについて、説明します。

9.1 DeviceServer (ABAppServer)

ABS-9000 DeviceServer のメインサービスプログラムです。Windows のサービスアプリケーションとして動作しています。ユーザーが直接このプログラムを操作することはありません。通常は、API ライブラリやスクリプトライブラリを通して、ABAppServer サービスプログラムを操作します。

DeviceServer がセットアップされた PC が起動されると同時に、ABAppServer サービスプログラムは自動的に開始され、ユーザーからのリクエストを受付可能な状態になります。PC がシャットダウンする場合は、自動的に全コネクションが切断されてサービスプログラムは終了します。Windows にログインしていない状態でも動作していますので、リモートからWindows を再起動した場合にも自動的にサーバーが動作した状態になります。コマンドプロンプトの NET コマンドでサービスの起動や終了を指示することもできます。

Windows サービス名	ABAppService
プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABAppServer.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABAppServer.exe
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
設定ファイル名	C:¥Program Files¥AllBlueSystem¥ABAppService.xml Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABAppService.xml

上記設定ファイルは、通常はサーバー設定プログラムから修正を行いますのでエディタ等で変更しないで下さい。サービスプログラム不具合の原因になります。

ABAppServer プログラムで使用するネットワークリソースについては、”インストール”の章の “動作環境” の項目を参照してください。

9.2 ログサービス(ABLogServer)

ABS-9000 DeviceServerで出力した全てのログ情報を記録しています。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Windows のサービスアプリケーションとして動作しています。ユーザーが直接このプログラムを操作することはありません。サーバー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
```

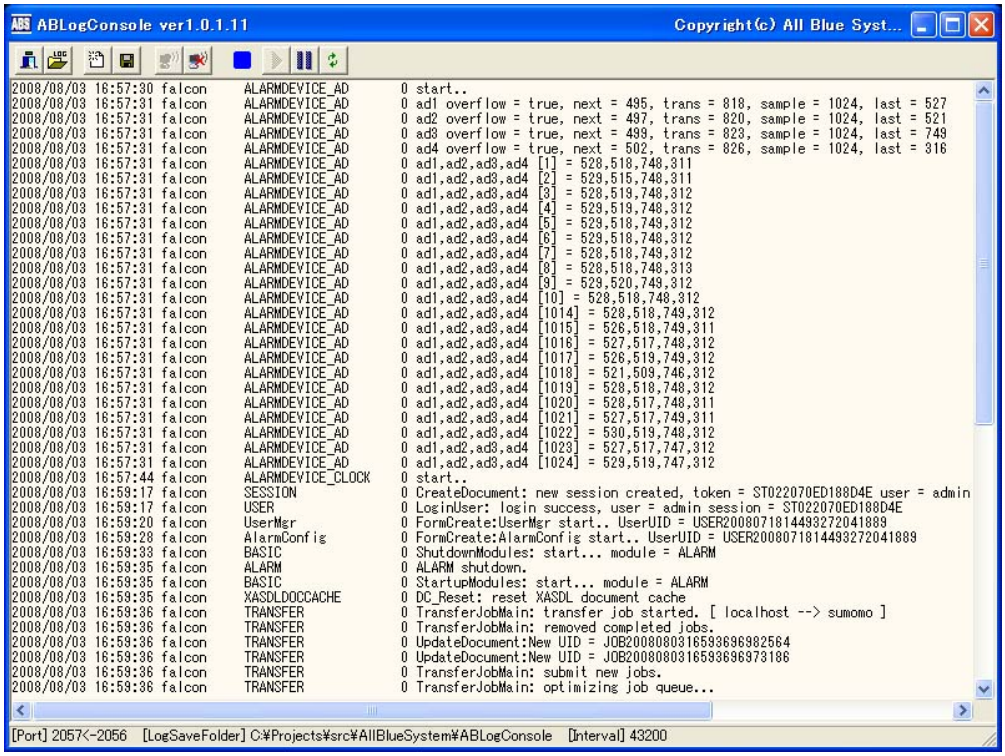
9.3 ログ管理コンソール(LogConsole)

DeviceServer 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ログサービスが停止している時には使用できません。

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。

9.4 サーバー設定(ServerInit)

DeviceServer の動作するPC で起動させて、ライセンス設定や詳細機能の設定を変更することができます。設定変更を行うと、DeviceServerが再起動されますので、運用中に使用する場合は注意してください。（接続中の全てのクライアントは切断されます）

設定変更はウィザード形式で順次行われますので、”次へ” ボタンを押すことで自動的に設定変更とサービス再起動が行われます。サーバー設定プログラムは何回でも実行できますので、途中で ”終了” または ”中止” ボタンを押して設定作業を中断した場合にも、後でもう一度サーバー設定プログラムを起動して設定をやり直すことができます。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ServerInit.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ServerInit.exe
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

次に、ウィザード画面毎の設定項目とその説明を行います。

9.4.1 サーバー設定プログラム起動画面

この画面で “次へ” ボタンを押すと、DeviceServer の機能毎の設定画面に移動します。”終了” ボタンを押すと、サーバー設定プログラムを終了します。

また、この画面では DeviceServer の起動と停止操作をすることができます。運用中にデバイスの切り離しを行いたい場合や、エラー等でサーバーを再起動したい場合に使用します。”サーバー停止” ボタンを押すと DeviceServer が停止します。各サービスモジュールのシャットダウンに時間がかかる場合がありますので、ログ管理コンソールプログラムを使用してシャットダウンメッセージを確認することをお勧めします。

（シャットダウン時のログメッセージ例、最後の “ABAppService shutdown.” が出力されると完全にサーバーが停止しています）

```
2011/02/26 08:45:18 falcon      LAST            0 LAST shutdown.
2011/02/26 08:45:18 falcon      TRANSFER        0 TRANSFER shutdown.
2011/02/26 08:45:18 falcon      XBEE            0 XBEE shutdown.
2011/02/26 08:45:18 falcon      MAIL            0 MAIL shutdown.
2011/02/26 08:45:18 falcon      SCRIPT          0 SCRIPT shutdown.
2011/02/26 08:45:18 falcon      UIOUSB          0 UIOUSB shutdown.
2011/02/26 08:45:18 falcon      WEBPROXY        0 ShutdownHTTPServer: stop listening.
2011/02/26 08:45:18 falcon      WEBPROXY        0 WEBPROXY shutdown.
2011/02/26 08:45:18 falcon      CSVIF           0 CSVIF shutdown.
2011/02/26 08:45:18 falcon      ALARM           0 ALARM shutdown.
2011/02/26 08:45:18 falcon      USER            0 USER shutdown.
2011/02/26 08:45:18 falcon      SESSION         0 SESSION shutdown.
2011/02/26 08:45:18 falcon      MASTERS         0 MASTERS shutdown.
2011/02/26 08:45:18 falcon      CONFIG          0 CONFIG shutdown.
2011/02/26 08:45:18 falcon      CONVERT         0 CONVERT shutdown.
2011/02/26 08:45:18 falcon      COUNTER         0 COUNTER shutdown.
2011/02/26 08:45:18 falcon      XASDLDOCCACHE   0 cleanup...
2011/02/26 08:45:18 falcon      BASIC           0 BASIC shutdown.
```

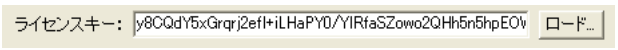
```
2011/02/26 08:45:18 falcon      MessageBackup         0 MessageBackup shutdown.
2011/02/26 08:45:18 falcon      ServiceMain           0 ServiceMain: ABAppService shutdown.      ver1.2.0.414
```

”サーバー起動” ボタンを押すと DeviceServer を起動します。各サービスモジュールのスタートアップに時間がかかる場合がありますので、ログ管理コンソールプログラムを使用して起動メッセージを確認することをお勧めします。(起動時のログメッセージ例、最後の “LAST startup...” が出力されると完全にサーバーの起動処理が完了しています)

```
2011/02/26 08:33:34 falcon      ServiceMain           0 ServiceMain: ABAppService startup.       ver1.2.0.414
2011/02/26 08:33:34 falcon      ServiceMain           0 IBLibraryPath is C:¥Program
Files¥Firebird¥Firebird_2_1¥bin¥fbclient.dll
2011/02/26 08:33:34 falcon      ServiceMain           0 ServiceMain: XASDL instance created. port = 27101
2011/02/26 08:33:34 falcon      MessageBackup         0 MessageBackup startup..(autobackup disable)
2011/02/26 08:33:34 falcon      XASDLDOCCACHE         0 initialize...
2011/02/26 08:33:34 falcon      BASIC                 0 BASIC startup...
2011/02/26 08:33:34 falcon      BASIC                 0 StartupModules: start...
2011/02/26 08:33:34 falcon      XASDLDOCCACHE         0 DC_Reset: reset XASDL document cache
2011/02/26 08:33:39 falcon      COUNTER               0 COUNTER startup...
2011/02/26 08:33:39 falcon      CONVERT               0 CONVERT startup...
2011/02/26 08:33:39 falcon      CONFIG                0 CONFIG startup...
2011/02/26 08:33:39 falcon      MASTERS               0 MASTERS startup...
2011/02/26 08:33:39 falcon      SESSION               0 SESSION startup...
2011/02/26 08:33:39 falcon      USER                  0 USER startup...
2011/02/26 08:33:39 falcon      ALARM                 0 ALARM startup...
2011/02/26 08:33:39 falcon      CSVIF                 0 CSVIF server instance created. port = 27102
2011/02/26 08:33:39 falcon      CSVIF                 0 CSVIF startup...
2011/02/26 08:33:39 falcon      WEBPROXY              0 StartupHTTPServer: listening for HTTP connections on
0.0.0.0:8080.
2011/02/26 08:33:39 falcon      WEBPROXY              0 WEBPROXY startup...(validate session mode)
2011/02/26 08:33:39 falcon      UIOUSB                0 Startup: restarting the COM port, port = COM9
2011/02/26 08:33:41 falcon      UIOUSB                0 UIOUSB startup...
2011/02/26 08:33:41 falcon      SCRIPT                0 PeriodicTimer started
2011/02/26 08:33:41 falcon      SCRIPT                0 SCRIPT startup...
2011/02/26 08:33:41 falcon      MAIL                  0 MailCheck timer started (for mail commands)
2011/02/26 08:33:41 falcon      MAIL                  0 MAIL startup...
2011/02/26 08:33:45 falcon      XBEE                  0 XBEE startup...
2011/02/26 08:33:45 falcon      TRANSFER              0 Startup: transastion timer(timeout) disabled.
2011/02/26 08:33:45 falcon      TRANSFER              0 TRANSFER startup...
2011/02/26 08:33:45 falcon      SERVER_START          0 start..
```

```
2011/02/26 08:33:45 falcon      LAST                    0 LAST startup...
```

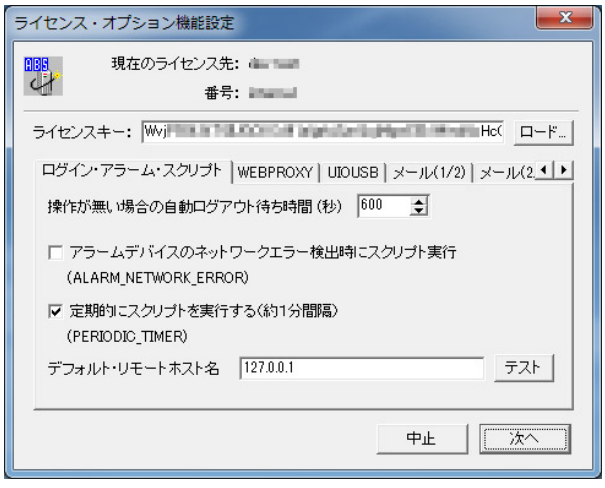
9.4.2 ライセンス・オプション機能設定画面



ここでは、ライセンス入力とDeviceServer の機能毎の設定を行います。項目入力後に “次へ” ボタンを押すと DeviceServer の再起動が行われて新しい設定値が自動的に有効になります。機能毎に設定項目がタブで分かれていますので、各タブ中の項目とその説明を行います。

ライセンス設定（上部入力項目）	
ライセンスキー	正規のライセンスキーもしくはデモライセンスキー文字列を入力する。空白または間違ったキーを入力した場合は DeviceServer はデモモードで動作します。デモモード動作中はサービス起動後 5 時間で DeviceServer は自動停止します。 “ロード” ボタンを押すことで、ライセンスファイル(license.xml) からライセンスキーを読み込んでテキストエリアにロードすることができます。ライセンスファイルはキット製品等を購入された場合に同梱メディア(DVD-R) 中に格納されています。 デフォルト値：””

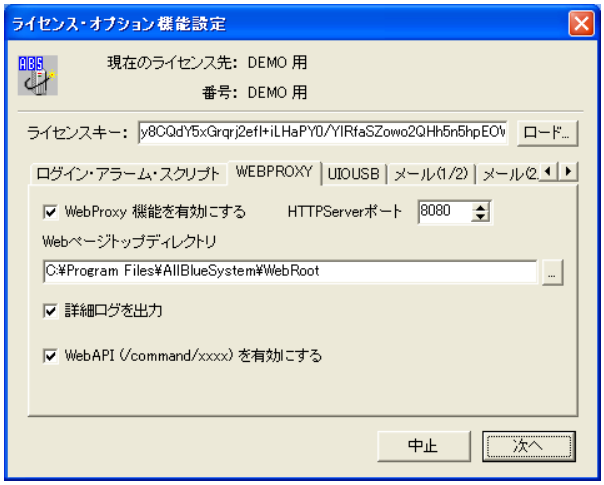
9.4.3 ログイン・アラーム・スクリプト設定タブ



ログイン・アラーム・スクリプト 設定タブ	
操作が無い場合の自動ログアウト待ち時間	ログイン認証を行った後に、設定された時間(秒)を超えてDeviceServerとの通信を行わなかったときに自動的にログインセッションを削除する。

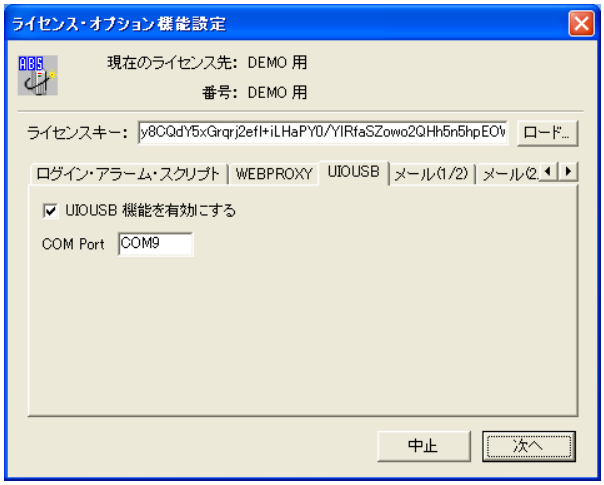
	デフォルト値：600
アラームデバイスのネットワークエラー検出時にスクリプト実行	DeviceServer がアラームデバイスへ通信を行ったときに、最初にネットワークエラーが発生した場合スクリプトを実行する。実行するスクリプト名は“ALARM_NETWORK_ERROR”です。 デフォルト値：OFF
定期的にスクリプトを実行する	一分間隔で定期的にスクリプトを実行する。実行するスクリプト名は“PERIODIC_TIMER”です。 デフォルト値：OFF
デフォルト・リモートホスト名	リモートPC に設置した DeviceServer に対してリクエストを行うライブラリ関数において、"remote_host" 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、ここで指定した IP アドレスまたは "ホスト名" が使用されます。 設定値に "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ので、できるだけ IP アドレスを設定してください。 "テスト" ボタンを押すと、リモートホスト側の DeviceServer で "SAMPLE" スクリプトを実際に実行して、アクセス可能かどうかを試験します。もしエラーになるときは、リモート側 DeviceServer のデスクトッププログラムを起動した後、"許可" ボタンを押してリクエスト元のサーバー名がアクセス許可リストに含まれているかを確認してください。 デフォルト値: 127.0.0.1

9.4.4 WebProxy 設定タブ



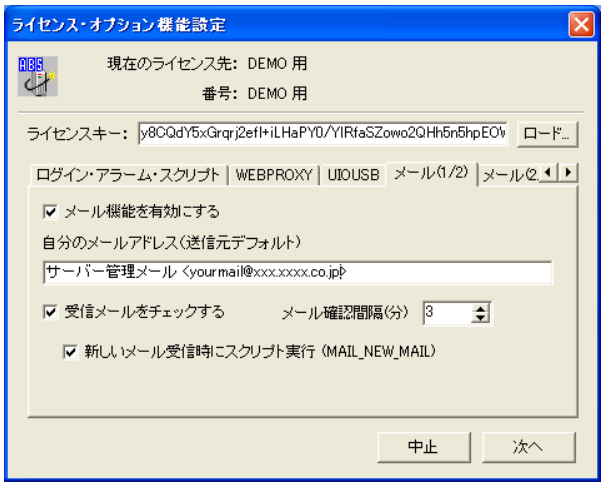
WEB PROXY 設定タブ	
WebProxy機能を有効にする	Webブラウザからアラームデバイス操作やUIOUSB 操作、スクリプト実行を行う機能を有効にする。DeviceServer でHTTP サーバー(Port 80)機能が動作して、Webブラウザ間（実際には Flashプログラム）との通信やセッションを管理します。 デフォルト値：OFF
HTTPServer ポート	WebProxy の HTTP サーバーが使用するポート番号 デフォルト値：80
Webページトップディレクトリ	WebProxy の HTTPサーバー機能で公開するルートディレクトリ。 デフォルト値："C:¥Program Files¥AllBlueSystem¥WebRoot" Windows7(64bit)の場合は "C:¥Program Files (x86)¥AllBlueSystem¥WebRoot"
詳細ログ出力	WebProxy の HTTPサーバー機能のHTTPコマンド処理のログをログサービスに出力する。 デフォルト値：OFF
WebAPI を有効にする	WebProxy サービスを経由して、XMLHttpRequest, JSON形式でDeviceServerの API を使用する場合にチェックを付ける デフォルト値: ON

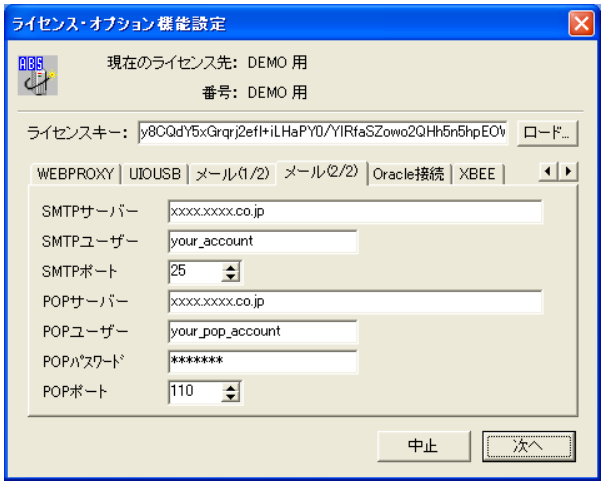
9.4.5 UIOUSB 設定タブ



UIOUSB 設定タブ	
UIOUSB機能を有効にする	USB ポートに接続した UIOUSB デバイスを使用する。UIOUSBデバイスは、予め DeviceServer の動作する PC にセットアップしておく必要があります。 デフォルト値：OFF
COM Port	UIOUSB デバイスをセットアップした仮想COM ポート名。（“COMxx” xx には数字が入る） デフォルト値：””

9.4.6 メール設定タブ

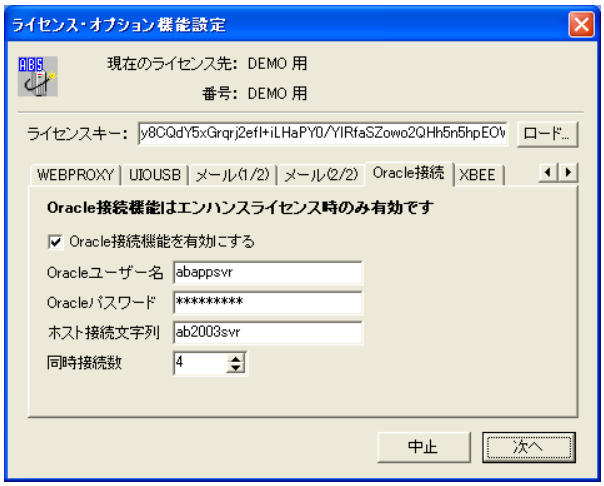




メール 設定タブ	
メール機能を有効にする	DeviceServerでメール受信や送信機能を使用する。 デフォルト値：OFF
自分のメールアドレス	スクリプトからメール送信を行った場合や、メールコマンドのリプライ送信を行うときの “From:” に入るメールアドレス。スクリプトからのメール送信時はパラメータで、ここで設定した値以外の文字列指定も可能。 デフォルト値：”your_mail_address@your.mail.domain”
受信メールをチェックする	POP サーバーに設定されたメールサーバーに新規メールの確認を定期的に行う。メールの件名にメールコマンドに該当するものがあった場合は、メールを取り出した後メールコマンドを実行する。 この設定を ON にした場合には、グロバール共有変数 $MAILBOX_COUNT の値が常に更新されて、最後に受信メールをチェックしたときに POPサーバーに保存されたメール数(文字列形式に変換したもの)が格納される。 デフォルト値：OFF
メール確認間隔	POP サーバーへメールを確認する間隔(分) デフォルト値：5
新しいメール受信時にスクリプト実行 (MAIL_NEW_MAIL)	POP サーバーに新規メールがあった場合に、スクリプトを実行する。実行するスクリプト名は “MAIL_NEW_MAIL” です。 デフォルト値：OFF
SMTPサーバー	メール送信に使用する SMTP サーバーホスト名を入力する デフォルト値：”smtp_server_name”
SMTPユーザー	メール送信に使用する SMTP サーバーユーザー名を入力する デフォルト値：”smtp_account_name”
SMTPポート	メール送信に使用する SMTP ポート番号を入力する デフォルト値：25
POPサーバー	メール受信に使用する POP3 サーバーホスト名を入力する デフォルト値：”pop3_server_name”
POPユーザー	メール受信に使用する POP3 サーバーユーザー名を入力する

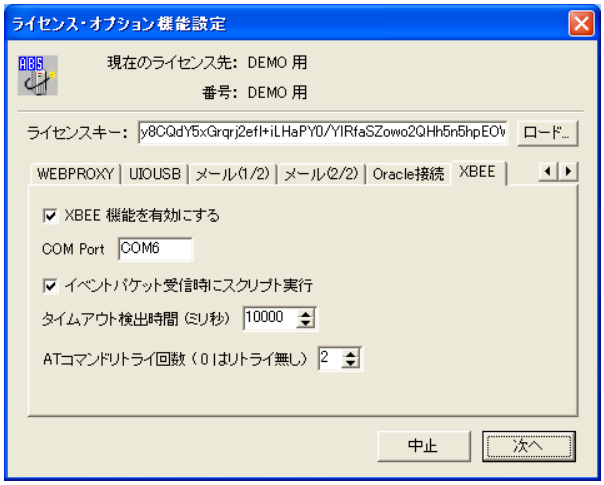
	デフォルト値：”pop_account_name”
POPパスワード	メール受信に使用する POP3 サーバーパスワードを入力する デフォルト値：””
POPポート	メール受信に使用する POP3 ポート番号を入力する デフォルト値：110

9.4.7 Oracle接続設定タブ



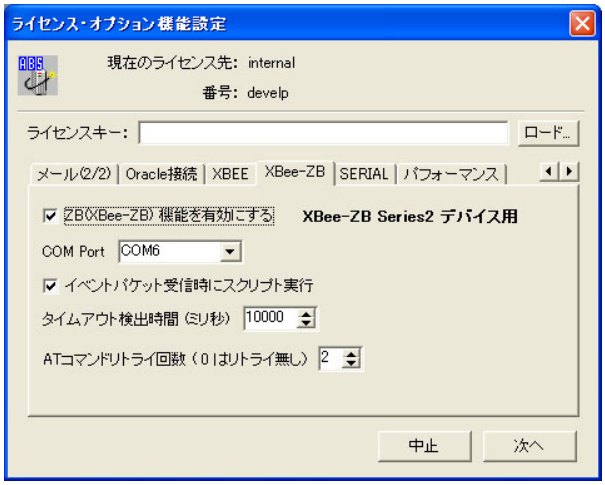
Oracle接続 設定タブ	
Oracle接続機能を有効にする	DeviceServerでOracle データベース接続機能を使用する。 デフォルト値：OFF
Oracleユーザー名	DeviceServer がデータ保存に使用するテーブルをアクセスする為のOracle ユーザー(詳しい内容は “Oracle接続”の章を参照してください) デフォルト値：なし
Oracleパスワード	Oracle ユーザーパスワード デフォルト値：なし
ホスト接続文字列	Oracle データベースサーバーへの接続文字列 デフォルト値：なし
同時接続数	DeviceServer がOracleサーバーに接続するセッション数 デフォルト値：3

9.4.8 XBee設定タブ (XBee 802.15.4 Series1 デバイス)



XBEE 設定タブ	
XBee 機能を有効にする	DeviceServer で XBee デバイス管理機能を使用する。 デフォルト値：OFF
COM Port	XBee デバイスをセットアップしたCOM ポート名。 (“COMxx” xx には数字が入る) デフォルト値：””
イベントパケット受信時にスクリプト実行	DeviceServer に接続した XBee デバイスが、以下の APITypeのデータフレームを受信した場合に、予め決められたイベントハンドラスクリプトを実行する。 対象となる API Typeとイベントハンドラスクリプト名 API Type = 0x80 : Receive Packet 64bit XBEE_PACKET_DATA or XBEE_TDCP_DATA API Type = 0x81 : Receive Packet 16bit XBEE_PACKET_DATA or XBEE_TDCP_DATA API Type = 0x82 : I/O Data Receive 64bit XBEE_IO_DATA or XBEE_IO_OTHER API Type = 0x83 : I/O Data Receive 16bit XBEE_IO_DATA or XBEE_IO_OTHER API Type = 0x8A : Modem Status XBEE_MODEM_STATUS 詳しくは、”イベント”の章を参照してください。 デフォルト値：ON
タイムアウト検出時間(ミリ秒)	DeviceServer に接続した XBee デバイスにリクエストフレームを送信して、レスポンスフレームを受信するまでに待機する最大時間(ミリ秒) デフォルト値: 10000
ATコマンドリトライ回数	DeviceServer に接続した XBee デバイスに AT コマンドを送信して、エラーまたはタイムアウトになった場合に、同一のコマンド送信を再試行する最大回数。 デフォルト値：2

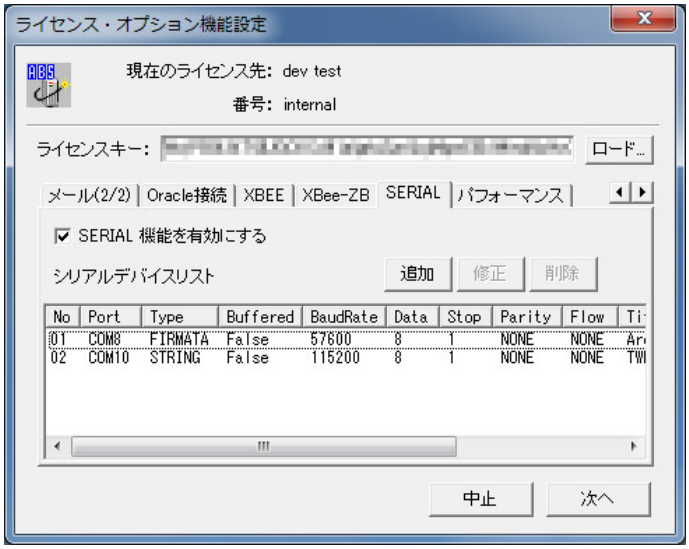
9.4.9 ZB 設定タブ (XBee-ZB Series2 デバイス)



ZB 設定タブ	
ZB 機能を有効にする	DeviceServer で XBee-ZB デバイス管理機能を使用する。 デフォルト値：OFF
COM Port	XBee-ZB デバイス(Coordinator)をセットアップしたCOM ポート名。 (“COMxx” xx には数字が入る) デフォルト値：””
イベントパケット受信時にスクリプト実行	DeviceServer に接続した XBee デバイスが、以下の APITypeのデータフレームを受信した場合に、予め決められたイベントハンドラスクリプトを実行する。 対象となる API Typeとイベントハンドラスクリプト名 API Type = 0x8A : Modem Status ZB_MODEM_STATUS API Type = 0x90 : ZigBee Receive Packet ZB_RECEIVE_PACKET or ZB_TDCP_DATA API Type = 0x91 : ZigBee Explicit Rx Indicator ZB_OTHER API Type = 0x92 : ZigBee I/O Data Sample Rx Indicator ZB_IO_DATA API Type = 0x94 : XBee Sender Read Indicator ZB_OTHER API Type = 0x95 : Node Identification Indicator ZB_OTHER API Type = 0xA1 : Route Record Indicator ZB_OTHER API Type = 0xA2 : Device Authenticated Indicator ZB_OTHER API Type = 0xA3 : Many-to-One Route Record Indicator ZB_OTHER API Type = 0xA4 : Register Joining Device Status ZB_OTHER 詳しくは、”イベント”の章を参照してください。 デフォルト値：ON
タイムアウト検出時間(ミリ秒)	DeviceServer に接続した XBee-ZB デバイスにリクエストフレームを送信して、レスポンスフレームを受信するまでに待機する最大時間(ミリ秒) デフォルト値: 10000
ATコマンドリトライ回数	DeviceServer に接続した XBee-ZB デバイスに AT コマンドを送信して、エラーまたはタイムアウトになった場合に、同一のコマンド送信を再試行する最大回数。

	デフォルト値：2

9.4.10 SERIAL デバイス設定タブ



SERIAL 設定タブ	
SERIAL 機能を有効にする	DeviceServer で SERIALデバイス管理機能を使用する。 デフォルト値：OFF

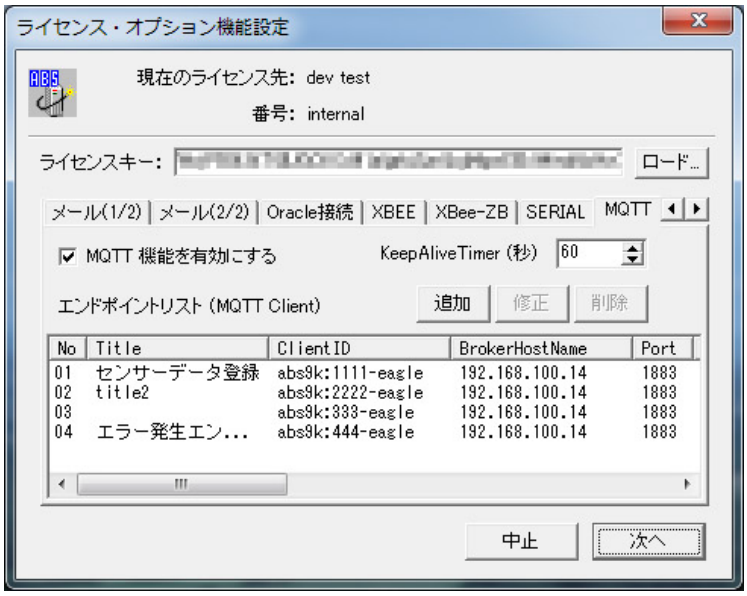
“追加”ボタンを押すと、新規シリアルデバイスの登録ができます。シリアルデバイスリストからデバイスを選択して、“修正”ボタンを押すとシリアルデバイスの設定を変更することができます。

シリアルデバイス（登録・修正）ダイアログ	
COM ポート名	シリアルデバイスを接続するCOM ポート名。 デフォルト値：””
タイトル	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 API 関数のパラメータで、COM ポート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時のパラメータに “Title” が追加されて、ここで設定したタイトル文字列が入ります。 デフォルト値：””
デバイスタイプ	シリアルデバイスの種別 “STRING”：ASCII 文字列や UTF-8 日本語文字列を扱います。文字列の終端を自動で識別して文字列毎にイベント発生または文字列バッファに格納します。 “FIRMATA”：FIRMATA プロトコル(Arduino デバイスの標準ライブラリで定義されたもの）のパケットを自動で識別して、１パケット受信毎にイベント発生または、バッファに格納します。 “RAW”：シリアルポートで受信したデータの固まりでイベント発生または、バッファに格納します。

	"TWE":TWEワイヤレスデバイスのファームウエアで使用されるアスキー形式のデータを扱います。イベントハンドラ中で各アスキー形式を判断してスクリプト中で扱い易い配列やテーブル形式に変換します。 デフォルト値：STRING
バッファー入力を行う	シリアルポートでデータを受信した時に、イベントを発生させるかまたはバッファに格納して、ライブラリ関数で順次取り出すかの区別を設定します。 デフォルト値: False
ボーレート	シリアルポートのボーレート デフォルト値：9600
データビット	シリアルポートのデータ幅 デフォルト値：8
ストップビット	シリアルポートのストップビット幅 デフォルト値：1
パリティ	シリアルポートのパリティチェックの使用 デフォルト値："NONE"
フローコントロール	シリアルポートのフローコントロールの使用 デフォルト値："NONE"

シリアルデバイスリストからデバイスを選択して、"削除"ボタンを押すとシリアルデバイスを削除することができます。

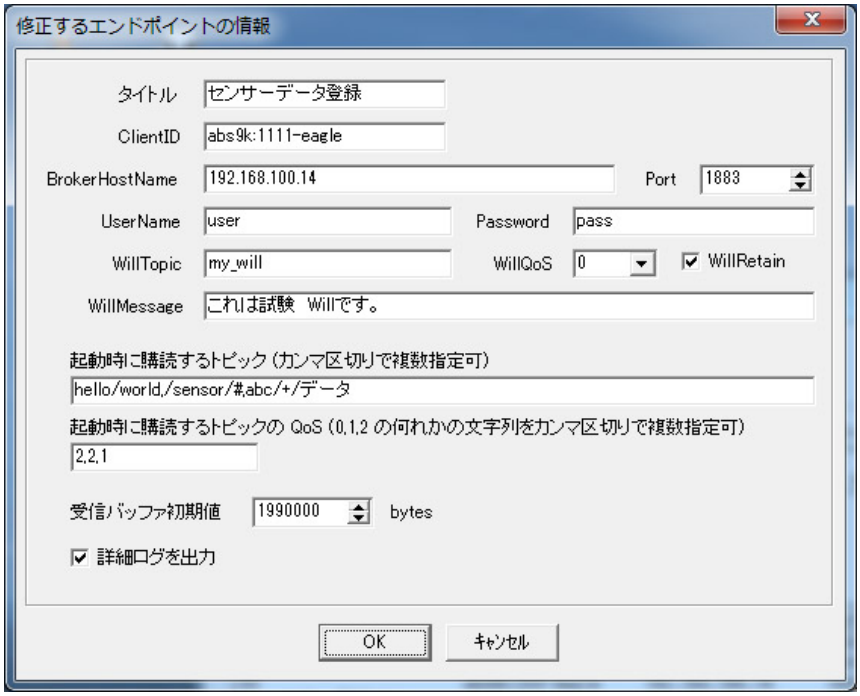
9.4.11 MQTT 設定タブ



MQTT 設定タブ	
MQTT 機能を有効にする	DeviceServer で MQTT クライアント接続を使用する。

	デフォルト値：OFF
KeepAliveTimer	MQTT ブローカに CONNECT メッセージを送信するときの KeepAliveTimer 値を指定する。またここで設定した間隔で 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 スクリプトでは 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。詳しくはイベント章中の "MQTT_KEEP_ALIVE_TIMER" の項を参照して下さい デフォルト値: 60秒

“追加”ボタンを押すと、新規エンドポイントの登録ができます。エンドポイントリストからエンドポイントを選択して、“修正”ボタンを押すとエンドポイントの設定を変更することができます。



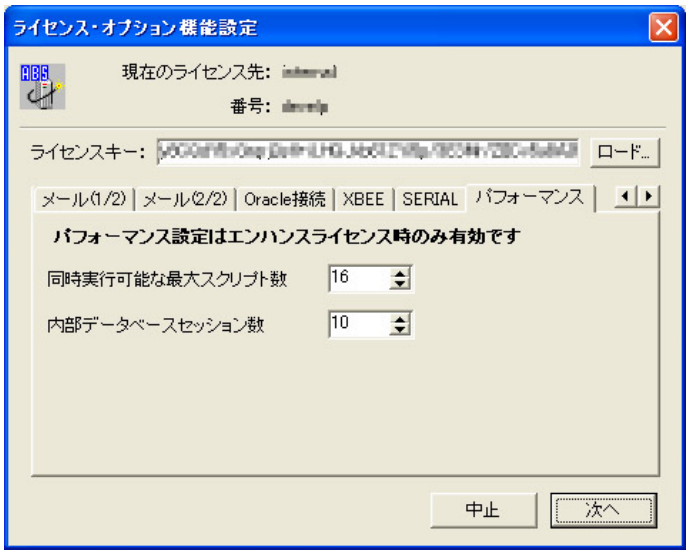
エンドポイント（登録・修正）ダイアログ	
タイトル	エンドポイントに任意のタイトル文字列を設定することができます。 タイトルを指定すると、エンドポイント(MQTT クライアント接続)を操作する API 関数の第一パラメータに、ClientID の代わりにタイトル文字列を指定することができます。ブローカからメッセージを受信した時に実行される MQTT_PUBLISH イベントハンドラのパラメータ “Title” に、ここで設定したタイトル文字列が入ります。 デフォルト値：””

CliendID	MQTT ブローカに接続するときの ClientID 文字列を設定します。 任意の文字列を指定できますが、必ず全てのエンドポイントでユニークになるようにします。 デフォルト値：<ホスト名と乱数によって決められた文字列>
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：""
Port	MQTT ブローカのポート番号 デフォルト値：1883
UserName	MQTT ブローカに接続するときの ユーザー名を設定します。 "Password" 設定項目と共に設定を省略("" 空文字列)した場合には、MQTT ブローカにはユーザー名・パスワードを省略して接続します。 デフォルト値：""
Password	MQTT ブローカに接続するときの パスワード文字列を設定します。 "UserName" 設定項目と共に設定を省略("" 空文字列)した場合には、MQTT ブローカにはユーザー名・パスワードを省略して接続します。 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省略("" 空文字列)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を設定します。 デフォルト値：0
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省略("" 空文字列)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：""
起動時に購読するトピック	DeviceServer 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。 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 DeviceServer 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：""
起動時に購読するトピックのQoS	"起動時に購読するトピック" 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"起動時に購読するトピック" 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。 デフォルト値：""
受信バッファ初期値	ブローカからデータを受信するときのバッファサイズを指定します。 ここで設定した値を超えたデータを受信した場合には、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確保することで、データサイズが大きくなる時のレスポンスが早くなります。 デフォルト値：2000000
詳細ログを出力	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。 デフォルト値：False

エンドポイントリストからエンドポイントを選択して、“削除”ボタンを押すとエンドポイントを削除することができます。

9.4.12 パフォーマンス設定タブ



パフォーマンス設定タブ	
同時実行可能な最大スクリプト数	イベントハンドラやユーザースクリプトが同時実行可能な最大数を指定する この値を大きくすると、Lua スクリプトエンジンのインスタンスを増やして処理パフォーマンスを向上させることができます。ただし、CPU 負荷が上限に近い場合にこの設定値を増やしても、PC全体の処理スピードはかえって低下する場合がありますので注意してください。 デフォルト値：16
内部データベースセッション数	DeviceServer が Firebird DBMS に接続するセッション数

	デフォルト値：10

9.4.13 ライセンス情報画面

先に入力されたライセンスキーによってライセンスされた情報を表示します。

ライセンス番号が空白の場合は、ライセンスキーが入力されていないか間違っています。もう一度サーバー設定プログラムを起動しなおして、正規のライセンスキーもしくはデモライセンスキーを入れてください。

タイトル	説明
ライセンス先	ライセンス先のお客様コードが入っています デモライセンスの場合はデモライセンス名称が入っています。
番号	ライセンスコードが入っています
最大同時ログイン数	DeviceServer に同時にログイン可能な最大のユーザーセッション数
サービスモジュール一覧	サーバー設定でセットアップされた機能に対応するモジュール名や、 DeviceServer でデフォルト動作している、必須のモジュール名の一覧を表示します

9.4.14 管理者アカウント設定画面

DeviceServer にユーザーアカウントが未登録の場合は管理者アカウントの登録画面を表示します。既に登録済みの場合はこの登録画面は表示されません。インストール直後にはユーザーは一つも登録されていない状態なのでこの画面から最初のユーザーを登録します。ここで登録するユーザーは管理者として登録されて、他のユーザー登録や削除等を行うことができます。

管理者ユーザー名(ログイン名)とパスワードを入力して、”登録”ボタンを押してください。ここで入力したパスワードはクライアントソフトウエア(ABDesktop)にログイン後にいつでも変更できます。

タイトル	説明
管理者ユーザー名	DeviceServer ログイン認証時に使用するアカウント名 文字列です。
パスワード	任意のパスワード文字列
パスワード(再入力)	任意のパスワード文字列

10 クライアントソフトウエア

DeviceServer のクライアントプログラムについて説明します。

10.1 クライアント起動(ABLauncher)

クライアントソフトウエアを起動するためのメインプログラムです。以降の “デスクトップ”, “アラーム管理”, “ユーザー管理” の各プログラムを利用する場合にも、クライアント起動(ABLauncher) から行います。

任意の PC からクライアントプログラムを利用したい場合は、ABLauncher プログラムのみを PC にコピーしてくだ

さい。その他のクライアントプログラムは自動的に DeviceServer からダウンロードされます。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABLauncher.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABLauncher.exe
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
ダウンロードフォルダ	C:¥Documents and Settings¥All Users¥Application Data¥AllBlueSystem Windows7(64bit)の場合は C:¥Users¥All Users¥AllBlueSystem
設定ファイル	ダウンロードフォルダ内の ABDesktop.ini 設定ファイルが存在しない場合は、起動時に接続先を入力するダイアログが表示されて、入力されたホスト名で新規の設定ファイルが作成される。

ダブルクリックで起動すると、ABLauncher は次に説明する “デスクトップ(ABDesktop)” プログラム が DeviceServer から既にクライアント PC にダウンロード済みであるかを確認します。まだダウンロードされていないか、クライアント PC にダウンロード済みのファイルバージョンが、現在 DeviceServer の持っているものと一致しない場合は自動的にダウンロードされて “デスクトップ(ABDesktop)” プログラムが起動します。

ダウンロード済みのクライアントプログラムは、表中のダウンロードフォルダに保管されています。SHIFT キーを押しながら ABLauncher を起動するとダウンロード済みのクライアントプログラムと設定ファイルを全てクライアント PC から削除することができます。

10.2 デスクトップ(ABDesktop)

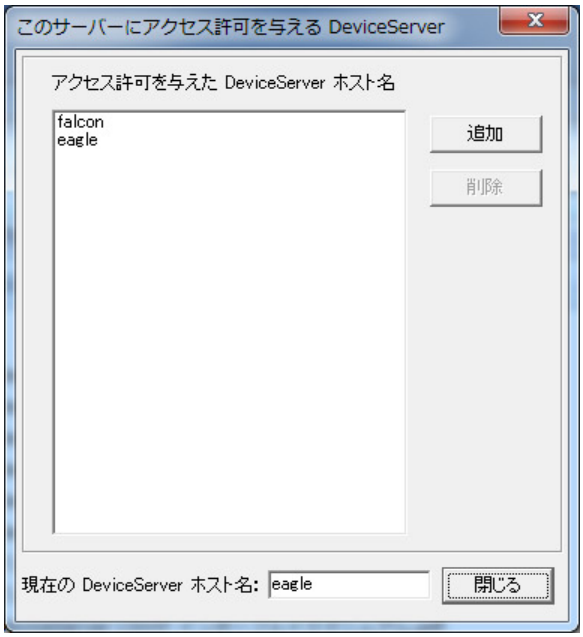
DeviceServer クライアントプログラムのメインプログラムで、ABLauncher から自動起動されます。アラーム管理やユーザー管理などのプログラムをツールボタンから起動することができます。また、ログインしたユーザーのパスワード変更等もこのプログラムから行います。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ABDesktopr.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ABDesktopr.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
ダウンロードフォルダ	C:¥Documents and Settings¥All Users¥Application Data¥AllBlueSystem
設定ファイル	ダウンロードフォルダ内の ABDesktop.ini

ABDesktop プログラムを起動すると最初にユーザー認証ダイアログを表示します。ここでホスト名項目に DeviceServer の動作している PC のホスト名を入力します。ユーザー名とパスワードは DeviceServer に登録済みのアカウントのログイン名とパスワードを入力します。

認証が成功するとデスクトップ画面を表示します。各ツールボタンの機能を以下に説明します。ログインしたアカウントのアプリケーション許可フラグが設定されていない場合には、対象機能のツールボタンを表示しません。

ツールボタン	説明
終了 	ログアウトしてから、ABDeskTop プログラムを終了します。アラーム管理(AlarmConfig)または、ユーザー管理(UserMgr) プログラムを起動していた場合は同時に終了させます。
ユーザー 	ユーザー管理(UserMgr) プログラムを起動します。既に起動中の場合には、最前面にユーザー管理(UserMgr) プログラムを表示します。 このボタンを有効にするためには、ログインするユーザーアカウントに、[UserMgr, AdminTask] 両方のアプリケーション許可フラグが必要です。
アラーム 	アラーム管理(AlarmConfig) プログラムを起動します。既に起動中の場合には、最前面にアラーム管理(AlarmConfig) プログラムを表示します。 このボタンを有効にするためには、ログインするユーザーアカウントに、[AlarmConfig, AdminTask] 両方のアプリケーション許可フラグが必要です。
XBee 	XBeeデバイス管理(XBeeConfig) プログラムを起動します。既に起動中の場合には、最前面にXBeeデバイス管理(XBeeConfig) プログラムを表示します。 このボタンを有効にするためには、ログインするユーザーアカウントに、[XBeeConfig, AdminTask] 両方のアプリケーション許可フラグが必要です。
ZB 	XBee-ZBデバイス管理(ZBConfig) プログラムを起動します。既に起動中の場合には、最前面にXBee-ZBデバイス管理(ZBConfig) プログラムを表示します。 このボタンを有効にするためには、ログインするユーザーアカウントに、[ZBConfig, AdminTask] 両方のアプリケーション許可フラグが必要です。
パスワード 	ログインしているユーザーの現在のパスワードを変更します。
許可 	リモート側 DeviceServer にアクセスするリモートアクセス・ライブラリ関数(script_rexec(), script_rexec2(),get_net_data(),....等) を使用するときに、この PC にアクセス可能なホスト名リストを設定します。

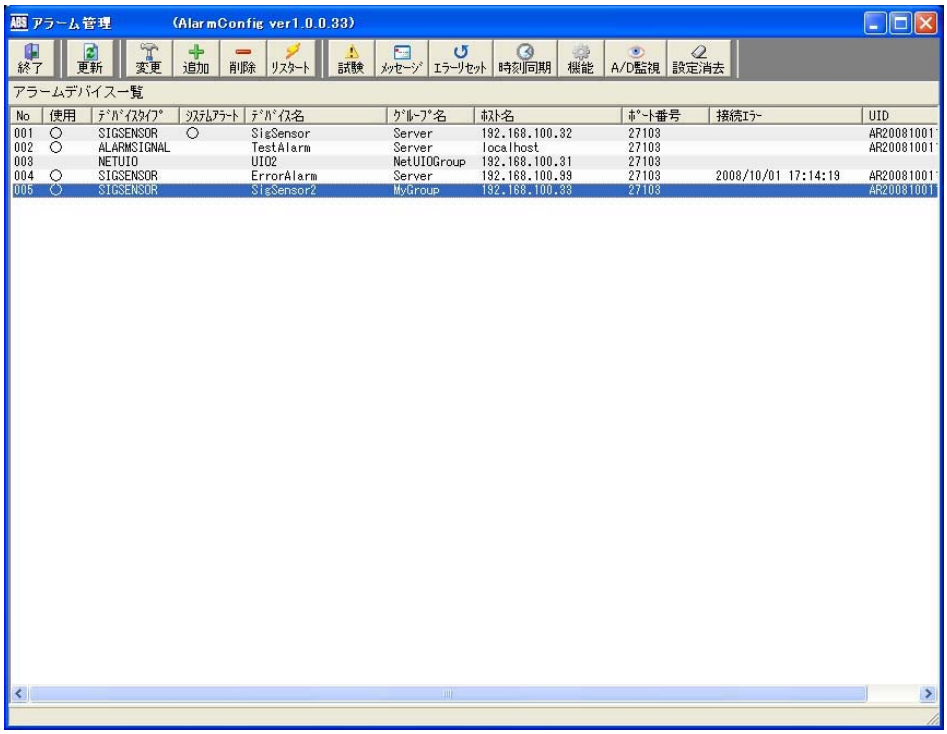
	 別 PC で動作している DeviceServer から、この PC に対して実行されたリモートアクセス・ライブラリ関数は、ここでアクセス許可が与えられているかを常にチェックしています。 ここで指定するホスト名は、同様にリモート側で起動した "デスクトッププログラム" の"許可" ボタンを押して、一番下の "現在の DeviceServerホスト名"に表示されている名前を使用します。IP アドレスを許可リストに登録することはできませんので注意してください。
情報  	DeviceServer のバージョンやライセンス情報などの表示と、ログインしているユーザーアカウントの情報を表示します。

10.3 アラーム管理(AlarmConfig)

SigSensor, NetUIO, アラームシグナルデバイスを DeviceServer で管理するためのクライアントプログラムです。DeviceServer ではSigSensor, NetUIO, アラームシグナルなどのデバイス管理用のデータベースを保管しています。また SigSensor, NetUIO デバイスにも自分のIPアドレスやイベント送信先の DeviceServerの IP アドレス情報等を、CPU ボード上の24C256(EEPROM) チップに保管しています。アラーム管理プログラムではこれら両方の設定情報を変更することができます。デバイス追加の詳しい手順については、"デバイス管理" の章も参照してください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥AlarmConfig.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥AlarmConfig.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

起動すると、登録済みアラームデバイス一覧とツールボタンを表示します。



デバイス一覧に表示される項目を以下に説明します。また、一覧の項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	デバイス登録番号で、登録順に001 から順に自動でアサインします。
使用	このデバイスを DeviceServer から使用するかどうかを示します。○が付いていると使用可能な状態です。
デバイスタイプ	デバイスタイプ文字列です。(ALARMSIGNAL, SIGSENSOR, NETUIO のいずれか)
デバイス名	デバイス名を表します。スクリプトや API からはデバイス名で対象デバイスを指定します。
グループ名	デバイスに付けられたグループ名です。
ホスト名	デバイスの IP アドレスまたはホスト名です。
ポート番号	デバイスのポート番号です。
接続エラー	デバイスとの通信中にエラーが発生場合にエラー発生時刻が入ります。エラー発生時刻が入っているデバイスとは接続できません。”エラーリセット” ツールボタンを押すとこの情報をクリアできます。
UID	DeviceServer 起動時にデバイス付けられた UID です。

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
終了 	AlarmConfig プログラムを終了します。ログアウトは行わずに、AlarmConfig のメインフォームのみが閉じられます。ログアウトを行う場合は、デスクトップ(ABDesktop)の終了ボタンを押してください。

更新 	アラームデバイス一覧を最新の情報に更新します。
変更 	アラームデバイスを１つ選択して、デバイスの名前や IP アドレス等の設定内容を変更します。
追加 	アラームデバイスを追加します。
削除 	アラームデバイスを選択して、DeviceServer から登録削除します。
リスタート 	DeviceServer のアラーム管理モジュールを再起動します。通常はアラームデバイスの設定内容を変更すると、自動的にモジュールが再起動されるため直接操作することはありません。
試験 	アラームデバイス(ALARMSIGNAL, SIGSENSORデバイスのみ) を選択して、ランプとブザーの試験出力を行います。
メッセージ 	アラームデバイス(SIGSENSORデバイスのみ) を選択して、LCD 表示器にメッセージ送信試験を行います。
エラーリセット 	デバイスとの通信でネットワークエラーを検出したために、DeviceServer から一時的に切り離されていた状態をリセットします。リセットを行っても、ネットワークエラーとなった原因が解決していないと、再度接続を試みた時にエラーを検出します。
時刻同期 	アラームデバイス(SIGSENSORデバイスのみ)の時計を、DeviceServer の動作しているPC のローカル時刻に合わせます。
機能 	アラームデバイス(ALARMSIGNAL, SIGSENSORデバイスのみ) を選択して、デバイスの詳細設定項目を変更します。詳細設定項目の内容については後述します。
A/D 監視 	アラームデバイス(ALARMSIGNAL, SIGSENSORデバイスのみ) を選択して、デバイスの A/D 変換レンジチェック機能を有効または無効に設定します。アラームデバイスのリセット直後は、レンジチェックは無効になるため、このボタンを使用して該当 A/D channel のレンジチェックを有効にします。同様の操作は、スクリプトのライブラリ関数から実行することができます。また、一度レンジチェックの制限値のイベントを送信した後は、それ以降のチェックは自動的に無効になりますので、再び検知するためには同様にこのボタンまたはスクリプトから操作する必要があります。
設定消去 	アラームデバイス(ALARMSIGNAL, SIGSENSORデバイスのみ) を選択して、デバイスの詳細設定内容を初期化します。

10.3.1 アラームデバイス追加・変更

デバイス追加・変更時に表示される設定画面の項目説明は以下になります。この設定内容は、DeviceServer で管理されています。

DeviceServer のスクリプトや API からアラームデバイスをアクセスするときに、ここで設定したIPアドレスのデバイスに接続します。ここでの設定項目の IP アドレスやポート番号を変更しても、アラームデバイス自身の設定内容(EEPROM内容)は変更しませんので注意してください。デバイス自身の詳細設定内容(EEPROM) を変更するには、次の”

機能” ツールボタンを使用します。

タイトル	説明
このデバイスを有効にする	デバイスを使用する場合にチェックを付けてください。一時的に DeviceServerのネットワークから切断する場合には、チェックをはずしてください。
デバイスタイプ	ALARMSIGNAL, SIGSENSOR, NETUIO のいずれかを指定します。
名前	デバイスにつける名前を指定します。DeviceServer のスクリプトやAPI からは、ここで指定したデバイス名でデバイスを指定します。
グループ名	デバイスにグループ名をつけて、デバイス一覧をソートした時に見やすく表示できます。
システムアラートとして使用	チェックを付けると、スクリプトライブラリの alarm_sysalert_list() 関数で取得するデバイス名リストに入ります。ネットワークエラーや警報を出力したいデバイスに別途マークを付けておきたい場合に使用することができます。
ホスト名(もしくはIPアドレス)	デバイスの IP アドレスもしくはホスト名を指定します。**SigSensor, NetUIO デバイスの場合は必ず IP アドレスで入力してください。この場合は、ここで入力する値は、後に説明するデバイス詳細設定項目でデバイス自身に設定する IP アドレスと一致させる必要があります。**DeviceServer はイベントを受信した時に、ここに入力されたIP アドレス値を元に、送信元デバイスを特定します。
ポート番号	デバイスの ポート番号を指定します。

10.3.2 アラームデバイスの詳細機能設定

“機能” ツールボタン(前述)を押して、デバイスの詳細機能設定(EEPROM)フォームで以下の項目を設定できます。詳細機能設定フォームが有効なデバイスは、“SIGSENSOR”, “NETUIO” です。内容を変更した後は、必ずデバイスのリセットが必要になります。リセット後に、新しい設定内容でデバイスが動作します。

デバイス設定変更についての詳しい手順は、”デバイス管理” の章を参照してください。

タイトル	説明
IPアドレス	デバイスの IP アドレスを指定します。
ポート番号	デバイスのポート番号を指定します。
IPネットワークマスク	IP ネットワークマスクを指定します。デバイスを接続するネットワーク環境に合わせてください。
デフォルトゲートウェイアドレス	デフォルトゲートウェイアドレスを指定します。デバイスを接続するネットワーク環境に合わせてください。
イベント送信先ホストアドレス	デバイスでイベントが発生したときに、ここで指定したIPアドレスにイベントパケット(TCP/IP ストリームソケットを使用) を送信します。DeviceServer の IP アドレスを指定してください。
送信先ポート番号	イベント送信に使用するポート番号。デフォルトの27102 を指定してください。
ボタンを押したときにサーバーに CSVIFパケット送信	チェックを付けると、デバイスのボタンを押したときに DeviceServer にイベントを送信します。イベント名は ALARM_BUTTON_PUSH

DINPUT を有効にする	チェックを付けると、デジタル入力(DINPUT) ポートを有効にします。
DINPUT 値が変化した時にサーバーに CSVIFパケット送信	チェックを付けると、デジタル入力(DINPUT) ポートの値が変化したときに、DeviceServer にイベントを送信します。イベント名は ALARM_DINPUT_CHANGE
デバイス設定のLCD表示を有効にする	チェックを付けると、下記のボタンの組み合わせで LCD に情報を表示します。 A_BTN & B_BTN　ファームウエアバージョン表示 B_BTN & C_BTN　デバイスのネットワーク設定等を順番に表示 (表示内容と詳しい説明は SigSensor_NetUIOユーザーマニュアルを参照の事)
A/D入力を有効にする	チェックを付けると、A/D 変換を有効にします。
A/D サンプリングレート	A/D 変換をここで指定した間隔で自動的に行います。 変換データはデバイス内部のバッファに格納され最大 channel毎に 1024 エントリ分保存します。バッファに格納されたデータは何時でもスクリプトライブラリもしくは API を使用してクライアントから取り出せます。
A/D レンジチェックを有効にする	チェックを付けると、A/D 変換値が制限値をこえたときに、 DeviceServer にイベントを送信します。イベント名は ALARM_ADRANGE_EXCEED レンジチェックイベントを有効にするためには、この設定に加えて “A/D 監視”ツールボタンもしくはスクリプトライブラリから各channelのレンジチェックを有効にする必要があります。
AD1 LOW	A/D1 のレンジチェック下限値
AD2 LOW	A/D2 のレンジチェック下限値
AD3 LOW	A/D3 のレンジチェック下限値
AD4 LOW	A/D4 のレンジチェック下限値
AD1 HIGH	A/D1 のレンジチェック上限値
AD1 HIGH	A/D2 のレンジチェック上限値
AD1 HIGH	A/D3 のレンジチェック上限値
AD1 HIGH	A/D4 のレンジチェック上限値

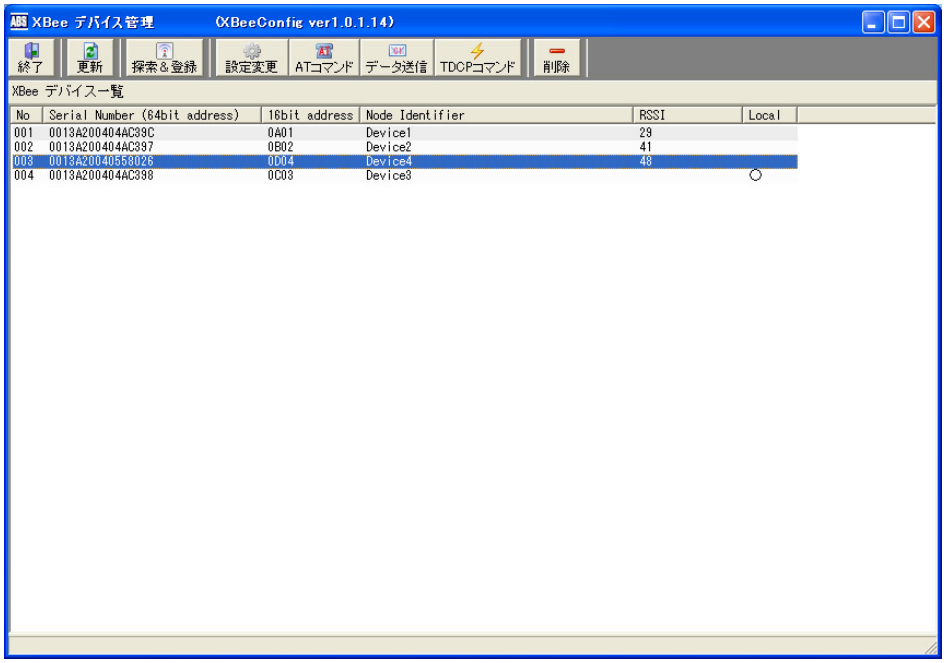
10.4　XBee デバイス管理(XBeeConfig)

XBee デバイスを DeviceServer で管理するためのクライアントプログラムです。DeviceServer に COM Port経由で接続した XBee デバイスに加えて、同一 PAN(Personal Area Network) 内の複数の XBee デバイスの管理を行うことができます。リモートにある XBee デバイスの設定内容もこのプログラムから変更することができます。

デバイス追加の詳しい手順については、”デバイス管理” の章も参照してください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥XBeeConfig.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥XBeeConfig.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

起動すると、登録済みXBee デバイス一覧とツールボタンを表示します。



デバイス一覧に表示される項目を以下に説明します。また、一覧の項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	XBeeデバイス番号で、マスタファイル登録順に001 から順に自動でアサインします
Serial Number	XBee デバイスに設定済みのシリアル番号 64 bit のデバイスアドレスとしても使用します。
16 bit address	ユーザーが XBee デバイスに設定した任意の 16 bit アドレス ユニークな番号を割り振る必要があります。
Node Identifier	ユーザーが XBee デバイスに設定した任意のデバイス名 ユニークなASCII 文字列をアサインする必要があります。DeviceServer のスクリプト中から、Serial Number, 16 bit address と同様に、デバイスを特定する名前として使用できます。
RSSI	“探索＆登録”ボタンでデバイスを登録したときに、取得した RSSI データ
Local	DeviceServer のCOM ポートに直接接続された XBee デバイスの場合には、’○’ を表示します

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
終了 	XBeeConfig プログラムを終了します。ログアウトは行わずに、XBeeConfig のメインフォームのみが閉じられます。ログアウトを行う場合は、デスクトップ(ABDesktop)の終了ボタンを押してください
更新 	XBee デバイス一覧を最新の情報に更新します。 XBee デバイスが登録されているマスターファイルを再読み込みして、デバイス一覧を更新します

探索＆登録 	DeviceServer に接続された XBee デバイスから “Node Discover” コマンドを実行して、同一 PAN ID 内のデバイスを検索します。見つかった XBee デバイスは自動的にマスターファイルに登録します
設定変更 	XBee デバイスを選択して、詳細設定項目を変更します。詳細設定項目の内容については後述します
ATコマンド 	XBee デバイスを選択して、任意の AT コマンドを送信します。AT コマンド送信機能については後述します
データ送信 	XBee デバイスを選択して、任意のバイナリデータまたは ASCII 文字列を送信します。データ XBee デバイスを選択して、任意のデータを送信します。 パケット送信機能については後述します
TDCPコマンド送信 	TDCPプログラムが動作している CPU ボードに接続されたXBee デバイスを選択して、TDCPコマンドを送信します。TDCP コマンド送信機能については後述します。
削除 	XBeeデバイスを選択して、DeviceServer のマスターファイルから削除します

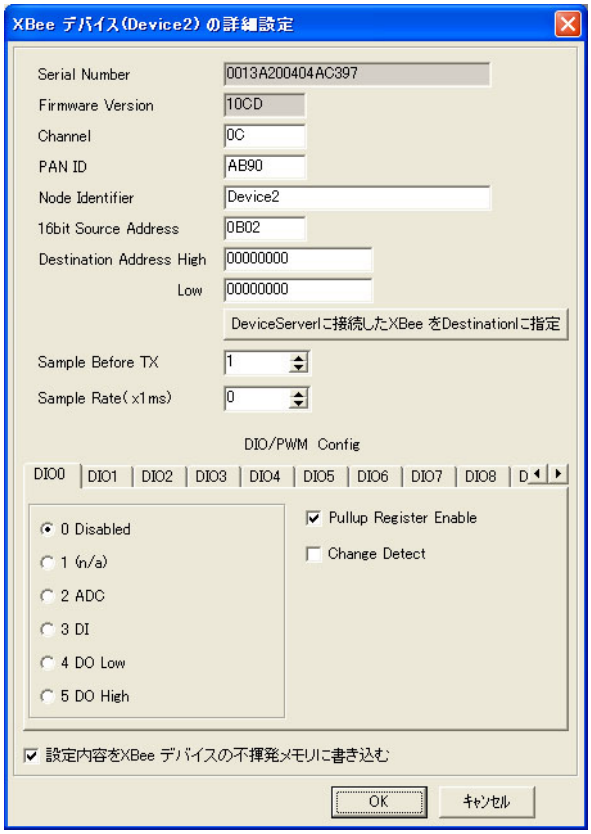
10.4.1 XBee デバイス詳細設定

XBee デバイス管理プログラムの “設定変更” ツールボタン(前述)を押して、デバイスの詳細機能設定フォームで以下の項目を確認・設定変更できます。

設定項目の中には読み出ししかできないものがありますが、このようにユーザーが変更できない項目についてはボーダーカラーで表示されて、GUI を操作しても変更できないようにしています。

16進数表記で表示・変更する項目と、Ascii 文字列で表示・変更する設定項目については、テキスト入力エリアコンポーネント を使用します。16進数で入力するときには '0'-'9' と 'A'-'F' の文字を使用してください。また、16進数のプリフィックス文字列 "0x" 部分は除いて入力してください。10 進数表記で表示・変更する項目は スピンエディットコンポーネント(上下の矢印がついた、整数入力のみができるコンポーネント)を使用します。

表中の “対応 AT コマンド” の詳しい内容は、Digi社の “XBee 802.15.4 OEM RF Modules” ドキュメント中の AT コマンドの説明を参照してください。



タイトル	説明
Serial Number	デバイスシリアル番号　対応 AT コマンド:SH, SL
Firmware Version	デバイスファームウエアバージョン　対応 AT コマンド:VR
Channel	RF Channel を設定します　対応 AT コマンド:CH
PAN ID	PAN ID を設定します　対応 AT コマンド:ID
Node Identifier	Node Identifier を設定します　対応 AT コマンド:NI
16bit Source Address	16bit Source Address を設定します　対応 AT コマンド:MY
Destination Address High/Low	Destination Address High/Low を設定します　対応 AT コマンド:DH, DL
DeviceServer に接続したXBee を Destination に指定	このボタンを押すと、DeviceServer 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 Destination Address High/Low に設定します。 デバイス上で発生した Change Detect の I/O データフレームをDeviceServer のイベント(XBEE_IO_DATA)として受信するためには、この設定を行ってデータフレームの送信先を設定してください　対応 AT コマンド:DH, DL
Sample Before TX	Sample Before TX を設定します　対応 AT コマンド:IT
Sample Rate	Sample Rateを設定します　対応 AT コマンド:IR
DIO/PWM Config (タブ選択)	各 DIO ポートと PWM の Configuration を選択します 対応 AT コマンド:D0, D1, D2, D3, D4, D5, D6, D7, D8, P0, P1
DIO(xx) Pullup Register Enable	DIO の各ポートの Pullup Resister Enable を設定します 対応 AT コマンド:PR

DIO(xx) Change Detect	DIO の各ポートの Change Detect を設定します 対応 AT コマンド:IC
設定内容をXBee デバイスの不揮発メモリに書き込む	現在の設定内容を XBee の不揮発メモリに書き込みます 対応 AT コマンド:WR

10.4.2 AT コマンド送信機能

XBee デバイスに任意の AT コマンドを送信してレスポンスフレームを受信する機能です。

詳細設定画面では直接設定できないXBee の機能を設定する場合や、テスト時などに使用します。

AT コマンドボタンを押すと送信フォームが表示され、任意の ATコマンドとパラメータを送信することができます。XBee デバイスのAT コマンドはパラメータを省略すると現在の設定値をレスポンスフレームに返します。パラメータを指定すると、そのパラメータ値に設定内容が更新されて実行結果ステータスをレスポンスフレームに返します。

選択した XBee デバイスによって、自動的に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に切り替えて送信します。AT コマンドフレーム中の FrameID は常に 0x01 になります。またリモートコマンド中のコマンドオプションは 0x02(Apply Change on remote) になります。

送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier	(変更不可) コマンド送信対象となる、XBee デバイスの Node Identifier
AT コマンド	AT コマンド文字列、2 文字のASCII 大文字を指定する
コマンドパラメータ	コマンドパラメータを 16進数形式で入力する。例: 0102FF コマンドパラメータを送信しない場合は、入力フィールドを空にすること
レスポンスフレーム	AT コマンド送信後に、XBee デバイスからのレスポンスフレームが入ります

ATコマンドとコマンドパラメータを入力して、”送信” ボタンを押すと AT コマンドが実行され、レスポンスフレームを表示します。

ATコマンドで XBee 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コマンド “WR” も送信してください。もしくは、AT コマンドを送信した後に、XBee デバイス詳細設定フォーム（前述）を開いて “設定内容をXBee デバイスの不揮発メモリに書き込む” にチェックをつけて、書き込むこともできます。

終了する場合は、”閉じる” を押してください。

10.4.3 データパケット送信機能

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機能です。
XBee 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。

データ送信コマンドボタンを押すと送信フォームが表示され、任意のバイナリデータまたはASCII 文字列を送信することができます。バイナリデータを送信する場合は、送信データフィールドに16進数表記の文字列で入力して下さい。例えば 0x0A, 0x0B, 0x0C の３バイトを送信する場合は 0A0B0C を入力します。

データパケットは、XBee API の Transmit Request(0x00)コマンドを使用して送信します。 FrameID は常に 0x01 を使用します。

送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier	（変更不可）コマンド送信対象となる、XBee デバイスの Node Identifier
データフォーマット指定	バイナリデータを送信する場合は “HEX文字列” を選択する ASCII 文字列を送信する場合は “ASCII文字列” を選択する
送信データ	送信データを 16進数形式または、ASCII で入力する。 送信データを空にするとエラーになりますので注意してください。

”送信” ボタンを押すとデータパケットを作成して送信します。連続して送信を行う場合は、”送信”ボタンを連続して押してください。

終了する場合は、”閉じる” を押してください。

10.4.4 TDCPコマンド送信機能

TDCPプログラムが動作しているCPU ボードに接続されたXBee デバイスに、TDCPコマンドを送信してリプライ文字列を受信する機能です。TDCPプログラムとCPUボードの詳細については “TDCPユーザーマニュアル”を参照してください。

送信コマンドボタンを押すとTDCPコマンド欄に入力された文字列を送信します。リモートデバイスからリプライパケットを受信するとTDCP リプライ欄にその内容を表示します。

”リプライを受信しない” のチェックボックスをチェックすると、リプライパケットの受信を行いません。TDCP コマンドの “reset” コマンドなど、リプライパケットを送信しないコマンドを実行する場合にチェックを付けます。”リプライを受信しない”のチェックを外しておくことで、リモートからのリプライパケットが受信できない場合には、自動的にコマンド送信のリトライが行われます。

データパケットは、XBee API の Transmit Request(0x00)コマンドを使用して送信します。 FrameID は常に 0x01 を使用します。

送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier(list)	コマンド送信対象となる、XBee デバイスの Node Identifierを指定する。 複数のデバイスをカンマで並べて表記した場合には、左に指定したデバイスから順番にコマンドがバケツリレー方式で転送されて、一番右側に指定したデバイスでコマンドが実行される(ルータ経由のコマンド実行)。ルータ機能の詳細はスクリプトライブラリ関数の “xbee_tdcp_far_command()”の説明を参照してください。

TDCPコマンド	TDCPコマンド文字列を入力する
“>>”,”<<”	以前に実行したコマンド文字列をコマンド入力欄に表示する
リプライを受信しない	リモートデバイスからの TDCP リプライパケットを受信しない場合にチェックする。
TDCPリプライ	リモートデバイスで実行したTDCPコマンドのリプライ文字列が設定される

”送信” ボタンを押すとTDCPコマンドを送信します。連続して送信を行う場合は、”送信”ボタンを連続して押してください。

終了する場合は、”閉じる” を押してください。

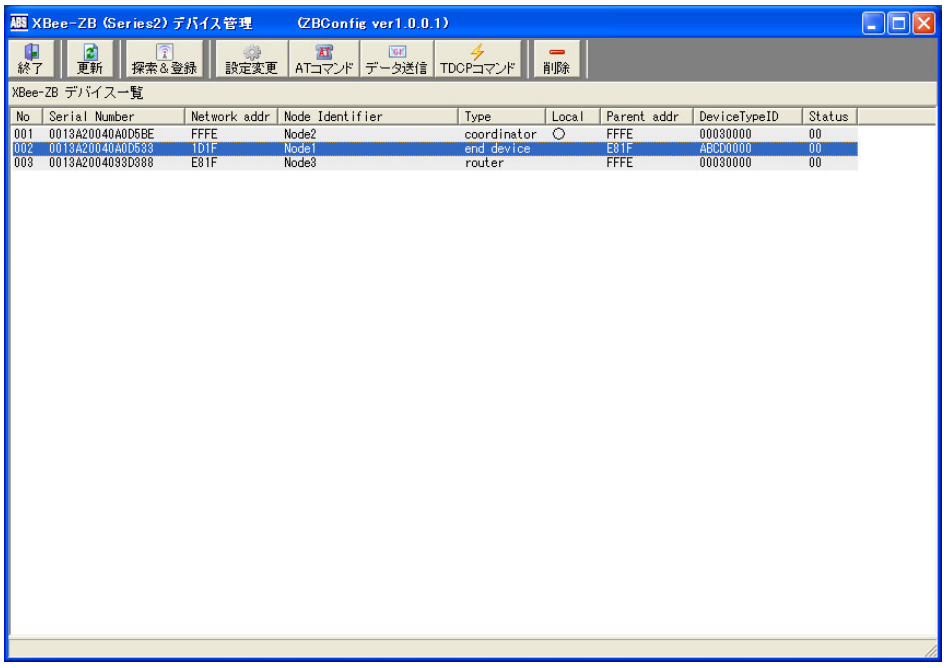
10.5 XBee-ZB デバイス管理(ZBConfig)

XBee-ZB デバイスを DeviceServer で管理するためのクライアントプログラムです。前項の XBee デバイス管理プログラムは XBee 802.15.4 Series1 デバイスが対象でしたが、XBee-ZB デバイス管理プログラムは XBee-ZB Series2 デバイスの管理を行います。DeviceServer に COM Port経由で接続した XBee-ZB デバイスに加えて、同一 PAN(Personal Area Network) 内の複数の XBee-ZB デバイスの管理を行うことができます。リモートにある XBee-ZB デバイスの設定内容もこのプログラムから変更することができます。

デバイス追加の詳しい手順については、”デバイス管理” の章も参照してください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ZBConfig.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ZBConfig.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

起動すると、登録済みXBee デバイス一覧とツールボタンを表示します。



デバイス一覧に表示される項目を以下に説明します。また、一覧の項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	XBee-ZB デバイス番号で、マスタファイル登録順に001 から順に自動でアサインされます
Serial Number	XBee デバイスに設定済みのシリアル番号 64 bit のデバイスアドレスとしても使用されます。
Network address	XBee-ZB デバイスが ZigBee ネットワークにに参加したときに自動的にアサインされる 16 bit 幅のネットワークアドレスです。 デバイスが接続しているネットワークのペアレントが変わった場合などにこのアドレス値は変化する場合があります。最新のNetwork address を確認する場合にはツールボタンの "更新" を押してください。DeviceServer 内部でキャッシュしている最新の Network address で表示内容を更新します。
Node Identifier	ユーザーが XBee デバイスに設定した任意のデバイス名 ユニークなASCII 文字列をアサインする必要があります。DeviceServer のスクリプト中から、Serial Number, Network address と同様に、デバイスを特定する名前として使用できます。
Type	XBee-ZB のファームエアで設定された ZigBee デバイスタイプを表す文字列です。"coordinator", "router", "end device" の何れかになります。
Local	DeviceServer のCOM ポートに直接接続された XBee デバイスの場合には、'○' を表示します
Parent addr	XBee-ZB デバイスがネットワークに参加したときのペアレントデバイスのネットワークアドレスを示します。この値は "探索・登録" を行った時点の情報をマスターから読み出して表示しています。このため Network address 項目とは違って実

	際の現在の値を示しているとは限りませんので注意してください。現在の値を参照するときは、"設定変更" ボタンを押して XBee-ZB デバイスからパラメータ値を取得してください。
DeviceTypeID	XBee-ZB デバイスに設定された任意の DeviceTypeID を示します。 Parent addr 項目と同様に最後に "探索・登録" を行った時点の情報をマスターから読み出して表示しています。現在の値を参照するときは、"設定変更" ボタンを押して XBee-ZB デバイスからパラメータ値を取得してください。
Status	Discover コマンド実行時に返された XBee-ZB デバイスのステータス値。Parent addr 項目と同様に最後に "探索・登録" を行った時点の情報をマスターから読み出して表示しています。

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
終了 	ZBConfig プログラムを終了します。ログアウトは行わずに、ZBConfig のメインフォームのみが閉じられます。ログアウトを行う場合は、デスクトップ(ABDesktop)の終了ボタンを押してください
更新 	XBee-ZB デバイス一覧を最新の情報に更新します。 XBee デバイスが登録されている、マスターファイルを再読み込みして、デバイス一覧を更新します。このとき、DeviceServer 内部でキャッシュしている最新の Network address データで表示内容を更新します。
探索＆登録 	DeviceServer に接続された XBee-ZB デバイスから “Node Discover” コマンドを実行して、同一PAN ID 内のデバイスを検索します。見つかった XBee-ZB デバイスは自動的にマスターファイルに登録します
設定変更 	XBee-ZB デバイスを選択して、詳細設定項目を変更します。詳細設定項目の内容については後述します
ATコマンド 	XBee-ZB デバイスを選択して、任意の AT コマンドを送信します。AT コマンド送信機能については後述します
データ送信 	XBee-ZB デバイスを選択して、任意のバイナリデータまたは ASCII 文字列を送信します。データXBee-ZB デバイスを選択して、任意のデータを送信します。 パケット送信機能については後述します
TDCPコマンド送信 	TDCPプログラムが動作している CPU ボードに接続されたXBee-ZB デバイスを選択して、TDCPコマンドを送信します。 TDCP コマンド送信機能については後述します。
削除 	XBee-ZB デバイスを選択して、DeviceServer のマスターファイルから削除します

10.5.1 XBee デバイス詳細設定

XBee-ZB デバイス管理プログラムの “設定変更” ツールボタン(前述)を押して、デバイスの詳細設定フォームで以下の項目を確認・設定変更できます。

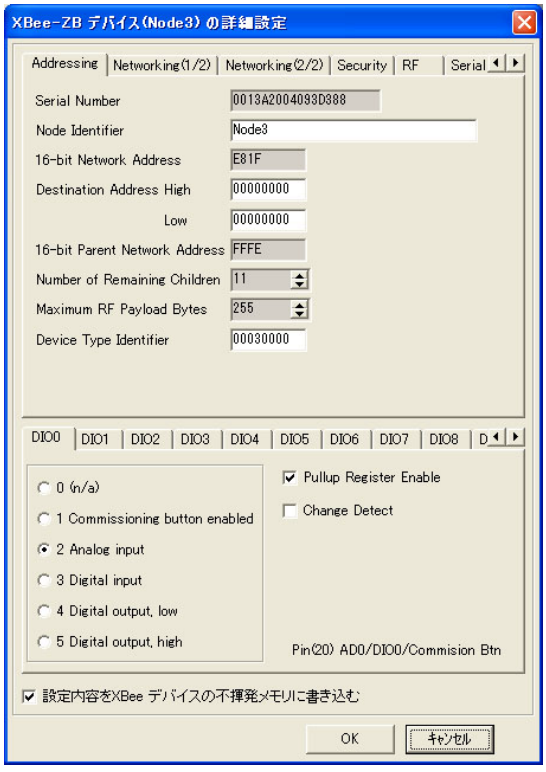
XBee-ZB のパラメータは多くの種類がありますので、詳細設定フォームは機能毎にタブで分けられています。タブの名称は Digi International 社の "XBee ZB Modules" マニュアルと同じ区分になっています。XBee-ZB デバイスの ZigBee デバイスタイプ("coordinator", "router", "end device") によっては、サポートしていない設定項目がありますが、この場合には設定項目は非表示になります。 Execution 関連のコマンドは次の項で説明している "AT コマンド" ツールボタンを使用して直接 AT コマンドを実行してください。

設定項目の中には読み出ししかできないものがありますが、このようにユーザーが変更できない項目についてはボーダーカラーで表示されて、GUI を操作しても変更できないようにしています。

16進数表記で表示・変更する項目と、Ascii 文字列で表示・変更する設定項目については、テキスト入力エリアコンポーネント を使用します。16進数で入力するときには '0'-'9' と 'A'-'F' の文字を使用してください。また、16進数のプリフィックス文字列 "0x" 部分は除いて入力してください。10 進数表記で表示・変更する項目は スピンエディットコンポーネント(上下の矢印がついた、整数入力のみができるコンポーネント)を使用します。

表中の "対応 AT コマンド" の詳しい内容は、Digi社の "XBee ZB RF Modules" ドキュメント中の AT コマンドの説明を参照してください。タブ毎の設定項目は以下のようになっています。

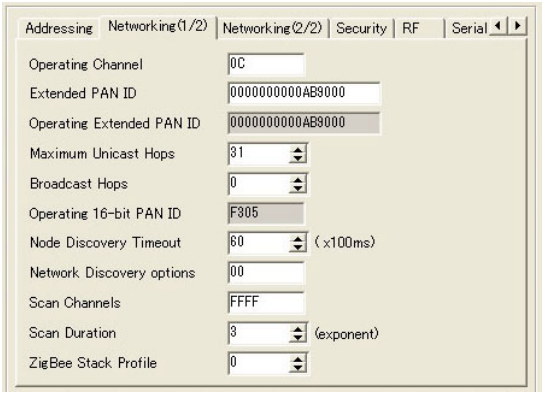
Addressing タブ, DIO タブ, 不揮発メモリ書き込み

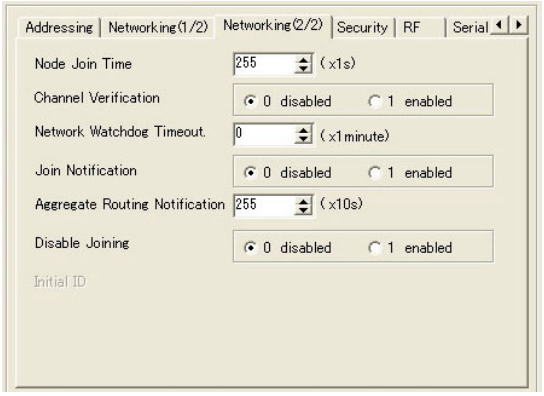


タイトル	説明
Serial Number	デバイスシリアル番号　対応 AT コマンド:SH, SL
Node Identifier	Node Identifier を設定します 対応 AT コマンド:NI

16-bit Nerwork Adress	Network Address 対応 AT コマンド:MY
Destination Address High/Low	Destination Address High/Low を設定します 対応 AT コマンド:DH, DL XBee-ZB デバイスから送信される I/O イベントメッセージを DeviceServerに送信する場合には、ZigBee Coordinator を示す 0 を設定してください。
16-bit Parent Network Address	16-bit network address 対応 AT コマンド:MP
Number of Remaining Children	Number of Remaining Children 対応 AT コマンド:NC
Maximum RF Payload Bytes	Maximum RF Payload Bytes 対応 AT コマンド:NP
Device Type Identifier	Device Type Identifier 対応 AT コマンド:DD
DIOxx (タブ選択)	XBee-ZB デバイスの各ポートについて、digital input, digital output, ADC, その他機能を選択します 対応 AT コマンド:D0, D1, D2, D3, D4, D5, D6, D7, D8, P0, P1, P2
DIO(xx) Pullup Register Enable	DIO の各ポートの Pullup Resister Enable を設定します AT コマンド:PR
DIO(xx) Change Detect	DIO の各ポートの Change Detect を設定します 対応 AT コマンド:IC
設定内容をXBee デバイスの不揮発メモリに書き込む	現在の設定内容を XBee の不揮発メモリに書き込みます 対応 AT コマンド:WR

Networking(1/2, 2/2) タブ

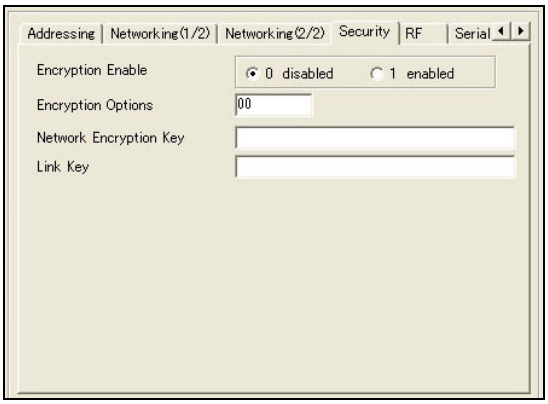




タイトル	説明
Operating Channel	Operating Channel 対応 AT コマンド:CH
Extended PAN ID	Extended PAN ID 対応 AT コマンド:ID

	デバイスの初期設定時に X-CTU から設定した値です。この値を変更することもできますが、同一の PAND ID を持ったデバイス間でしかネットワークに参加出来ないので注意してください。
Operating Extended PAN ID	Operating Extended PAN ID　対応 AT コマンド:OP
Maximum Unicast Hops	Maximum Unicast Hops　対応 AT コマンド:NH
16-bit Parent Network Address	16-bit network address 対応 AT コマンド:MP
Number of Remaining Children	Number of Remaining Children 対応 AT コマンド:NC
Maximum RF Payload Bytes	Maximum RF Payload Bytes 対応 AT コマンド:NP
Broadcast Hops	Broadcast Hops 対応 AT コマンド:BH 0 を設定すると最大数として解釈します
Operating 16-bit PAN ID	Operating 16-bit PAN ID　対応 AT コマンド:OI
Node Discovery Timeout	Node Discovery Timeout　対応 AT コマンド:NT 32 から 255 までの整数を設定します
Network Discovery options	Network Discovery options　対応 AT コマンド:NO
Scan Channels	Scan Channels　対応 AT コマンド:SC
Scan Duration	Scan Duration　対応 AT コマンド:SD
ZigBee Stack Profile	ZigBee Stack Profile 対応 AT コマンド:ZS
Node Join Time	Node Join Time 対応　AT コマンド:NJ
Channel Verification	Channel Verification　対応 AT コマンド:JV
Network Watchdog Timeout	Network Watchdog Timeout　対応 AT コマンド:NW
Join Notification	Join Notification　対応 AT コマンド:JN
Aggregate Routing Notification	Aggregate Routing Notification 対応 AT コマンド:AR
Disable Joining	Disable Joining　対応 AT コマンド:DJ
Initial ID	Initial ID　対応 AT コマンド:II

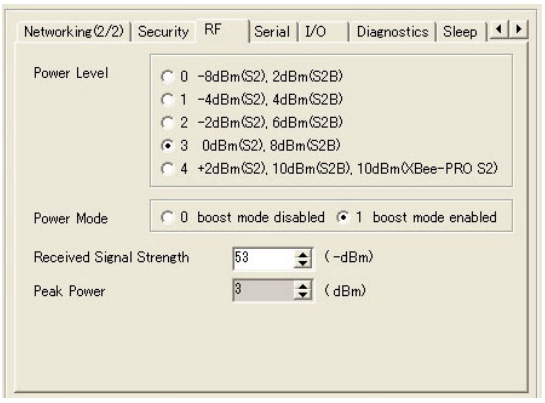
Security タブ



タイトル	説明
Encryption Enable	Encryption Enable　対応 AT コマンド:EE

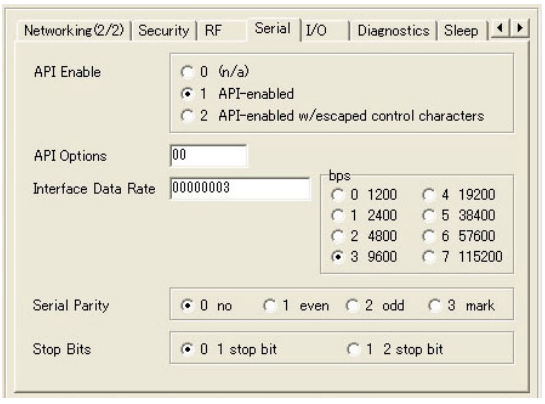
Encryption Options	Encryption Options　対応 AT コマンド:EO
Network Encryption Key	Network Encryption Key　対応 AT コマンド:NK 128bit長のネットワークキーを設定できます。この設定項目は書き込みのみできます。
Link Key	Link Key　対応 AT コマンド:KY 128bit長のリンクキーを設定できます。この設定項目は書き込みのみできます

RF タブ



タイトル	説明
Power Level	Power Level　　対応 AT コマンド:PL
Power Mode	Power Mode　対応 AT コマンド:PM
Received Signal Strength	Received Signal Strength　対応 AT コマンド:DB 0 を設定すると最後に受信した信号レベルをクリアします。
Peak Power	Peak Power　対応 AT コマンド:PP

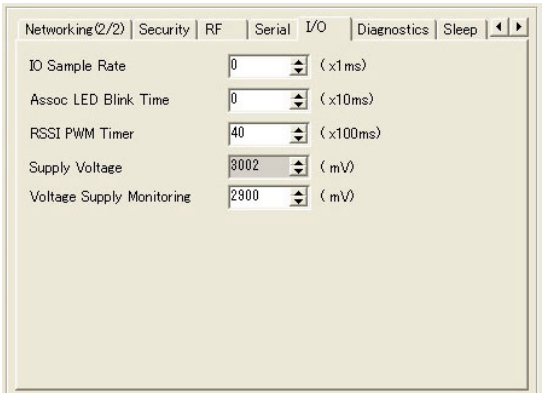
Serial タブ



タイトル	説明
API Enable	API Enable　対応 AT コマンド:AP DeviceServer で使用する場合には常に 1 を設定してください。

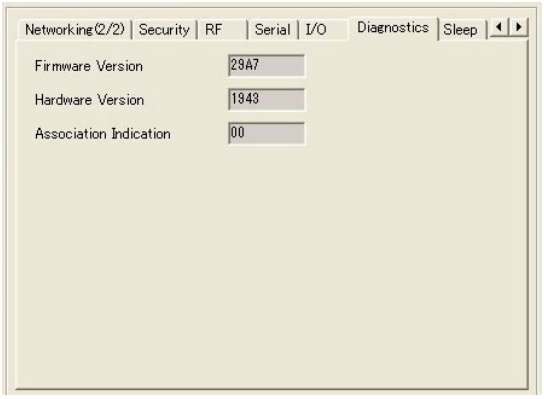
API Options	API Options　対応 AT コマンド:AO
Interface Data Rate	Interface Data Rate　対応 AT コマンド:BD 1200, 2400, 4800, 9600, 19200, 38400,57600, 15200 の各 bps で使用する場合にはラジオボタンで選択してください。任意のボーレートを設定する場合にはテキスト入力欄に直接 16進数で設定してください。
Serial Parity	Serial Parity　対応 AT コマンド:NB
Stop Bits	Stop Bits　対応 AT コマンド:SB

IO タブ



タイトル	説明
IO Sample Rate	IO Sample Rate　対応 AT コマンド:IR
Assoc LED Blink Time	Assoc LED Blink Time　対応 AT コマンド:LT
RSSI PWM Timer	RSSI PWM Timer　対応 AT コマンド:RP
Supply Voltage	Supply Voltage　対応 AT コマンド:%V
Voltage Supply Monitoring	Voltage Supply Monitoring　対応 AT コマンド:V+ 0 を設定するとモニタリングを行いません

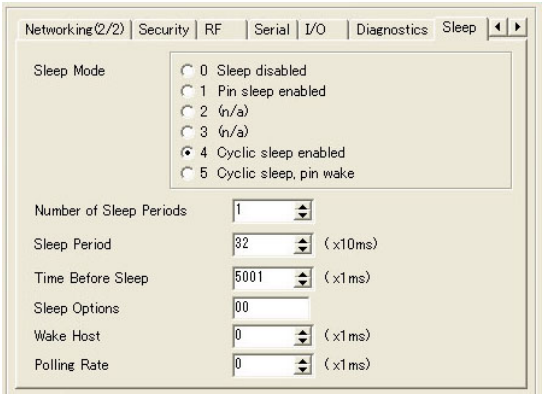
Diagnostics タブ



タイトル	説明

Firmware Version	Firmware Version　対応 AT コマンド:VR
Hardware Version	Hardware Version　対応 AT コマンド:HV
Association Indication	Association Indication　対応 AT コマンド:AI

Sleep タブ



タイトル	説明
Sleep Mode	Sleep Mode　対応 AT コマンド:SM
Number of Sleep Periods	Number of Sleep Periods　対応 AT コマンド:SN
Sleep Period	Sleep Period　対応 AT コマンド:SP
Time Before Sleep	Time Before Sleep　対応 AT コマンド:ST
Sleep Options	Sleep Options　対応 AT コマンド:SO
Wake Host	Wake Host　対応 AT コマンド:WH
Polling Rate	Polling Rate　対応 AT コマンド:PO

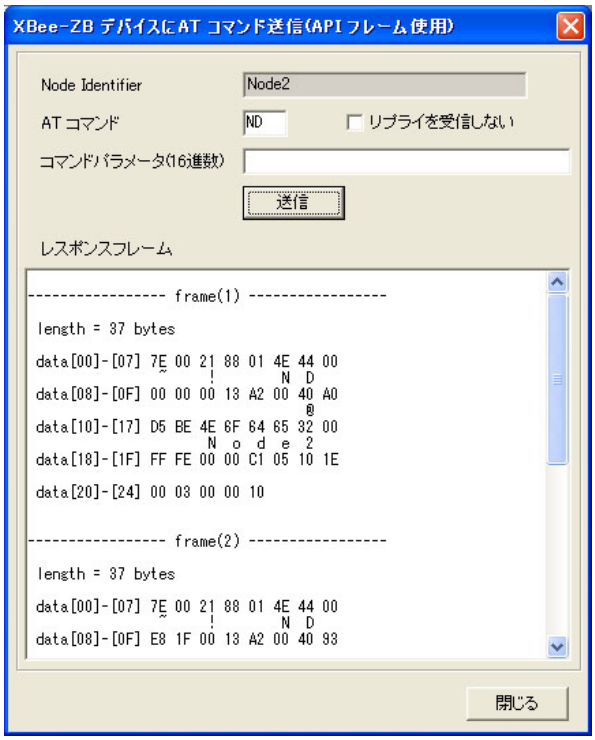
10.5.2　AT コマンド送信機能

XBee-ZB デバイスに任意の AT コマンドを送信してレスポンスフレームを受信する機能です。

詳細設定画面では直接設定できないXBee-ZB の機能を設定する場合や、テスト時などに使用します。

AT コマンドボタンを押すと送信フォームが表示され、任意の ATコマンドとパラメータを送信することができます。XBee デバイスのAT コマンドはパラメータを省略すると現在の設定値をレスポンスフレームに返します。パラメータを指定すると、そのパラメータ値に設定内容が更新されて実行結果ステータスをレスポンスフレームに返します。

選択した XBee-ZB デバイスによって、自動的に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が切り替えて送信します。AT コマンドフレーム中の FrameID は常に 0x01 になります。またリモートコマンド中のコマンドオプションは 0x02(Apply Change on remote) になります。



送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier	(変更不可) コマンド送信対象となる、XBee-ZB デバイスの Node Identifier
AT コマンド	AT コマンド文字列、2 文字のASCII 大文字を指定する
リプライを受信しない	チェックをつけるとレスポンスフレームの受信を省略します。レスポンスを返さない AT コマンドを実行するときにチェックを付けてください。 DeviceServer が XBee-ZB デバイスに AT コマンドを送信したときに、レスポンスフレームを受信できなかった時には自動的に AT コマンドのリトライを行います。"リプライを受信しない" にチェックを付けるとこのリトライ動作は行いません。
コマンドパラメータ	コマンドパラメータを 16進数形式で入力する。例: 0102FF コマンドパラメータを送信しない場合は、入力フィールドを空にすること
レスポンスフレーム	AT コマンド送信後に、XBee-ZB デバイスからのレスポンスフレームが入ります

ATコマンドとコマンドパラメータを入力して、"送信" ボタンを押すと AT コマンドが実行され、レスポンスフレームを表示します。

ATコマンドで 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コマンド "WR" も送信してください。もしくは、AT コマンドを送信した後に、XBee-ZB デバイス詳細設定フォーム (前述) を開いて "設定内容をXBee デバイスの不揮発メモリに書き込む" にチェックをつけて、書き込むこともできます。

終了する場合は、"閉じる" を押してください。

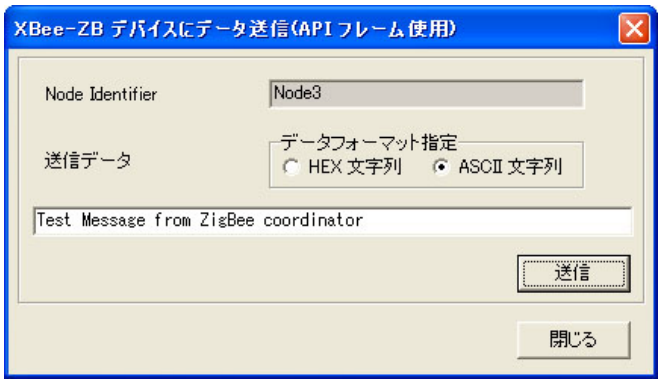
10.5.3 データパケット送信機能

XBee-ZB 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機能です。

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。

データ送信コマンドボタンを押すと送信フォームが表示され、任意のバイナリデータまたはASCII 文字列を送信することができます。バイナリデータを送信する場合は、送信データフィールドに16進数表記の文字列で入力して下さい。例えば 0x0A, 0x0B, 0x0C の３バイトを送信する場合は 0A0B0C を入力します。

データパケットは、XBee API の Transmit Request(0x10)コマンドを使用して送信します。 FrameID は常に 0x01 を使用します。



送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier	(変更不可) コマンド送信対象となる、XBee-ZB デバイスの Node Identifier
データフォーマット指定	バイナリデータを送信する場合は “HEX文字列” を選択する ASCII 文字列を送信する場合は “ASCII文字列” を選択する
送信データ	送信データを 16進数形式または、ASCII で入力する。 送信データを空にするとエラーになりますので注意してください。

”送信” ボタンを押すとデータパケットを作成して送信します。連続して送信を行う場合は、”送信”ボタンを連続して押してください。

終了する場合は、”閉じる” を押してください。

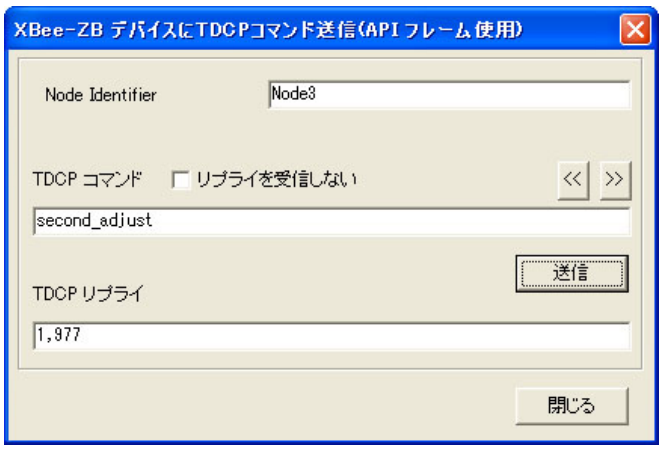
10.5.4 TDCPコマンド送信機能

TDCPプログラムが動作しているCPU ボードに接続されたXBee-ZB デバイスに、TDCPコマンドを送信してリプライ文字列を受信する機能です。TDCPプログラムとCPUボードの詳細については “TDCPユーザーマニュアル”を参照してください。

送信コマンドボタンを押すとTDCPコマンド欄に入力された文字列を送信します。リモートデバイスからリプライパケットを受信するとTDCP リプライ欄にその内容を表示します。

”リプライを受信しない” のチェックボックスをチェックすると、リプライパケットの受信を行いません。TDCP コマンドの “reset” コマンドなど、リプライパケットを送信しないコマンドを実行する場合にチェックを付けます。”リプライを受信しない”のチェックを外しておくことで、リモートからのリプライパケットが受信できない場合には、自動的にコマンド送信のリトライが行われます。

データパケットは、XBee API の Transmit Request(0x10)コマンドを使用して送信します。 FrameID は常に 0x01 を使用します。



送信フォーム中の各項目の内容は以下になります。

タイトル	説明
Node Identifier(list)	コマンド送信対象となる、XBee デバイスの Node Identifierを指定する。 複数のデバイスをカンマで並べて表記した場合には、左に指定したデバイスから順番にコマンドがバケツリレー方式で転送されて、一番右側に指定したデバイスでコマンドが実行される(ルータ経由のコマンド実行)。ルータ機能の詳細はスクリプトライブラリ関数の “xbee_tdcp_far_command()”の説明を参照してください。
TDCPコマンド	TDCPコマンド文字列を入力する
“>>”,”<<”	以前に実行したコマンド文字列をコマンド入力欄に表示する
リプライを受信しない	リモートデバイスからの TDCP リプライパケットを受信しない場合にチェックする。
TDCPリプライ	リモートデバイスで実行したTDCPコマンドのリプライ文字列が設定される

”送信” ボタンを押すとTDCPコマンドを送信します。連続して送信を行う場合は、”送信”ボタンを連続して押してください。

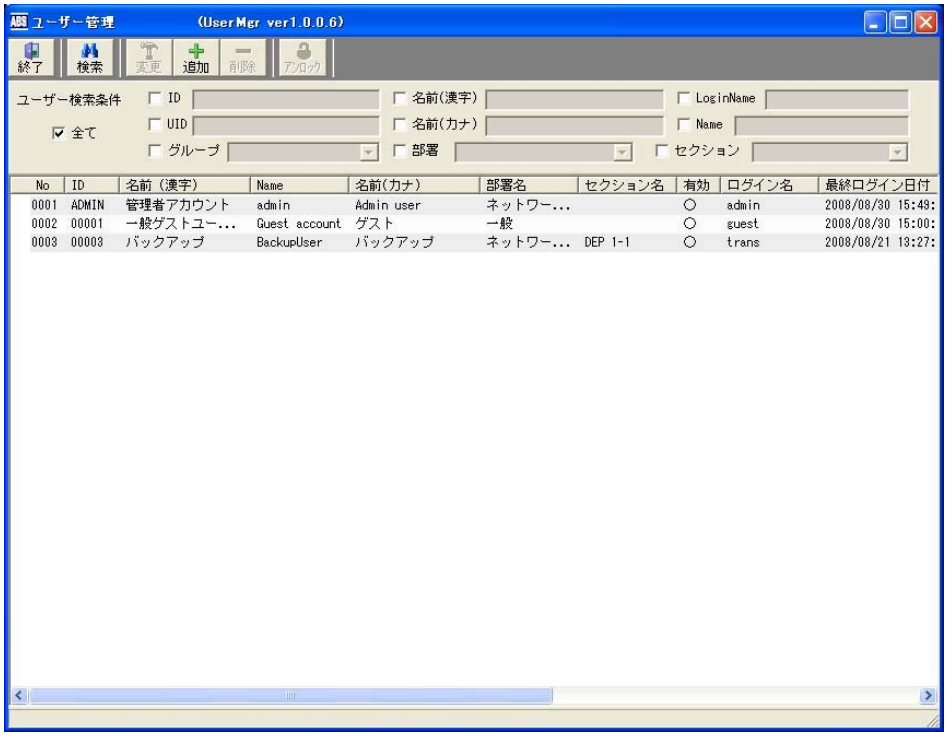
終了する場合は、”閉じる” を押してください。

10.6 ユーザー管理(UserMgr)

DeviceServer で使用する認証機能は、クライアントプログラムやAPI (XASDLCMD.DLL)を利用したユーザー作成のプログラムやEXCEL ワークシート、メール経由でスクリプト実行を行うときに使用する認証機能と共通です。認証に使用するアカウントはユーザー管理プログラムで管理します。ユーザー管理プログラムでは、ユーザーの追加や削除、設定内容の変更ができます。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥UserMgr.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥UserMgr.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

起動すると、ツールボタンと検索条件入力フィールド、ユーザー一覧を表示します。ユーザー一覧を表示するために”検索”ボタンを押すと、全ての登録済みユーザーを表示します。



ユーザー検索条件には、チェックを付けた検索項目で絞込み検索ができます。デフォルトでは “全て”にチェックがついていて、全てのユーザーを検索します。検索条件項目に複数チェックを付けると AND 条件で一致するものだけが検索対象になります。検索条件に指定できる項目を以下に説明します。

タイトル	説明
全て	検索条件を指定しないで、全てのユーザーを検索します。デフォルト指定。

ID	ユーザーID文字列を指定します。
名前（漢字）	名前を指定します。部分一致条件で検索します。
LoginName	LoginNameを指定します。部分一致条件で検索します。
UID	DeviceServer がユーザー毎に付けたユニークな文字列コードです。
名前（カナ）	名前を指定します。部分一致条件で検索します。
Name	Nameを指定します。部分一致条件で検索します。
グループ	グループを指定します。
部署	登録済みの部署から選択します。
セクション	登録済みのセクションから選択します。

検索結果のユーザー一覧に表示される項目を以下に説明します。また、一覧の項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	検索結果リスト順に0001 から順に自動でアサインされます。
ID	ユーザーID文字列
名前（漢字）	ユーザー名（漢字）
Name	UserName
名前（カナ）	ユーザー名（カナ）
部署名	ユーザー部署名
セクション名	ユーザーセクション名
有効	ログインが有効な場合は 〇が付く。ユーザーのアプリケーション許可フラグの[AllowLogin]フラグにチェックが付いている場合に有効になります。
ログイン名	ログイン名
最終ログイン日付	最後にログイン操作を行ったときの日時
使用有効期限	ユーザーに使用開始日付が設定されていた場合の日付
使用開始日付	ユーザーに使用終了日付が設定されていた場合の日付
UID	DeviceServer がユーザー毎に付けたユニークな文字列コード
最終更新日付	DeviceServer がユーザー情報を更新した最後の日時。ログイン、ログアウト、ユーザー情報更新などの操作でこの値は更新されます。

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
終了 	UserMgr プログラムを終了します。ログアウトは行わずに、UserMgr のメインフォームのみが閉じられます。ログアウトを行う場合は、デスクトップ(ABDesktop)の終了ボタンを押してください。
更新 	検索条件に一致したユーザー情報を検索して、ユーザー一覧を最新の情報に更新します。
変更 	ユーザーを選択して、ユーザー情報を変更します。ユーザー情報設定画面（後述）を表示します。
追加 	ユーザーを追加します。ユーザー情報設定画面（後述）を表示します。
削除 	ユーザーを選択して、ユーザーを削除します。 一度ユーザーを削除すると、元には戻せませんので注意してください。一時的にユーザーのログインを不可にしたいだけであれば、ユーザー情報変更を行って、アプリケーション許可フラグの[AllowLogin]フラグのチェックを外す操作を行ってください。
アンロック 	ユーザー情報を変更途中にシステムが停止し、データベースに更新中フラグがついたままになった場合に、強制的にフラグを解除して変更可能な状態に戻します。 ロック中は一覧を表示したときに、鍵マークを左端に表示します。通常運用時はアプリケーションから対象ユーザー情報が更新中を意味します。（通常運用時にこのボタンを操作することはありません）

ユーザー追加・変更時に表示される設定画面の項目説明は以下になります。

ダイアログのタイトルに “*” が付いているものは必須入力項目です。

基本情報	
タイトル	説明
*UID	DeviceServer がユーザー毎に付けたユニークな文字列コード。デフォルトのまま変更しないで下さい。
参照コード	ユーザーの管理番号等を入れます。通常は空白にしてください。
*ユーザーID	ユーザーID文字列。アルファベットまたは数字で 64文字以内にしてください。
自動でIDを設定する	チェックを付けると、ユーザーID 文字列を連番で自動的に設定します。
Name(アルファベット)	ユーザー名。アルファベットまたは数字で 128文字以内にしてください。
名前(漢字)	ユーザー名(漢字)
名前(カナ)	ユーザー名(カナ)
*ログイン名(LoginName)	ログイン時のユーザー認証に使用する名前。アルファベットまたは数字で 64文字以内にしてください。
新規または既存のパスワードを	チェックを付けると、新規または既存のパスワードを上書き設定します。

上書きする	
ログインパスワード	ログイン時のユーザー認証に使用するパスワード。アルファベットまたは数字で64文字以内にしてください。
ログインパスワード(再入力)	ログイン時のユーザー認証に使用するパスワード。アルファベットまたは数字で64文字以内にしてください。
付加情報	
郵便番号	7 桁の郵便番号。検索ボタンを押すと、入力された郵便番号から住所を検索することができます。
住所	住所。検索ボタンを押すと、入力された住所から郵便番号を検索することができます。
電話番号１	電話番号１
電話番号２	電話番号２
E-Mail	電子メールアドレス
メールコマンド応答先	メールコマンド応答先の電子メールアドレス。
施設名	施設名
役職・担当等	役職・担当等
所属セクション	所属セクション
有効期間	
使用開始日付を設定する	チェックを付けると、使用開始日付欄に入力した日時以降にならないとログインできなくなります。
使用開始日付	ログイン可能な開始日付
使用終了日付を設定する	チェックを付けると、使用終了日付欄に入力した日時以降にはログインできなくなります。
使用終了日付	ログイン可能な終了日付
同時にログイン可能なセッション数を設定する	チェックをつけて、右欄のセッション数を設定すると同時に同一ユーザーでログイン可能な人数を指定することができます。0 を入力することで、同一ユーザーが別ログインを、させないようにすることができます。ここに指定した値を超えていなくても、ライセンスで同時ログイン数が決められている場合はそちらの制限を優先します。
グループ	
現在の全てのグループ一覧	DeviceServer に登録済みの全グループ一覧 グループを選択して、“>” ボタンを押すと、ユーザーを選択したグループに加えます。
上記以外の新規グループの場合	現在の全てのグループ一覧にはない、新規のグループにこのユーザーを加える場合に、グループ名を入力して、”>”ボタンを押します。
このユーザーが属しているグループ	ユーザーが現在属しているグループです。
グループから外す	ユーザーが属しているグループを選択して、ボタンを押すと選択したグループから

	ユーザーを外します。
アプリケーション許可	
このユーザーに設定されたフラグ	ユーザーが実行可能なアプリケーションや動作モードを指定します。 詳細は、ダイアログ右欄に表示される説明文を参照してください。
履歴	
最終ログイン日付	ユーザーが最後にログインを行った日時
最終ログアウト日付	ユーザーが最後にログアウトを行った日時
変更履歴	“変更” ボタンでユーザー情報を変更したときの履歴。 変更時にコメント入力をした場合は、その内容も表示される。

10.7 スクリプトテスト(ScriptTest)

DeviceServer のスクリプトフォルダに設定したスクリプトを、クライアント PC から実行するためのプログラムです。実行時に任意のパラメータを指定することができます。スクリプト中でリターン値を設定した場合は、その内容を画面に表示します。主にスクリプト作成時のテスト用に使用します。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ScriptTest.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ScriptTest.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

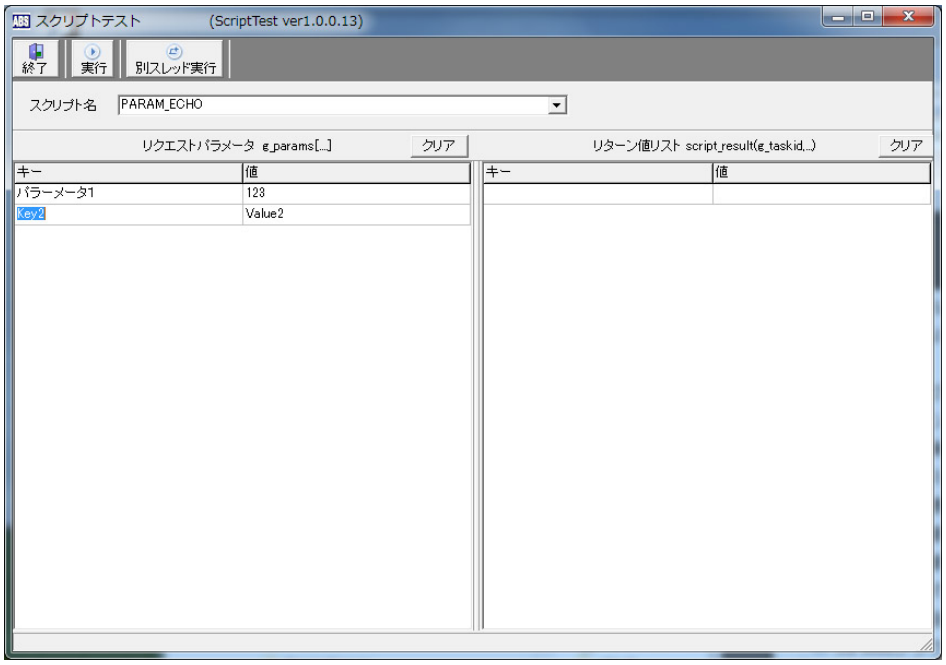
起動すると、ツールボタンとスクリプト名入力コンボボックスを表示します。
コンボボックスのプルダウンリストには、DeviceServer のスクリプトフォルダに設定されている全てのスクリプト名が入っています。実行したいスクリプトをプルダウンリストから選択するか、もしくはキーボードからスクリプト名を直接入力してください。

リクエストパラメータ部分に、キーと値をペアで指定することで任意のパラメータをスクリプトに渡すことができます。入力したい部分をマウスでクリックすると入力状態になりますので、キーボードから文字列を入れてください。”DEL”キー で入力したデータを削除できます。”キー” と ”値” の両方のデータを削除すると、その行に指定したパラメータ自身が消去されます。”DOWN” カーソルを押すことで、パラメータを必要なだけ追加することができます。

“クリア”ボタンを押すと、リクエストパラメータまたはリターン値リスト全体を削除できます。

″実行″ボタンを押すとスクリプトを実行します。スクリプト中で script_result() ライブラリ関数を使用してリターンパラメータを設定していた場合には、リターン値リスト部分に設定されます。
″別スレッドで実行″ボタンを押すとサーバー側で別のスレッドを新規に作成して実行します。スクリプト中で時間がかかる処理や、無限ループに入るようなデーモンタイプのスクリプトを起動する場合に使用します。ボタンを押すとスクリプト実行が開始されて直ぐに制御が返ります。現在のスクリプト実行状態を確認するには、″タスク管理・スクリプトセッションプール(TaskList)″ プログラムを使用します。別スレッドで実行したスクリプトのリターン値は取得できませんので、グローバル共有変数などに一旦格納しておいて、別のスクリプトからそれらの値を取得するよ

うにします。



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
終了 	ScriptTest プログラムを終了します。ログアウトは行わずに、ScriptTest のメインフォームのみが閉じられます。ログアウトを行う場合は、デスクトップ(ABDesktop)の終了ボタンを押してください。
実行 	スクリプト名に指定したスクリプトを実行します。リクエストパラメータを指定した場合は、スクリプト内の g_params[] グローバル変数にその内容を渡します。スクリプト中で script_result() 関数を使用してリターン値を設定した場合は、その内容がリターン値リストに表示します。
別スレッド実行 	スクリプト名に指定したスクリプトをサーバー側で別スレッドで実行して直ぐに制御が返ります。リクエストパラメータを指定した場合は、スクリプト内の g_params[] グローバル変数にその内容を渡します。別スレッドで実行したスクリプトのリターン値は取得できません。

11 その他のプログラム

DeviceServer のセットアップで提供されるその他のプログラムについて、説明します。

11.1 アラームシグナル(AlarmSignal)

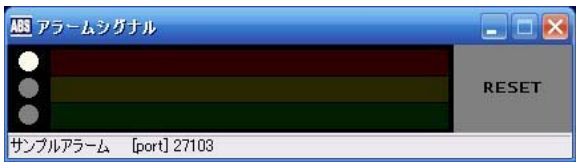
SigSensor と同様のアラーム機能(ランプ出力とブザー音出力のみで I/O は持たない)のソフトウエアです。ネットワークに接続された PC 上で起動して、DeviceServerから操作することができます。同一 PC 上に複数のインスタンスを同時に起動することもできます。アラームシグナルプログラムは、SigSensor, NetUIO の様なハードウエアを必

要としないので、手軽にアラームデバイスを DeviceServer に接続できます。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥AlarmSignal.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥AlarmSignal.exe
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
起動パラメータ	AlarmSignal.exe <PortNumber> <"デバイスコメント文字列"> 起動パラメータを指定すると、AlarmSignal 起動時に自動的にONLINE 状態になります。 <PortNumber>: デバイスのポート番号。通常は 27103 を指定 <"デバイスコメント文字列">: フォームのステータスラインに表示するデバイスコメント

起動すると、左端にステータスランプ(丸パネル)と中央にアラームランプ（赤、黄、緑のランプ）、右端にリセットボタンが配置されています。ステータスランプの一番上が ONLINE 状態とスイッチを兼ねていて白色が ONLINE を示します。OFFLINE と ONLINE の切り替えはマウスでクリックすることで行います。その下の２つのステータスランプはそれぞれ BEEP1, BEEP2 のブザー音出力時に同時に赤く点灯します。

中央のアラームランプはそれぞれ、シグナルがセットされたときに点灯または点滅します。右のリセットを押すと、全てのアラームランプとブザー出力を停止されます。



11.2 セッション管理(SessionList)

DeviceServer にログイン中のセッションの確認や、強制的にログインセッションを削除するためのプログラムです。DeviceServer をインストールした PC 上でのみ実行できます。

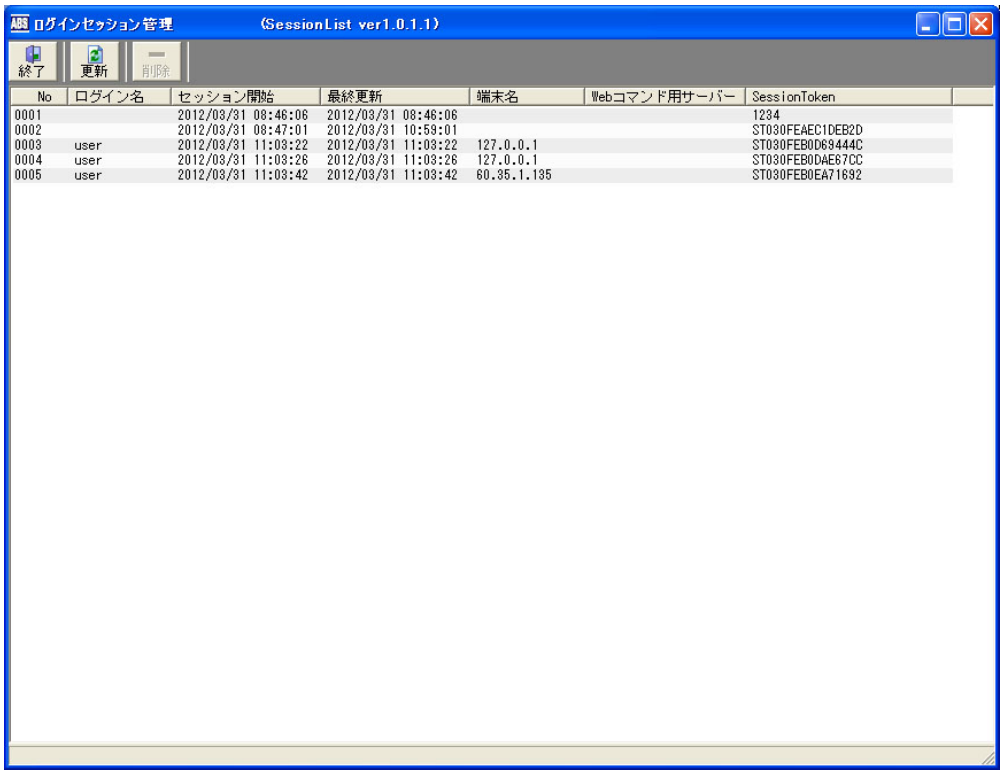
Webからログインしてクライアント機能を利用している状態で、ログアウト操作(WebのFlash アプリケーションのメインフォーム左上の“終了”ボタンを押す) を行わないで、Webブラウザ画面を閉じた場合に、 DeviceServer にはログインセッションが残されたままになっています。サーバー設定プログラムの “操作が無い場合の自動ログアウト待ち時間” で設定した期間を過ぎると自動的にセッションは削除されますが、セッション管理プログラムから操作することで、直ぐに該当セッションを削除することができます。

SessionListはログイン操作などを必要としないで、現在のセッション数の制限に関係なく DeviceServer の動作 PC 上で常に起動可能になっています。このため、DeviceServer の動作 PC 上で操作を行うWindows アカウントには必

ずパスワードを付けて、セキュリティを確保してください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥SessionList.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥SessionList.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

起動すると、ツールボタンと現在のセッション一覧を表示します。



検索結果のセッション一覧に表示される項目を以下に説明します。また、一覧の項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	検索結果リスト順に0001 から順に自動でアサインされます。
ログイン名	ログイン名
セッション開始	ログインしてセッションを開始した日時
最終更新	DeviceServer がセッション情報を更新した最後の日時。クライアントから DeviceServer に対してコマンドのやり取りを行うと更新される。 現在の時刻とこの値の差が、サーバー設定プログラムの “操作が無い場合の自動ログアウト待ち時間”で設定した値よりも大きくなると、セッションはDeviceServerで自動削除されます。
端末名	ログイン操作を実行したクライアントPC のホスト名または IP アドレス ライブラリ関数 create_session() を使用して作成したセッションは、ログイン操作を伴っていませんので端末名は表示されません。

SessionToken	DeviceServer がセッション毎に付けたユニークな文字列

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
終了 	SessionList プログラムを終了します。
更新 	セッション一覧を最新の情報に更新します。
削除 	セッションを選択して、強制的にログインセッションを終了します。 強制的にログインセッションを終了した場合、ユーザー情報の最終ログアウト日時は更新されません。

11.3 タスク管理・スクリプトセッションプール(TaskList)

DeviceServer で実行中のスクリプト一覧を表示することや、強制的に実行中のスクリプトタスクを終了させるためのプログラムです。DeviceServer をインストールした PC 上でのみ実行できます。

DeviceServer でイベントハンドラやユーザースクリプトが実行される時には、複数のLua スクリプトエンジンの実行インスタンスから構成されているセッションプール(スクリプトセッションプール) から、現在使用されていないエントリを見つけて実行させます。デフォルトではセッションプールの数は最大 16 個確保されていて、これ以上のスクリプト実行リクエストは、現在実行中のスクリプトまたはイベントハンドラが終了するまでペンディングします。

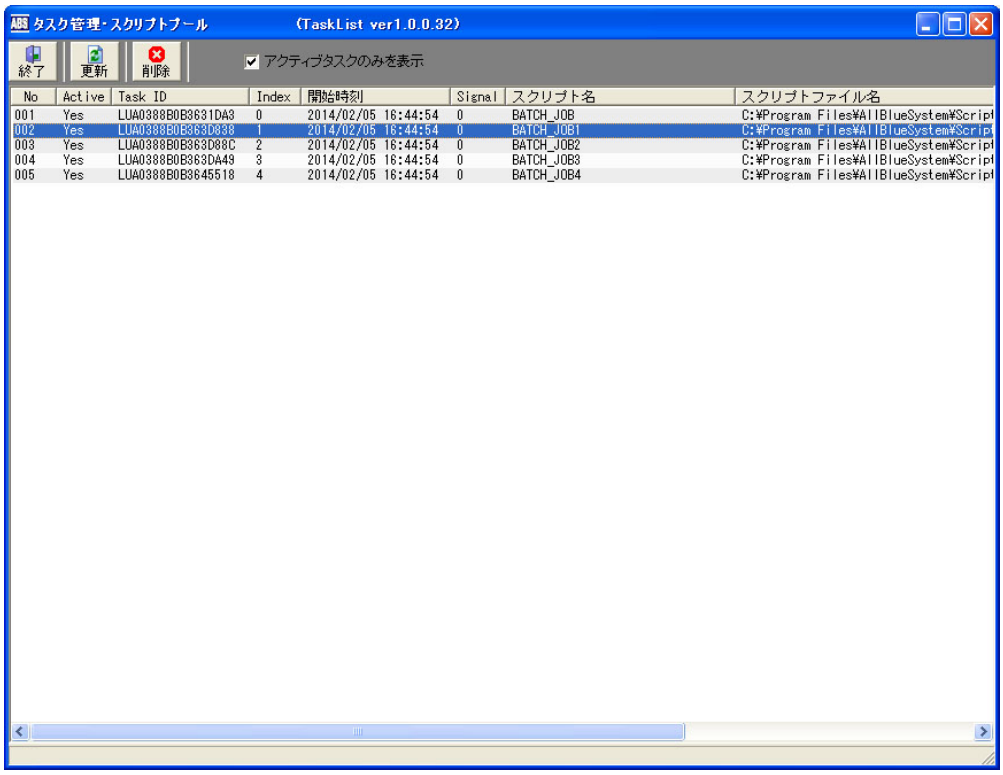
TaskList プログラムを起動すると現在実行中のスクリプトの一覧を表示します。実行中のタスクを選択して削除ボタンを押すと、該当するスクリプトタスクを強制的に終了させることができます。実行中のスクリプトタスクがライブラリ関数 wait_time(), event_wait(), critical_section_enter() 等を使用していて、ライブラリ関数内部でwait状態になっている場合には、これらの関数を抜けた後にスクリプトタスクの実行が強制的に終了します。タスクが強制的に終了された時に、ログには下記の様なメッセージが記録されます。

```
ScriptHookFunc:*WARNING* signal[9] received, g_taskid = LUA031E86E862EDCE
TForkScriptExecThread:*EXCEPTION* lua_pcall error detected
```

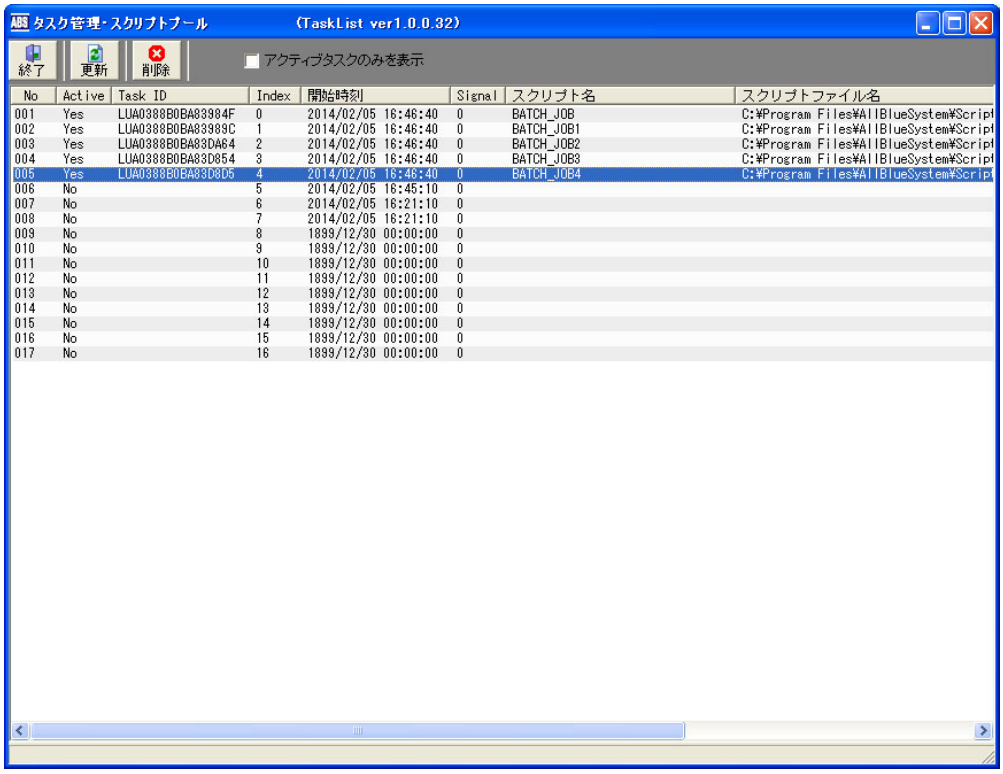
TaskListプログラムはログイン操作などを必要としないで、現在のセッション数の制限に関係なく DeviceServer の動作 PC 上で常に起動可能になっています。このため、DeviceServer の動作 PC 上で操作を行うWindows アカウントには必ずパスワードを付けて、セキュリティを確保してください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥TaskList.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥TaskList.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

起動すると、ツールボタンと現在実行中のスクリプト(タスク)一覧を表示します。



デフォルトでは実行中のタスクのみを表示します。もし何もタスクを実行していない場合にはリストは空になります。更新ボタンを押すことで最新の情報に更新できます。ツールボタン上の“アクティブタスクのみを表示”のチェックを外すと、セッションプール全体の状態を表示します。この場合には現在実行中ではない Lua スクリプトエンジンのエントリを含めて、全てのタスクを表示します。



スクリプト(タスク)一覧リストに表示される項目を以下に説明します。また、リストの項目タイトル部分をクリックするとソートを行います。クリックする毎に昇順・降順が切り替わります。

タイトル	説明
No	スクリプト(タスク)リスト順に001 から順に自動でアサインされます。
Active	タスク実行中の場合には “Yes” が入ります。 実行していない場合には “No” が入ります。この場合には検索リスト中の他の項目データ内容は使用されていません。
TaskID	実行中のスクリプトにアサインされた TaskID 実行中のスクリプトでg_taskid グローバル共有変数にアサインされた値と同じ内容
Index	スクリプトセッションプール中のインデックス番号 スクリプト実行リクエストをDeviceServer が受け付けると、この番号の少ないものから順に空きエントリを見つけて実行します。
開始時刻	DeviceServer がスクリプト又はイベントハンドラを開始した時刻
signal	実行中のスクリプトに送信されたシグナル情報。通常は 0 が設定されています。 タスクを強制的に終了させる場合にはシグナル値 9 に設定されます。
スクリプト名	クライアントプログラムや API 関数から起動したときに指定したスクリプト名
スクリプトファイル名	実行中のスクリプトファイルのファイル名（Windows ファイルパス名）

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
終了 	TaskList プログラムを終了します。

更新 	スクリプト(タスク)一覧を最新の情報に更新します。
削除 	実行中のタスクを選択して、強制的にタスクを終了させます。 複数のタスクを選択して終了させることもできます。

11.4 スクリプト実行プログラム・単独アプリGUI版(ScriptExecDemo)

DeviceServer でスクリプト実行を行うプログラムです。任意のクライアントPC から実行できます。実行する PC には予め XASDLCMD.DLL をインストールしておく必要があります。（インストールの方法は、”インストール” の章中の “ユーザーアプリケーションを利用する場合(APIライブラリを使用)” を参照してください）

プログラムファイル名	ScriptExecDemo.exe（アーカイブファイルを展開してください）
アーカイブファイル名	C:¥Program Files¥AllBlueSystem¥Demo¥ScriptExecDemo.zip Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥Demo¥ScriptExecDemo.zip

プログラムを起動すると、メインフォームを表示します。

最初に、“ログイン” ボタンを押して DeviceServer にログインします。”スクリプト実行” ボタンを押すと “スクリプト名”に指定したスクリプトを実行することができます。”パラメータをスクリプトに渡す” にチェックをつけておくと、指定したキーと値のペアをスクリプトパラメータとして渡すことができます。

“スクリプト実行(繰り返し)” を押すと、”繰り返し間隔” で指定した間隔で定期的にスクリプト実行を繰り返します。“繰り返しの中止”ボタンで繰り返しを停止します。

11.5 スクリプト実行プログラム・コマンドライン版(ScriptExecCmd)

DeviceServer でスクリプト実行を行うコマンドラインプログラム（コマンドライン版）です。Windows のタスクスケジューラを使用して予めスケジュールされたタイミングでスクリプト実行したいときに使用できます。

このプログラムは別 PC にコピーして実行することもできます。任意のフォルダに ScriptExecCmd.exe をコピーして使用できます。このとき、ScriptExecCmd.exe プログラムを実行する PC には、予め XASDLCMD.DLL ファイルを Windows システムフォルダにインストールしておく必要があります。（インストールの方法は、”インストール” の章中の “ユーザーアプリケーションを利用する場合(APIライブラリを使用)” を参照してください）

プログラムファイル名	ScriptExecCmd.exe DeviceServerをインストールしたフォルダに格納されています。 任意のフォルダや別PC にコピーして使用することができます。別PC にコピーする場合には ″XASDLCMD.DLL″ ファイルも Windows システムフォルダにコピーしてください。
起動パラメータ	ScriptExecCmd.exe <ScriptName> <HostName> <UserName> <Password>
設定ファイル	ScriptExecCmd.ini プログラムファイルを配置した同一フォルダに置かれる。ファイルが存在しない場合は起動時にファイルが作成される。

コマンドプロンプトから上記プログラムファイルを起動します。(サーバーホスト名が “localhost” ユーザー名が “guest” パスワードが “mypassword” で、スクリプト SAMPLE を実行します)

```
C:¥Program Files¥AllBlueSystem>cd ″C:¥Program Files¥AllBlueSystem″
C:¥Program Files¥AllBlueSystem>scriptexeccmd.exe SAMPLE localhost guest mypassword
C:¥Program Files¥AllBlueSystem>
```

プログラムは、設定ファイルを使用して実行時に指定するパラメータを省略することや、スクリプトパラメータを指定することができます。設定ファイルが存在しない状態でプログラムを起動すると、デフォルトの設定ファイルがプログラムファイルと同一フォルダに作成されます。設定ファイル内の項目は以下になります。

項目名	説明
ScriptName	実行するスクリプト名
HostName	DeviceServer のホスト名
UserName	ログインユーザー名
Password	パスワード
KeyList	スクリプトパラメータを渡す場合は、カンマ区切りでキー名を入力する
ValList	スクリプトパラメータを渡す場合は、カンマ区切りでキー名に対応する値を入力する。KeyList とValList のカンマ区切りで指定する項目数は同一にすること。（下記

	例を参照の事）

設定ファイル中に項目の内容を記述すると、プログラム起動時に、実行時の起動パラメータの指定を省略できます。下記の例の様に設定ファイルを作成すると、実行時の起動パラメータはスクリプト名のみ指定することで実行できます。

```
[ScriptExecCmd]
ScriptName=
HostName=servername
UserName=guest
Password=mypassword
KeyList=Parm1,出力,ABC##123
ValList=Val1,値,DEF##999
```

ScriptExecCmd.exe プログラムのソースファイルは DEMO フォルダ内に zip 形式で提供しています。DLL ライブラリで提供する API 関数を使用するプログラム作成時の参考にしてください。**このソースファイルはサポート対象外となります。**デモソースファイルのコンパイルは Delphi(win32 version 6 Update Pack2)で試験を行っておりますが、Delphi のバージョンによってはソースやプロジェクトファイル修正が必要な場合があります。

11.6 UDPServerデモプログラム (UDPServer)

簡易の UDPServer プログラムです。ポート番号を指定して任意のUDP サーバーとして動作させることができます。受信可能なパケットはテキスト文字列のみで、画面にそのまま表示します。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、ネットワークリソースを使用可能な様に設定変更が必要な場合があります。詳しくはご使用中のファイアウォールプログラムまたはセキュリティソフトのマニュアル等を参照ください。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥Demo¥UDPServer.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥Demo¥UDPServer.exe

プログラムを起動すると、メインフォームを表示します。
ポート番号を指定して、”オンライン”にチェックを付けると、UDPサーバーとして動作します。”相手アドレス”にチェックを入れておくと、接続先のアドレスを表示します。
UDPパケットを受信すると、内容をテキスト文字として解釈して画面に１行で表示します。”クリア”ボタンを押すと、画面の内容を消去します。ポート番号を変更する場合は、一旦”オンライン”のチェックを外してから操作して下さい。

UDPServer デモプログラムは、スクリプトライブラリ関数の udp_send_data() 関数を使用する時に、ユーザー側のUDPサーバープログラムが準備されていない状態でも、動作確認可能なように提供しています。**このプログラムはサポート対象外となります。**

11.7 CPU情報表示(ShowCPUInfo)

DeviceServer の動作しているPC の情報を表示するプログラムです。DeviceServerライセンスをご購入される場合に、このプログラムで表示される内容を使用します。

プログラムファイル名	C:¥Program Files¥AllBlueSystem¥ShowCPUInfo.exe Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥ShowCPUInfo.exe
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

プログラム起動時に CPU 情報をメインフォームに表示します。

DeviceServer ライセンスについては、”初期設定”の章中の ”DeviceServerライセンスについて”の項目を参照してください。CPU情報表示プログラムを使用した、ライセンス購入の方法につきましては、DeviceServer のインストールフォルダのEtcフォルダ “C:¥Program Files¥AllBlueSystem¥Etc” または “C:¥Program Files (x86)¥AllBlueSystem¥Etc”に格納された “デモライセンス.txt” ファイルに記載してありますので参照して下さい。

12　イベント

DeviceServer では 各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)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。

インストール直後の各イベントハンドラのスクリプトファイルには、イベント発生をログに記録するだけのスクリプトなどが記述されています。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-8N形式で記述)ですのでテキストエディタで簡単に変更できます。スクリプトファイル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PC や DeviceServer を再起動する必要はありません。

システムの目的に合わせてこれらのイベントハンドラスクリプトファイルをカスタマイズして運用します。

DeviceServer では、イベントハンドラもユーザーが独自に作成するスクリプトと同様の Lua スクリプトとして扱いますので、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、コールする側で与える必要があります）

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 "C:¥Program Files¥AllBlueSystem¥Scripts" または "C:¥Program Files (x86)¥AllBlueSystem¥Scripts"フォルダに保管されています。

12.1　MAIL_NEW_MAILイベント

- **イベント発生条件**

DeviceServer が POP サーバー上のメールを検索したときに、新規メールが見つかった場合に発生します。イベントは、各新規メール単位に発生します。（メール検索時に２つの新規メールが見つかった場合は、合計２回連続してイベントが発生します）

予め "サーバー設定" プログラムで、POP サーバー設定を行って、" メール機能を有効にする"、"受信メールをチェックする"、"新しいメール受信時にスクリプト実行"の３つを有効にしておく必要があります。

- **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
$RECIPIENTS$	メールの宛先アドレス	xxxさん <xxxx@your.corp.jp>
$FROM$	メールの送信元アドレス	abc@xxxx.yyyy.zzzzz
$SUBJECT$	メールの件名	お知らせメールです

ID	メールのID	id123.456.789.abc
$PRIORITY$	メールのプライオリティ	3
$MAILBOX$	メールボックス中のメール数	10
$TIMESTAMP$	メールの送信日時	2000/01/01 01:00:01
$SCANTIMESTAMP$	メール検索を行った日時	2000/01/01 01:05:31

- **備考**

$SCANTIMESTAMP$ の値はメールを検索したときの時刻が入ります。新規メールが複数あった場合には、それぞれのメール毎に呼び出された、MAIL_NEW_MAIL イベントでは同一の値になります。

$MAILBOX$ の値は、イベント発生時に POP サーバーにあるメールの総数を示します。メールクライアントアプリケーションでメールを取得・表示する時に、POP サーバーのメールを削除する設定にしている場合は、常に未読メール数に等しくなります。

メールヘッダのMIME デコード時に、スペース文字が余分に入ったり抜けたりする場合があります。（日本語 <-> アルファベット の区切り部分に多い）文字列のマッチをスクリプト中で操作する場合は、これらを考慮して実際のメールを受信して、パターンマッチの調整が必要になります。送信元で記入したヘッダと、完全には一致しない場合がありますので注意してください。メールの転送経路によってもヘッダ内容が変化する場合があります。これらのヘッダ情報中のスペース文字の増減は、使用上の制限事項になります。

- **イベントハンドラ例**

```
-----------------------------------------------------
-- 新着メールの数を SigSensor デバイスの LCD に表示する
-----------------------------------------------------
stat = alarm_signal_message("SigSensor"," ** new mail ** mail box: " .. g_params["$MAILBOX$"])
if not stat then error() end
```

12.2 PERIODIC_TIMERイベント

- **イベント発生条件**

約 1 分毎に発生します。

予め “サーバー設定” プログラムで、”定期的にスクリプトを実行する”を有効にしてください。

- **イベントハンドラに渡されるパラメータ**

なし

- **備考**

実行間隔を大きくしたい場合や、定時のスケジュール実行を設定したい場合には、PERIODIC_TIMER イベントではなく、Windows OS のタスクスケジューラを利用することを検討してください。Windows OS の “at”, ”schtasks” コマンドと、DeviceServer の ScriptExecCmd.exeプログラム(¥Demoフォルダにインストールされています)を組み合わせて使用することで、簡単にスクリプト実行をスケジュールできます。

イベントハンドラ例

```
------------------------------------------------------------
-- DeviceServer 起動時に一回だけ SAMPLE スクリプトを実行する
------------------------------------------------------------
stat,val = get_shared_data("STARTUP_SCRIPT")
if not stat then error() end
if val == "" then
   if not inc_shared_data("STARTUP_SCRIPT") then error() end
   if not script_exec("SAMPLE","スタートメッセージ","DeviceServer起動") then error() end
end
```

12.3 XBEE_MODEM_STATUSイベント

- **イベント発生条件**

XBEE サービスモジュールで、API タイプ（0x8A）のフレームを受信した時に発生します。

予め“サーバー設定”プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行”を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	8A
ModemStatus	フレームデータ中のStatus(10進数)	6

- **備考**

フレーム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が実行されます。ただし、DeviceServer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

```
File_id = "XBEE_MODEM_STATUS"
if g_params["ModemStatus"] == "0" then log_msg("Hardware reset",file_id)
elseif g_params["ModemStatus"] == "1" then log_msg("Watchdog timer reset",file_id)
elseif g_params["ModemStatus"] == "2" then log_msg("Associated",file_id)
elseif g_params["ModemStatus"] == "3" then log_msg("Disassociated",file_id)
elseif g_params["ModemStatus"] == "4" then log_msg("Synchronization lost",file_id)
elseif g_params["ModemStatus"] == "5" then log_msg("Coordinator realignment",file_id)
elseif g_params["ModemStatus"] == "6" then log_msg("Coordinator started",file_id)
```

```
else log_msg("Unknown status " .. g_params["ModemStatus"],file_id)
end
```

12.4 XBEE_IO_DATAイベント

- **イベント発生条件**

XBEE サービスモジュールで、API タイプ (0x82)または (0x83) の I/O データフレームを受信した時に発生します。予め “サーバー設定” プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	8A
SourceAddress	フレームデータ中のSourceAddress 16bit アドレスの場合(16進数4桁) 64bit アドレスの場合(16進数16桁)	 0A01 0013A200404AC397
SerialNumber	XBee デバイスの SerialNumber DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにSerialNumber未登録の場合は"" が設定される	0013A200404AC397
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにNodeIdentifier未登録の場合は"" が設定される	Device1
RSSI	フレームデータ中のRSSI(16進数2桁)	45
Options	フレームデータ中のOptions(16進数2桁)	00
SAMPLE_COUNT	I/O データのサンプル数	1
SAMPLE_DIO	I/O データ中のサンプル対象となったDIOビット番号リスト (10進数、カンマ区切り)	0,1,4
SAMPLE_ADC	I/O データ中のサンプル対象となったADCビット番号リスト (10進数、カンマ区切り)	2,3
SAMPLE_<Sample#>_<"DIO"|"ADC">_<Bit#>	I/O サンプルデータ値 DIO の場合は、High で "1"、Low で "0" ADC の場合は 10進数	 1 1023

- **備考**

サンプルデータを示すパラメータ名は、<Sample#> には 最大、SAMPLE_COUNT まで 1から順番にインクリメントされた値が入ります。<"DIO"|"ADC">は、I/O サンプルデータが ADC もしくは、DIO のどちらであるかを示します。<Bit#>

は、サンプルデータのビット番号が入ります。

例：サンプル総数が 2 で DIO のサンプル対象ビットが 0,1,4,6 、ADC のサンプル対象ビットが 2,3 の場合には、イベントハンドラが実行時に、下記のようなパラメータが設定されます。(g_params を スクリプト中から pairs() 関数でログ出力するときには、表示順序が異なる場合があります)

```
g_params[APIType] = 83
g_params[SourceAddress] = 0A01
g_params[NodeIdentifier] = Device1
g_params[RSSI] = 37
g_params[Options] = 00
g_params[SAMPLE_COUNT] = 2
g_params[SAMPLE_DIO] = 0,1,4,6
g_params[SAMPLE_ADC] = 2,3
g_params[SAMPLE_1_DIO_0] = 0
g_params[SAMPLE_1_DIO_1] = 1
g_params[SAMPLE_1_DIO_4] = 1
g_params[SAMPLE_1_DIO_6] = 0
g_params[SAMPLE_1_ADC_2] = 657
g_params[SAMPLE_1_ADC_3] = 864
g_params[SAMPLE_2_DIO_0] = 0
g_params[SAMPLE_2_DIO_1] = 1
g_params[SAMPLE_2_DIO_4] = 1
g_params[SAMPLE_2_DIO_6] = 0
g_params[SAMPLE_2_ADC_2] = 657
g_params[SAMPLE_2_ADC_3] = 864
```

フレーム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、DeviceServer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

```
for key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.5 XBEE_IO_OTHERイベント

- **イベント発生条件**

XBEE サービスモジュールで、API タイプ（0x82)または（0x83）を受信して、かつI/O データパケット以外のデータフレームを受信した時（XBEE_IO_DATA イベントの I/O データパケットの解析に失敗したとき）に発生します。予め“サーバー設定”プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行”を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	8A
SourceAddress	フレームデータ中のSourceAddress 16bit アドレスの場合(16進数4桁) 64bit アドレスの場合(16進数16桁)	 0A01 0013A200404AC397
SerialNumber	XBee デバイスの SerialNumber DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにSerialNumber未登録の場合は"" が設定される	0013A200404AC397
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにNodeIdentifier未登録の場合は"" が設定される	Device1
RSSI	フレームデータ中のRSSI(16進数2桁)	45
Options	フレームデータ中のOptions(16進数2桁)	00
RFData	フレームデータ中の受信したRFData (16進数、可変長)	0102030A0B

- **備考**

フレーム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、DeviceServer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

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.6 XBEE_PACKET_DATAイベント

- **イベント発生条件**

XBEE サービスモジュールで、API タイプ (0x80)または (0x81) のフレームを受信した時に発生します。
予め “サーバー設定” プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	81
SourceAddress	フレームデータ中のSourceAddress 16bit アドレスの場合(16進数4桁) 64bit アドレスの場合(16進数16桁)	 0A01 0013A200404AC397
SerialNumber	XBee デバイスの SerialNumber DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにSerialNumber未登録の場合は"" が設定される	0013A200404AC397
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにNodeIdentifier未登録の場合は"" が設定される	Device1
RSSI	フレームデータ中のRSSI(16進数2桁)	45
Options	フレームデータ中のOptions(16進数2桁)	00
RFData	フレームデータ中の受信したRFData (16進数、可変長)	0102030A0B

- **備考**

フレーム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、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、DeviceServer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

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.7 XBEE_TDCP_DATAイベント

- **イベント発生条件**

XBEE サービスモジュールで、API タイプ (0x80)または (0x81) のフレームを受信して、かつフレームデータ中

RFData 部の最初の 3 byte が 0x24, 0x24, 0x24 (ASCII で “$$$”) の場合に発生します。このときには 同時に XBEE_PAKCET_DATA イベントは発生しません。

“$$$” から始まるRFData は TDCP コマンドやイベント通知に使用されて、カンマ区切りでデータが入ります。

予め “サーバー設定” プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	81
SourceAddress	フレームデータ中のSourceAddress 16bit アドレスの場合(16進数4桁) 64bit アドレスの場合(16進数16桁)	 0A01 0013A200404AC397
SerialNumber	XBee デバイスの SerialNumber DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにSerialNumber未登録の場合は”” が設定される	0013A200404AC397
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServer のマスターファイルを使用して SourceAddress から変換した値が設定される マスターファイルにNodeIdentifier未登録の場合は”” が設定される	Device1
RSSI	フレームデータ中のRSSI(16進数2桁)	45
Options	フレームデータ中のOptions(16進数2桁)	00
TDCP_COUNT	TDCP データカラム数	2
TDCP_<Column#>	TDCP データ値(ASCII 文字列) <Column#>には 1 から TDCP_COUNT で指定された数までの値が入る。	”$$$1234”
TDCP_WHOLE	カンマで区切られたTDCP データ全体が入っています。	“$$$,SAMPLING,0A01,8,FF,0,760,759,759,51”

- **備考**

TDCP_<Column#> は、複数のデータが入り、TDCP_1, TDCP_2 ... のキー名が順に使用されます。

TDCP_1 には常に、TDCP コマンドプリフィックス文字列が入ります。”$$$” で始まり、0文字以上 5 文字以内の任意の文字列が後に続きます。(例 : ”$$$”, “$$$abc”, “$$$12345”)

TDCP_2 以降(TDCP_2, TDCP_3, TDCP_4 ...) には、TDCPコマンド・イベントで決められたデータが格納されます。

フレーム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、DeviceServer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。

XBEE_TDCP_DATA イベントは下記の詳細なイベント項目に分かれています。詳しいイベント毎のフォーマットやイベント発生条件については “TDCP ユーザーマニュアル” を参照して下さい。

イベント種別。”TDCP_2” データ項目の値	イベントの説明
CHANGE_DETECT	I/O ポートの値が変化した場合に発生します
RANGE_EXCEED	A/D 入力ポートの値が予め決められた制限値(上限または下限)を越えた場合に発生します。
COUNT_EXCEED	TDCP の ”app_mode” で入力ポート（カウント入力）に指定されたポートに対する入力値の変化回数が、予め決められた上限値を越えた場合に発生します。
SAMPLING	“sampling_rate” コマンドを使用して、自動サンプリング間隔(秒)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。
$GPRMC	シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。
GPS	シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。
LIVE	“heartbeat_rate” コマンドを使用して、LIVEイベント発生間隔(秒)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。

イベントハンドラ例

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.8 ZB_MODEM_STATUSイベント

- **イベント発生条件**

ZB サービスモジュール(XBee-ZB Series2 デバイス管理用のサービスモジュール) で、API タイプ (0x8A) のフレームを受信した時に発生します。予め “サーバー設定” プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
APIType	フレームデータ中のAPI Type(16進数2桁)	8A
ModemStatus	フレームデータ中のStatus(10進数)	6

- **備考**

フレーム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、DeviceServer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

```
File_id = "ZB_MODEM_STATUS"


if g_params["ModemStatus"] == "00" then log_msg("Hardware reset",file_id)
elseif g_params["ModemStatus"] == "01" then log_msg("Watchdog timer reset",file_id)
elseif g_params["ModemStatus"] == "02" then log_msg("Joined network",file_id)
elseif g_params["ModemStatus"] == "03" then log_msg("Disassociated",file_id)
elseif g_params["ModemStatus"] == "06" then log_msg("Coordinator started",file_id)
elseif g_params["ModemStatus"] == "07" then log_msg("Network security key was updated",file_id)
elseif g_params["ModemStatus"] == "0D" then log_msg("Voltage supply limit exceeded",file_id)
elseif g_params["ModemStatus"] == "11" then log_msg("Modem configuration changed while join in
progress",file_id)
elseif g_params["ModemStatus"] == "80" then log_msg("stack error",file_id)
else log_msg("Unknown status " .. g_params["ModemStatus"],file_id)
end
```

12.9 ZB_IO_DATAイベント

- **イベント発生条件**

ZB サービスモジュールで、API タイプ (0x92) の I/O データフレームを受信した時に発生します。

予め “サーバー設定” プログラムで、ZB 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
FrameType	フレームデータ中のAPI Type(16進数2桁)	92
SourceAddress	フレームデータ中のSourceAddress	0013A200404AC397

	64bit アドレスの場合(16進数16桁)	
NetworkAddress	フレームデータ中の SourceNetworkAddress 16bit アドレス(16進数4桁)	D565
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServerのマスターファイルを検索して設定される。マスターにNodeIdentifier未登録の場合は"" が設定される	Node1
DeviceType	XBee デバイスの Device Type DeviceServerのマスターファイルを検索して設定される。マスターにDeviceType未登録の場合は"" が設定される 8bit値(16進数2桁) 00: coordinator 01: router 02: end device	01
DeviceTypeID	XBee デバイスの Device Type Identifier DeviceServerのマスターファイルを検索して設定される。マスターにDeviceTypeID未登録の場合は"" が設定される 32bit値(16進数8桁)	00030000
ReceiveOptions	フレームデータ中 ReceiveOptions 8bit値(16進数2桁)	01
SAMPLE_COUNT	I/O データのサンプル数 現在のXBee-ZB のファームウエアでは常に 1 になります。	1
SAMPLE_DIO	I/O データ中のサンプル対象となったDIOビット番号リスト (10進数、カンマ区切り)	0,1,4
SAMPLE_ADC	I/O データ中のサンプル対象となったADCビット番号リスト (10進数、カンマ区切り)	2,3
SAMPLE_<Sample#>_<"DIO"|"ADC">_<Bit#>	I/O サンプルデータ値 DIO の場合は、High で "1"、Low で "0" ADC の場合は 10進数	 1 1023

- **備考**

サンプルデータを示すパラメータ名は、<Sample#> には 最大、SAMPLE_COUNT まで 1から順番にインクリメントされた値が入ります。<"DIO"|"ADC">は、I/O サンプルデータが ADC もしくは、DIO のどちらであるかを示します。<Bit#>は、サンプルデータのビット番号が入ります。

例：サンプル総数が 1 で DIO のサンプル対象ビットが 1, 2, 7, 10, 12 、ADC のサンプル対象ビットが 3 の場合には、イベントハンドラが実行時に下記のようなパラメータが設定されます。

```
g_params[DeviceType] = 02
g_params[DeviceTypeID] = ABCD0000
```

```
g_params[FrameType] = 92
g_params[NetworkAddress] = B198
g_params[NodeIdentifier] = Node1
g_params[ReceiveOptions] = 01
g_params[SAMPLE_1_ADC_3] = 523
g_params[SAMPLE_1_DIO_1] = 0
g_params[SAMPLE_1_DIO_10] = 1
g_params[SAMPLE_1_DIO_12] = 1
g_params[SAMPLE_1_DIO_2] = 0
g_params[SAMPLE_1_DIO_7] = 1
g_params[SAMPLE_ADC] = 3
g_params[SAMPLE_COUNT] = 1
g_params[SAMPLE_DIO] = 1,2,7,10,12
g_params[SourceAddress] = 0013A20040A0D533
```

フレーム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、DeviceServer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("start..",file_id)
for key,val in orderedPairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end
```

12.10 ZB_OTHERイベント

- **イベント発生条件**

ZB サービスモジュールで、API タイプ (0x91, 0x94, 0x95, 0xA1, 0xA2, 0xA3, 0xA4) を受信した時に発生します。予め “サーバー設定” プログラムで、ZB機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
FrameType	フレームデータ中のAPI Type(16進数2桁)	92
FRAME_WHOLE	フレームデータ全体の16進数の可変長データ	7E001891.....

- **備考**

フレーム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、DeviceServer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

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.11 ZB_PACKET_DATAイベント

- **イベント発生条件**

ZB サービスモジュールで、API タイプ（0x90）のフレームを受信した時に発生します。

予め“サーバー設定”プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行”を有効にしてください。

- **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
FrameType	フレームデータ中のAPI Type(16進数2桁)	92
SourceAddress	フレームデータ中のSourceAddress 64bit アドレスの場合(16進数16桁)	0013A200404AC397
NetworkAddress	フレームデータ中の SourceNetworkAddress 16bit アドレス(16進数4桁)	D565
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServerのマスターファイルを検索して設定される。マスターにNodeIdentifier未登録の場合は"" が設定される	Node1
DeviceType	XBee デバイスの Device Type DeviceServerのマスターファイルを検索して設定される。マスターにDeviceType未登録の場合は"" が設定される 8bit値(16進数2桁) 00: coordinator 01: router 02: end device	01
DeviceTypeID	XBee デバイスの Device Type Identifier DeviceServerのマスターファイルを検索して設定される。マスターにDeviceTypeID未登録の場合は"" が設定される 32bit値(16進数8桁)	00030000
ReceiveOptions	フレームデータ中 ReceiveOptions 8bit値(16進数2桁)	01

RFData	フレームデータ中の受信したRFData (16進数、可変長)	00112233

- **備考**

フレーム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、DeviceServer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

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.12 ZB_TDCP_DATAイベント

- **イベント発生条件**

ZB サービスモジュールで、API タイプ (0x90) のフレームを受信して、かつフレームデータ中 RFData 部の最初の 3 byte が 0x24, 0x24, 0x24 (ASCII で “$$$”) の場合に発生します。このときには 同時に ZB_PAKCET_DATA イベントは発生しません。

“$$$” から始まるRFData は TDCP コマンドやイベント通知に使用されて、カンマ区切りでデータが入ります。

予め “サーバー設定” プログラムで、XBEE 機能のセットアップと、”イベントパケット受信時にスクリプト実行” を有効にしてください。

- **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
FrameType	フレームデータ中のAPI Type(16進数2桁)	92
SourceAddress	フレームデータ中のSourceAddress 64bit アドレスの場合(16進数16桁)	0013A200404AC397
NetworkAddress	フレームデータ中の SourceNetworkAddress 16bit アドレス(16進数4桁)	D565
NodeIdentifier	XBee デバイスの NodeIdentifier DeviceServerのマスターファイルを検索して設定される。マスターにNodeIdentifier未登録の場合は"" が設定される	Node1
DeviceType	XBee デバイスの Device Type DeviceServerのマスターファイルを検索して設定される。マスターにDeviceType未登録の場合は"" が設定される 8bit値(16進数2桁)	01

	00: coordinator 01: router 02: end device	
DeviceTypeID	XBee デバイスの Device Type Identifier DeviceServerのマスターファイルを検索して設定される。マスターにDeviceTypeID未登録の場合は"" が設定される 32bit値(16進数8桁)	00030000
ReceiveOptions	フレームデータ中 ReceiveOptions 8bit値(16進数2桁)	01
TDCP_WHOLE	カンマ区切りのTDCP データ全体	$$$,CHANGE_DETECT,8,01,FE
TDCP_COUNT	TDCP データカラム数	5
TDCP_<Column#>	TDCP データ値(ASCII 文字列) TDCP_1 は常にコマンドプリフィックス文字列を表す "$$$" で始まり、0文字以上の任意の文字列が後に続く TDCP_2 はコマンド実行ステータスを表す "1" はコマンド実行成功、"0" は失敗を示す イベントデータの場合にはイベント名が入る TDCP_3以降のデータはTDCPコマンド毎に決められた、オプション文字列が入る <Column#> には 最大、TDCP_COUNT まで 1から順番にインクリメントされた値が入る。	"$$$1234" "1"

- **備考**

TDCP_<Column#> は、複数のデータが入り、TDCP_1, TDCP_2 ... のキー名が順に使用されます。

TDCP_1 には常に、TDCP コマンドプリフィックス文字列が入ります。"$$$" で始まり、0文字以上 5 文字以内の任意の文字列が後に続きます。(例："$$$", "$$$abc", "$$$12345")

TDCP_2 以降(TDCP_2, TDCP_3, TDCP_4 ...) には、TDCPコマンド・イベントで決められたデータが格納されます。

フレーム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、DeviceServer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。

XBEE_TDCP_DATA イベントは下記の詳細なイベント項目に分かれています。詳しいイベント毎のフォーマットやイベント発生条件については “TDCP ユーザーマニュアル” を参照して下さい。

イベント種別。”TDCP_2” データ項目の値	イベントの説明
CHANGE_DETECT	I/O ポートの値が変化した場合に発生します
RANGE_EXCEED	A/D 入力ポートの値が予め決められた制限値(上限または下限)を越えた場合に発生します。
COUNT_EXCEED	TDCP の ”app_mode” で入力ポート（カウント入力）に指定されたポートに対する入力値の変化回数が、予め決められた上限値を越えた場合に発生します。
SAMPLING	“sampling_rate” コマンドを使用して、自動サンプリング間隔(秒)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。
$GPRMC	シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。
GPS	シリアルコンソールに接続した GPS レシーバから、NMEA-0183 センテンスを受信した時に発生します。
LIVE	“heartbeat_rate” コマンドを使用して、LIVEイベント発生間隔(秒)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。

イベントハンドラ例

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.13 UIOUSB_EVENT_DATAイベント

- **イベント発生条件**

UIOUSB サービスモジュールで、UIOUSB デバイスからイベント文字列(最初が “$$$” で始まる、カンマ区切りの文字列)を受信したときに発生します。

予め “サーバー設定” プログラムで、UIOUSB 機能のセットアップをしてください。

- **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	イベントを送信したUIOUSB デバイスのポート名	COM3
EVENT_DATA_COUNT	イベントデータカラム数	2

EVENT_DATA_<Column#>	イベントデータ値(ASCII 文字列) <Column#>には 1 から EVENT_DATA_COUNT で指定された数までの値が入る。	"$$$" "FF"
EVENT_DATA_WHOLE	カンマで区切られたイベントデータ全体が入っています。	"$$$,SAMPLING,FF,0,903,766,511,254"

- **備考**

EVENT_DATA_<Column#> は、複数のデータが入り、EVENT_DATA_1, EVENT_DATA_2 ... のキー名が順に使用されます。

EVENT_DATA_1 には常に "$$$" が入ります。EVENT_DATA_2 以降(EVENT_DATA_2, EVENT_DATA_3, EVENT_DATA_4 ...) には、UIOUSB のイベント毎に決められたデータが格納されます。

イベン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、DeviceServer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。

UIOUSB_EVENT_DATA イベントは下記の詳細なイベント項目に分かれています。詳しいイベント毎のフォーマットやイベント発生条件については "UIOUSB ユーザーマニュアル" を参照して下さい。

イベント種別。"EVENT_DATA_2" データ項目の値	イベントの説明
CHANGE_DETECT	I/O ポートの値が変化した場合に発生します
COUNT_EXCEED	UIOUSB の カウンタ入力ポートに対する入力値の変化回数が、予め決められた上限値を越えた場合に発生します。
SAMPLING	"sampling_rate" コマンドを使用して、自動サンプリング間隔(100ミリ秒単位)を 0 以外に設定した場合に、その設定した時間間隔で繰り返し発生します。
ADVAL_UPDATE	A/D 変換値が ad_margin(ch#) で設定した値以上に変化した場合に発生します。

イベントハンドラ例

```
For key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.14 SERVER_STARTイベント

- **イベント発生条件**

DeviceServer 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"
--[[
*********************************************************************************
このスクリプトは DeviceServer 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。
*********************************************************************************
]]
log_msg("start..",file_id)
```

12.15 SERVER_STOPイベント

- **イベント発生条件**

DeviceServer 終了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**

このイベントはシステムのシャットダウン時などに、DeviceServer 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 DeviceServer 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、DeviceServer の各サービスモジュールの終了処理が行なわれます。

このイベントハンドラから別のスクリプトをコールすることも出来ますが、この場合には必ず同一スレッドで実行して、script_fork_xxxx() ではじまる別スレッドで実行する関数は使用しないでください。

- **イベントハンドラ例**

```
file_id = "SERVER_STOP"
--[[
```

```
*****************************************************************************
このスクリプトは DeviceServer 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、DeviceServer 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)
```

12.16 SERIAL_STRINGイベント

- **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。
予め “サーバー設定” プログラムで、SERIAL デバイスのデバイスタイプを “STRING” にして下さい。
またバッファー入力モードは 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	“COM1”
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、DeviceServer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);
```

12.17 SERIAL_FIRMATAイベント

- **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。

予め “サーバー設定” プログラムで、SERIAL デバイスのデバイスタイプを “FIRMATA” にして下さい。

またバッファー入力モードは 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	“COM1”
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、DeviceServer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
```

12.18 SERIAL_RAWイベント

- **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。

予め “サーバー設定” プログラムで、SERIAL デバイスのデバイスタイプを “RAW” にして下さい。

またバッファー入力モードは 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	“COM1”
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、DeviceServer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);
```

12.19 SERIAL_TWEイベント

- **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TWEワイヤレスデバイスのアスキー形式データパケットを受信した時に発生します。
予め“サーバー設定”プログラムで、SERIAL デバイスのデバイスタイプを “TWE” にして下さい。
またバッファー入力モードは 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	“COM1”
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”
TWE_DATA	COM ポートから入力されたアスキー形式のデータパケット全体を格納しています。 データパケット､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] TWE_DATA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	":01890902010A01020 3040405060708092F"

- **TWE_DATAで渡されたデータパケットを解析して作成されるテーブルと文字列変数**

下記のテーブルと文字列変数は、SERIAL_TWE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TWEデータパケットの内容を、イベントハンドラやイベントハンドラからコールされるスクリプト中で簡単に扱うことができます。

TWEワイヤレスデバイスに独自のファームウエアを作成した場合でも、下記に該当するアスキー形式の場合には自動的に解析されて同様のテーブルや文字列データに変換されます。

独自のアスキー形式を作成した場合には、SRIAL_TWEイベントハンドラ中にそれらの解析部分を記述するようにしてください。

テーブルまたは文字列変数名	説明	値の例
byte_arr[]	TWE_DATA が ":" 1文字から始まっている場合に、16進数文字列部分を１バイト毎に数値に変換して配列に格納したものが入る TWE_DATA 例 ":01890902010A01020304040506070809 2F"	byte_arr[1] = 0x01 byte_arr[2] = 0x89 byte_arr[3] = 0x09 ..
key_val[]	TWE_DATA が "::" 2文字から始まっている場合に、続く ":" 文字毎にカラムを分けて "<key>=<val>" で記述された部分を連想配列に格納したものが入る。 TWE_DATA 例 "::rc=80000000:lq=84:ct=0001:ed=81002A1E:id=1:ba=3320:a1=0009:a2=0009:p0=001:p1=000"	key_val["rc"] = "80000000" key_val["lq"] = "84" key_val["ct"] = "0001" ..
tag_arr[]	TWE_DATA が ";" 1文字から始まっている場合に、続く ";" 文字毎にカラムを分けたものを文字列形式で配列に格納したものが入る。 TWE_DATA 例 ";116;00000000;054;001;1002ecd;3330;0007;0042;0007;0007;S;"	tag_arr[1] = "116" tag_arr[2] = "00000000" tag_arr[3] = "054" ..
comment	TWE_DATA が ":"または ";" 文字以外から始まっている場合に、データパケット全体の文字列を格納したものが入る TWE_DATA 例 "!INF TOCOS TWELITE DIP APP V1-00-2, SID=0x81000038, LID=0x78"	"!INF TOCOS TWELITE DIP APP V1-00-2, SID=0x81000038, LID=0x78"

- **TWE_DATAを解析してこのスクリプト中で作成されるグローバル共有変数と共有文字列リスト**

グローバル共有変数と共有文字列リストは、DeviceServerのスクリプトや WebAPI 等からTWEワイヤレスデバイス子機の最新のセンサ値を簡単に利用できるように作成しています。

グローバル共有変数名または、共有文字列リスト Channel名	説明	値の例
TWE_<COMPort>_<ChildID>	TWE_DATA が ":" 1文字から始まっていて、かつコマンド種別を示すバイト値が 0x81(状態通知)の場合に、メッセージ内容を解析した値がカンマ	グローバル共有変数名 例：

	区切りでグローバル共有変数に格納されます。この変数の値は常に最後のイベント発生時の内容で更新されます。 <COMPort>は、イベントを送信したシリアルデバイスの COMポート名(親機が接続されているポート)になります。<ChildID>は、メッセージ中の送信元論理デバイスIDを10進数にしたものになります。 変数の内容に設定されるカンマ区切りの文字列は下記のフォーマットになります。 <LQI>,<Batt>,<DI1>,<DI2>,<DI3>,<DI4>,<AD1>,<AD2>,<AD3>,<AD4> <LQI> にはLQI値フィールドの値を10進数に変換したものが入ります <Batt> には電源電圧[mV]フィールドの値を10進数に変換したものが入ります <DI1>..<DI4> にはDI の状態ビットが Lowの場合に1, High の場合に 0 が入ります <AD1>..<AD4> にはAD変換値の値を10進数に変換したものが入ります	"TWE_COM10_1" 変数の値例： "63,3286,0,0,0,1,8,8,8,8"
TWE_<COMPort>_CHILD_LIST	TWE ワイヤレスデバイスの子機のID 値を共有変数文字列リストに格納します。<COMPort>は、イベントを送信したシリアルデバイスの COMポート名(親機が接続されているポート)になります。 TWE_DATA が ":" 1文字から始まっているパケットデータを受信したときの <ChildID> 部分を文字列リストに保存します。文字列リストにはメッセージ中の送信元論理デバイスIDを10進数表現にしたものを重複なく保存しています。	[1] = "1" [2] = "2" ..

- **備考**

シリアルポートからアスキー形式のデータパケットを受信する毎に、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、DeviceServer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

```
file_id = "SERIAL_TWE"
local str = ""
for key,val in pairs(g_params) do
  str = str .. key .. " = " .. val .. " "
end
log_msg(str,file_id)
-------------------------------------------------------------
```

```
-- g_params["TWE_DATA"] データパケット文字列をデコードする
------------------------------------------------------------
local data = g_params["TWE_DATA"]
local byte_arr,key_val,tag_arr,comment
if string.match(data,"^::") then
  key_val = key_val_to_tbl(string.sub(data,3,-1))
elseif string.match(data,"^:%x") then
  byte_arr = hex_to_tbl(string.sub(data,2,-1))
  local id = tostring(byte_arr[1])
  if not add_shared_strlist("TWE_" .. g_params["COMPort"] .. "_CHILD_LIST",id,true) then error() end --
子機のIDリストを更新
  if byte_arr[2] == 0x81 then -- 状態通知の場合には共有変数に現在のセンサ値を保存
    local di1,di2,di3,di4,ad1,ad2,ad3,ad4
    if bit_and(byte_arr[17],0x01) ~= 0 then di1 = "1" else di1 = "0" end
    if bit_and(byte_arr[17],0x02) ~= 0 then di2 = "1" else di2 = "0" end
    if bit_and(byte_arr[17],0x04) ~= 0 then di3 = "1" else di3 = "0" end
    if bit_and(byte_arr[17],0x08) ~= 0 then di4 = "1" else di4 = "0" end
    if byte_arr[19] == 0xFF then ad1 = "-1" else ad1 = tostring((byte_arr[19]*4 +
bit_and(byte_arr[23],0x03))*4) end
    if byte_arr[20] == 0xFF then ad2 = "-1" else ad2 = tostring((byte_arr[20]*4 +
bit_and(bit_rshift(byte_arr[23],2),0x03))*4) end
    if byte_arr[21] == 0xFF then ad3 = "-1" else ad3 = tostring((byte_arr[21]*4 +
bit_and(bit_rshift(byte_arr[23],4),0x03))*4) end
    if byte_arr[22] == 0xFF then ad4 = "-1" else ad4 = tostring((byte_arr[22]*4 +
bit_and(bit_rshift(byte_arr[23],6),0x03))*4) end
    local val = tostring(byte_arr[5]) .. "," .. tostring(byte_arr[14]*256 + byte_arr[15])
        .. "," .. di1 .. "," .. di2 .. "," .. di3 .. "," .. di4
        .. "," .. ad1 .. "," .. ad2 .. "," .. ad3 .. "," .. ad4
    if not set_shared_data("TWE_" .. g_params["COMPort"] .. "_" .. id,val) then error() end
    -- log_msg(val,file_id)
  end
elseif string.match(data,"^;") then
  tag_arr = ssv_to_tbl(string.sub(data,2,-2))
else
  comment = data
end
------------------------------------
-- デコード後のデータをログに出力
------------------------------------
```

```
local line = ""
if key_val then
  for key,val in pairs(key_val) do
    line = line .. "[" .. key .."]" .. val .. " "
  end
end
if byte_arr then
  for key,val in ipairs(byte_arr) do
    line = line .. "0x" .. bit_tohex(val,2) .. " "
  end
end
if tag_arr then
  for key,val in ipairs(tag_arr) do
    line = line .. "[" .. tostring(key) .. "]" .. val .. " "
  end
end
if comment then
  line = comment
end
log_msg("decoded: " .. line,file_id)
```

12.20 DEVICE_MESSAGEイベント

- **イベント発生条件**

DeviceServer にネットワークで接続されたデバイスから CSVIF プロトコルで “DEVICE_MESSAGE” パケットを受信した時に発生します。CSVIF プロトコルの詳細は、”DeviceServer CSVIF インターフェイスマニュアル” を参照してください。イベントハンドラに渡されるパラメータは、デバイスから指定されたパラメータの他に、デバイスの IP アドレスが常に指定されます。

- **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
DeviceIPAddress	デバイスの IP アドレス	192.168.100.30
<任意のキー#1>	デバイスから指定された任意のキーと値(オプション)	<任意の値#1>
..	..	..
<任意のキー#n>	..	..

- **備考**

ネットワーク接続されたデバイスから、DeviceServer 側に対してメッセージを送信するために使用します。このイベントハンドラ内で設定したリターンパラメータを、CSVIF プロトコルでリクエスト元のデバイスに渡すことができます。

- **イベントハンドラ例**

```
file_id = "DEVICE_MESSAGE"
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
```

12.21 GLOBAL_WATCHイベント

- **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。

グローバル共有変数を監視対象にするには、DeviceServerの共有データ文字列リストのチャンネル名 $GLOBAL_WATCH にグローバル共有変数の名前を追加します。文字列リストに監視対象となるグローバル共有変数の名前を指定して追加します。

監視対象から外す場合には、DeviceServerの共有データ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	DeviceStaus
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 等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を実行します。監視対象となっているグローバル共有変数を更新したAPI 関数は、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。ダイナミックに監視対象を変更する場合には、DeviceServerのデータベースにも監視対象の変数名リストを保存して、サーバー起動時にデータベースから共有データ文字列リスト $GLOBAL_WATCHにロードするようなスクリプトをSERVER_START イベントハンドラに記述して実現できます。

⚠ 注意 GLOBAL_WATCHイベントハンドラから監視対象となったグローバル共有変数を変更しないこと

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
```

12.22 MQTT_KEEP_ALIVE_TIMERイベント

- **イベント発生条件**

“サーバー設定”プログラムで 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。

(*1) “サーバー設定”プログラムの MQTT サービスモジュールタブ中で設定した全てのエンドポイントについて以下の (*2),(*3) の処理を行う

(*2) mqtt_open() ライブラリ関数を使用して MQTT ブローカーに接続中で、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。

(*2) MQTT ブローカーに接続中であるが、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をクローズした後、MQTTブローカーに再接続を試みます。

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
```

12.23 MQTT_PUBLISHイベント

- **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。

予め “サーバー設定” プログラムで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″ 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。	″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"

- **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 2048 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"	PubslihString = "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DeviceServer 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MQTTT_PUBLISH イベントについては、必ず先行する MQTTT_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 DeviceServer 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、DeviceServer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

```
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 2048 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 2048 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 < 2048 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)
```

12.24 ALARM_BUTTON_PUSHイベント

- **イベント発生条件**

SigSensor, NetUIO デバイスでプッシュボタンが押された時に発生します。同時にデバイス上の複数のボタンを押された場合には、イベントは一回だけ発生します。この時は、イベントハンドラに渡されるパラメータを調べることで、同時に押されていたボタンの情報を取得できます。

- **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
DeviceType	アラームデバイスタイプ。SIGSENSOR, NETUIO のいずれか	SIGSENSOR
IPAddress	デバイスのIPアドレス	192.168.100.30
PortNumber	デバイスのポート番号	27102

DeviceName	アラームデバイス名	SigSensor
SW_A	現在の BUTTON A の状態	ON
SW_B	現在の BUTTON B の状態	OFF
SW_C	現在の BUTTON C の状態	ON
SW_D	現在の BUTTON D の状態 (NetUIOデバイスのみ)	OFF

- **備考**

ボタンリリース時と、SigSensor デバイスのリセットボタンを押した場合にはイベントは発生しません。

DeviceServer が他のデバイスと通信している間、デバイス管理の排他制御の為一時的にイベントの発生が遅延することがあります。他のデバイスでネットワークエラーが発生している時に、エラーを検出するまでのタイムアウト時間のために、数秒..30秒程度までイベントの受信が遅延する可能性があります。

デバイスの割り当てポート番号や検出条件などの詳しい内容は “SigSensor_NetUIOユーザーマニュアル” の”機能” の章を参照してください。

イベントを送信したデバイスが、送信先の DeviceServer に登録されていない場合は、DeviceServer のログにエラーが記録されるだけで、イベントハンドラスクリプトは実行されません。登録されている場合でも、DeviceServer の対応するアラームデバイスに、ネットワークエラーが発生したフラグが立っている場合も、イベントハンドラスクリプトは実行されずにエラーが記録されます。ネットワークエラーをリセットするには、デバイス管理プログラムからエラーリセットボタンを押すか、スクリプトライブラリ関数 alarm_network_reset() もしくは API関数 SX_alarm_network_reset()を実行してください。

- **イベントハンドラ例**

```
------------------------------------------------------------
-- 押されたボタンの名前とデバイス名をログに表示する
------------------------------------------------------------
push_list = {}
for key,val in pairs(g_params) do
  if string.match(key,"^SW_") and (val == "ON") then
      table.insert(push_list,key)
   end
end
log_msg(g_params["DeviceName"] .. " " .. table.concat(push_list,","))
```

12.25 ALARM_DINPUT_CHANGEイベント

- **イベント発生条件**

SigSensor, NetUIO デバイスのDINPUT 値が変化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を調べることで、変化した時の DINPUT の各ポート値を取得できます。

- **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
DeviceType	アラームデバイスタイプ。SIGSENSOR, NETUIO のいずれか	NETUIO
IPAddress	デバイスのIPアドレス	192.168.100.30
PortNumber	デバイスのポート番号	27102
DeviceName	アラームデバイス名	uio#1
DIN_1	現在の DIGITAL INPUT 1 の状態	1
DIN_2	現在の DIGITAL INPUT 2 の状態	0
DIN_3	現在の DIGITAL INPUT 3 の状態	1
DIN_4	現在の DIGITAL INPUT 4 の状態	0

- **備考**

デバイスの割り当てポート番号や検出条件などの詳しい内容は “SigSensor_NetUIOユーザーマニュアル” の”機能” の章を参照してください。

DeviceServer が他のデバイスと通信している間、デバイス管理の排他制御の為一時的にイベントの発生が遅延することがあります。他のデバイスでネットワークエラーが発生している時に、エラーを検出するまでのタイムアウト時間のために、数秒..30秒程度までイベントの受信が遅延する可能性があります。

イベントを送信したデバイスが、送信先の DeviceServer に登録されていない場合は、DeviceServer のログにエラーが記録されるだけで、イベントハンドラスクリプトは実行されません。登録されている場合でも、DeviceServer の対応するアラームデバイスに、ネットワークエラーが発生したフラグが立っている場合も、イベントハンドラスクリプトは実行されずにエラーが記録されます。ネットワークエラーをリセットするには、デバイス管理プログラムからエラーリセットボタンを押すか、スクリプトライブラリ関数 alarm_network_reset()もしくは API関数 SX_alarm_network_reset()を実行してください。

- **イベントハンドラ例**

```
------------------------------------------------------------
-- 変化した時のDINPUT の値を 16進数表記でログに出力
------------------------------------------------------------
din = 0
for key,val in pairs(g_params) do
  bit_str,cnt = string.gsub(key,"^DIN_(%d+)","%1")
  if (cnt > 0) and (val == "1") then
    din = din + math.pow(2,tonumber(bit_str) - 1)
  end
end
log_msg(string.format("dinput = %2.2x",din))
```

12.26 ALARM_ADRANGE_EXCEEDイベント

- **イベント発生条件**

SigSensor, NetUIO デバイスのA/D変換値が予め設定した上限、下限値を超えたときに発生します。イベントハンドラに渡されるパラメータを調べることで、設定した制限値を超えた A/D channel と上限・下限の区別を取得できます。

- **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
DeviceType	アラームデバイスタイプ。SIGSENSOR, NETUIO のいずれか	NETUIO
IPAddress	デバイスのIPアドレス	192.168.100.30
PortNumber	デバイスのポート番号	27102
DeviceName	アラームデバイス名	uio#1
CHANNEL	A/D チャネル番号(“ADn” n=1, 2, 3, 4)	AD1
LOW_HIGH	上限または下限のどちらの範囲を超えたか	LOW

- **備考**

ALARM_ADRANGE_EXCEED は 各A/D channel の値が、上限値または下限値を超えたときに、それぞれ一回だけイベントが発生します。一回だけ発生するのは、イベント発生と同時に有効フラグをリセットすることで、同一イベントが連続して発生しないようにしているためです。再びイベント送信可能な状態にするためには、アプリケーションサーバーのアラーム管理プログラムから、レンジチェック有効フラグをセットしてください。デバイスのリセット直後は、全てのレンジチェック有効フラグはリセットされていますので、イベントを送信する状態にするために、レンジチェック有効フラグをセットしてください。スクリプトの中で、レンジチェック有効フラグを操作することもできます。ALARMDEVICE_ADRAGE_FLAG.luaファイルを参照してください。

DeviceServer が他のデバイスと通信している間、デバイス管理の排他制御の為一時的にイベントの発生が遅延することがあります。他のデバイスでネットワークエラーが発生している時に、エラーを検出するまでのタイムアウト時間のために、数秒..30秒程度までイベントの受信が遅延する可能性があります。

デバイス上のA/D 変換機能についての詳しい内容は “SigSensor_NetUIOユーザーマニュアル” の”機能” の章を参照してください。

イベントを送信したデバイスが、送信先の DeviceServer に登録されていない場合は、DeviceServer のログにエラーが記録されるだけで、イベントハンドラスクリプトは実行されません。登録されている場合でも、DeviceServer の対応するアラームデバイスに、ネットワークエラーが発生したフラグが立っている場合も、イベントハンドラスクリプトは実行されずにエラーが記録されます。ネットワークエラーをリセットするには、デバイス管理プログラムからエラーリセットボタンを押すか、スクリプトライブラリ関数 alarm_network_reset()もしくは API関数 SX_alarm_network_reset()を実行してください。

- **イベントハンドラ例**

```
--
-- レンジチェックで範囲を超えたデバイスとチャンネル、上限下限の情報をログに表示
--
for key,val in pairs(g_params) do
   log_msg(string.format("g_params[%s] = %s",key,val))
end
```

12.27 ALARM_NETWORK_ERRORイベント

- **イベント発生条件**

DeviceServer がアラームデバイス(SigSensor, NetUIO, AlarmSignal)へ通信を行った時に、エラーが発生してデバイスから正常な応答が得られなかった時に発生します。

- **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
DeviceName	エラーを検出したアラームデバイス名	uio#1

- **備考**

一度エラーが発生したデバイスについて同様のエラーが次に発生しても、２回目以降はこのイベントは発生しません。エラーが発生したデバイスで、再びエラー発生時にこのイベントを発生させるようにするためには、スクリプトライブラリ関数や API を使用してネットワークエラーをリセットしてください。ALARMDEVICE_NETWORK_RESET.luaファイルを参照してください。

ALARM_NETWORK_ERROR イベントハンドラのスクリプトで、アラームデバイスの操作を行う等の処理を行う時に、再び通信エラーが発生（別のデバイスで）した場合は、ALARM_NETWORK_ERROR イベントハンドラが別スレッドで平行して実行します。

アラームデバイスから DeviceServer への通信（デバイスで発生したイベント通知）を行う時にエラーが発生した場合には、 ALARM_NETWORK_ERROR イベントは発生しません。この時に、SigSensorデバイスの場合は、LCD にエラー表示と Beep1 ブザー、赤ランプ点滅を行って、デバイス自身がエラー発生を知らせます。

- **イベントハンドラ例**

```
--------------------------------------------------------------------------
-- システムアラートとしてフラグが設定されたデバイスに,エラーが発生したことを
-- Red ランプと Beep1 ブザー, LCD メッセージで知らせる
-- エラー発生を通知する対象のデバイスで再びエラーが発生した場合には
-- LCD の内容は最後に検出したエラーの内容で上書きされる
--------------------------------------------------------------------------
local stat,name,type = alarm_sysalert_list()
```

```
if not stat then error() end
for key,val in ipairs(name) do
  if (type[key] == "SIGSENSOR") or (type[key] == "ALARMSIGNAL") then
    if (type[key] == "SIGSENSOR") then
     stat = alarm_signal_message(val,"**DEVICE ERROR**" .. g_params["DeviceName"])
      if not stat then error() end
    end
   stat = alarm_signal_set(val,"BlinkingRed",true)
   if not stat then error() end
   stat = alarm_signal_set(val,"Beep1",true)
   if not stat then error() end
  end
end
```

13 システム関連 API (Lua)

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 DeviceServer で定義されたライブラリ関数とグローバル変数です。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” の項目で関数名とパラメータ、リターン値とその Lua 型名が説明します。

DeviceServer で使用している Lua のバージョンは “5.1.4” です。
このマニュアルで記載した DeviceServer 専用の API 以外にも、Lua5.1 で定義された ”Standard Library” 関数が使用できます。ただし、このときは、下記の使用上の制限事項があります。

制限事項
DeviceServer 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Windows OS のログイン状態を想定した関数や変数は使用できません。これは DeviceServer が Windows サービスプログラムで動作しているためです。この場合は、DeviceServer で新たに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(通常は stat) が入る様に定義されている時には、実行時にエラーを検出した場合には false が返り、実行が成功した場合には true が返ります。ただし、スクリプト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、そのスクリプトやイベントハンドラの実行自身がエラーで中断されます。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、ライブラリ関数呼び出し部分で処理が中止されますので注意してください。

13.1 g_startup:String

- **機能概要**

  DeviceServer(ABAppService)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))
```

13.2 g_hostname:String

- **機能概要**

  DeviceServer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))
```

13.3 g_params:Table

- **機能概要**

  スクリプト実行時に渡されたパラメータ。パラメータ名と値がペアで格納される。

- **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、その内容が "val1" に設定されています。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。

- **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
```

13.4 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**

  script_result() を使用するときの taskid パラメータに使用します。

- **使用例**

```
if not script_result(g_taskid,"key1","val1") then error() end
```

13.5 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 DeviceServer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、クライアントのホスト名が入ります。

  WebProxy 経由でブラウザから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 DeviceServer 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))
```

13.6 g_json:Table

- **機能概要**

  json.decode() 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() 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() 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)
```

13.7 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**

- **使用例**

```
log_msg(string.format("起動スクリプト名 %s ",g_script))
```

13.8 log_msg()

- **機能概要**

  メッセージをログサーバーに出力する。

- **関数定義**

  log_msg(message [,modulename])

- **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)

  modulename:String　　モジュール名(任意の文字列) 省略時は "LUA Script"が指定される。

- **備考**

  DeviceServer の Lua スクリプトエンジンでは、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。

- **使用例**

```
log_msg("Hello World","TestModule")
```

13.9 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
```

13.10 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_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。

  event_name:String　　対象となるイベントオブジェクトの名前を指定する

- **備考**

  Windows OS の Win32 API CreateEvent(), SetEvent() を使用していますので、他のアプリケーションと同一のイベントを指定して連係動作させることが可能です。

- **使用例**

```
If not event_set(“FLAG1”) then error() end
```

```
-----------------------------------------------------------------
-- 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)
-----------------------------------------------------------------

--[[
この関数を利用する場合には、SERIAL_STRING イベントハンドラ内に下記の様な、
イベント送信と文字列を共有メモリに格納するルーチンを追加する

local wait_port = "COM1"
if g_params["COMPort"] == wait_port then
        -- set result data
        if not set_shared_data("INPUT_DATA_" .. wait_port,g_params["STRING"]) then error() end
        -- event set for input()
        if not event_set("INPUT_EVENT_" .. wait_port) then error() end
```

```
end;
]]

function serial_input(com_port)
  -------------------------------------
  -- waiting for serial data
  -------------------------------------
  if not event_wait("INPUT_EVENT_" .. com_port,5000) then
    log_msg("serial_input:*ERROR* timeout or error",file_id)
    error();
  end
  -------------------------------------
  -- get a string from the buffer
  -------------------------------------
  local stat,val = get_shared_data("INPUT_DATA_" .. com_port)
  if stat then
    return val
  else
    error()
  end
end
```

13.11 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_name, 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_name:String	対象となるイベントオブジェクトの名前を指定する
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)

- **備考**

  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。

  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。

Windows OS の Win32 API CreateEvent(), ResetEvent(), WaitForSingleObject() を使用していますので、他のアプリケーションと同一のイベントを指定して連係動作させることが可能です。

- **使用例**

```
If not event_wait("FLAG1",100000) then
    -- wait timeout --
end
```

event_set() 関数の使用例も参照してください

13.12 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(flag_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。

flag_name:String　　対象となるイベントフラグの名前を指定する

- **備考**

event_wait2() 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 event_set() ライブラリ関数とグローバル共有データを使用しています。

- **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)
```

13.13 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(flag_name1[,... flag_name<n>][,timeout])

  stat = event_wait_either2(flag_name1[,... flag_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。
flag_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は、event_wait2() の場合に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() の場合には、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**

イベントフラグを複数指定してセット状態になるのを待ちます。１つのイベントのみしか扱わない場合には、event_wait()と event_set()ライブラリ関数で代用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内部で event_wait() ライブラリ関数とグローバル共有データを使用しています。

- **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。

13.14 critical_section_enter()

- **機能概要**

パラメータで指定した名前の MUTEX を使用した排他制御区間に入る

- **関数定義**

stat, handle = critical_section_enter(mutex_name, 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。
handle:Number	critical_section_leave() コールで使用するハンドルが返る。
mutex_name:String	対象となるMUTEX オブジェクトの名前を指定する
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)

- **備考**

指定した timeout 時間内に MUTEX 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_enter()の stat 値が true の場合は、DeviceServer が MUTEXリソースを確保した状態になっています。必ず同一スクリプト内で critical_section_leave() を使用してリソースの開放を行ってください。 stat 値が false の場合(タイムアウト等) の場合は critical_section_leave() をコールする必要はありません。

コールが成功した場合に返される handle は、排他制御区間から抜けるときにコールする、
critical_section_leave() のパラメータに指定してください。

critical_section_enter() をコールしたスクリプトの実行が終了する前に、critical_section_leave()をコールしていない場合には、DeviceServer は強制的に使用中のMUTEX を開放した後 handle をクローズします。このときログには、 “ExecuteScript:*WARNING* removed g_mutex_handles[xxx]” の様に記録されます。

Windows OS の Win32 API OpenMutex()又はCreateMutex(), WaitForSingleObject() を使用していますので、他のアプリケーションと同一の MUTEXを指定して連係動作させることが可能です。

別スレッドに跨って排他制御区間を設けたい場合には、critical_section_enter2(),
critical_section_leave2() 関数を使用してください。これらの関数はグローバル共有変数を使用して名前つきの排他制御区間を作成することができます。

- **使用例**

```
local stat,handle = critical_section_enter(“FLAG1”,100000)
if stat then
    ----------------------
    -- 排他制御が必要な部分
    ----------------------
    critical_section_leave(handle)
end
```

```
file_id = "ARDUINO_IO_GET"
--[[
*******************************************************************************
Arduino デバイスのデジタルポート値を取得する
Arduino では FIRMATA プロトコル [DIGITAL_PORTS_QUERY] [DIGITAL_PORTS_REPLY] をサポートした
```

```
SensorControlModule.ino スケッチプログラムを動作させてください。
スクリプト起動時に渡されるパラメータ
-------------------------------------------------------------------------------
キー値                  値                                      値の例
-------------------------------------------------------------------------------
com             Arduino が接続された COM ポート名               "COM4"

リターン時に返されるパラメータ
-------------------------------------------------------------------------------
キー値                  値                                      値の例
-------------------------------------------------------------------------------
PORT_0          Arduino port#0(8bit)の現在の値(16進数2桁)       "FC"
PORT_1          Arduino port#1(8bit)の現在の値(16進数2桁)       "00"
..
PORT_n          Arduino port#n(8bit)の現在の値(16進数2桁)       "00"

ATmega328 を使用した Arduino ボードの場合には、 digital I/O port は 0,1 の2 つで
それぞれ、port0:PORTD, port1:PORTB(bit#0-bit#5) + (PORTC bit#0-#1)に対応しています。
詳しくは、Arduino IDE に同梱されている pins_arduino.h ファイルを参照して下さい。
*******************************************************************************
]]
-------------------------
-- パラメータチェック
-------------------------
if not (g_params["com"]) then
        log_msg("parameter error",file_id)
        error()
end
-----------------------------------------------------------------------
-- Arduino digital I/O ポート取得 QUERY 送信と REPLY 受信
-----------------------------------------------------------------------
local com = g_params["com"]
local sysex_start = "F0"
local sysex_end = "F7"
local firmata_request = "61"
local firmata_reply = "62"
local send_data = sysex_start .. firmata_request .. sysex_end
-----------------------------------------------------------------------
-- 排他制御開始
```

```
-- 同じ Arduino デバイスに対して、QUERY コマンドが複数同時実行される場合に備えています。
-- 別のArduino デバイスを操作する場合や、同じ Arduino デバイスに対して異なる FIRMATA REPLY
-- コマンドバイト値を取得する場合には排他制御の必要はありません
--------------------------------------------------------------------------
local cstat,handle = critical_section_enter("Firmata" .. firmata_request,5000);
if not cstat then
        log_msg("critical_section_enter() failed",file_id)
        error()
end
--------------------------------------------------------------------------
-- リクエストフレーム送信とリプライ受信
--------------------------------------------------------------------------
local stat,rdata = serial_write(com,send_data,firmata_reply)
--------------------------------------------------------------------------
-- 排他制御終了
--------------------------------------------------------------------------
cstat = critical_section_leave(handle)
if not cstat then
        log_msg("critical_section_leave() failed",file_id)
        error()
end
if not stat then
        log_msg("serial_write() failed",file_id)
        error()
end
--------------------------------------------------------------------------
-- 受信した FIRMATA フレームの内容をリターンパラメータに格納
--------------------------------------------------------------------------
local data = hex72_to_tbl(string.sub(rdata,5,-3),2)
if data ~= nil then
  for key,val in ipairs(data) do
    if not script_result(g_taskid,"PORT_" .. tostring(key - 1),bit_tohex(val,2)) then error() end
  end
end
```

13.15 critical_section_leave()

- **機能概要**

  パラメータで指定した MUTEX ハンドルを使用した排他制御区間から抜ける

- **関数定義**

stat = critical_section_leave(handle)

- **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。
handle:Number	critical_section_enter() コールで取得したハンドルを指定する。

- **備考**

critical_section_enter() 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_leave() 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。また critical_section_leave() では使用していた MUTEX リソースの開放も同時に行います。

Windows OS の Win32 API ReleaseMutex(), CloseHandle() を使用していますので、他のアプリケーションと同一のイベントを指定して連係動作させることが可能です。

- **使用例**

```
local stat,handle = critical_section_enter("FLAG1",100000)
if stat then
    ------------------------
    -- 排他制御が必要な部分
    ------------------------
    critical_section_leave(handle)
end
```

critical_section_enter() 関数の使用例も参照してください。

13.16 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。1000 を指定すると最大 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。critical_section_leave2() 関数にcritical_section_enter2()コール時に指定した同じ key_name を指定して排他制御権を開放してください。 stat 値が false の場合(リトライ回数エラー等) の場合は 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
```

13.17 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(key_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。
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 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。

13.18 exclusive_check()

- **機能概要**

  コールしたスクリプトがサーバー上の別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。

  scruipt_name:String　　自身のスクリプト名。"g_script" を指定する

- **備考**

  サーバー上で２重起動したくないスクリプトを作成する場合に、先頭にこの関数を使用して判定することができます。

  この関数をコールする場合には "script_name" パラメータにはグローバル変数 "g_script" を指定します。また、この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) 様にします。

  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ます。無限ループになるようなスクリプトを強制的に終了したとき("script_kill()" ライブラリ関数や "TaskList" タスク管理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します。

- **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
```

13.19 create_session()

- **機能概要**

  DeviceServerに新規セッションを登録する。

  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アカウント情報が含まれない。

- **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DeviceServer側で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。 省略時は DeviceServer の設定で決められた自動ログアウト時間によってセッションが自動削除される。

- **備考**

  ライセンスで指定された最大同時ログイン数を超えるセッションを作成することはできません。

- **使用例**

```
stat,token = create_session(“PUBLIC_SESSION”,true)
if not stat then error() end
```

13.20 delete_session()

- **機能概要**

  DeviceServer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() で作成されたセッションを削除する。このとき、セッションが保持していた全てのセッション共有データは削除されます。

ログイン認証してから作成されたセッションをログアウトしないで、この関数で削除することも可能ですが、その場合にはユーザーアカウント情報の最終ログアウト日時は更新されません。ログイン認証したセッションをログアウトしてからセッションを削除する場合には、XASDLCMD.DLLからアクセス可能な、SX_LogoutUser() API 関数または、WebAPI の /command/json/session_logout を使用して下さい。

- **使用例**

```
stat = delete_session("PUBLIC_SESSION")
if not stat then error() end
```

13.21 renew_session()

- **機能概要**

DeviceServer 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。

- **使用例**

```
stat,current_token = renew_session(current_token,60)
if not stat then error() end
```

13.22 service_module_status()

- **機能概要**

DeviceServer のサービスモジュールが ONLINE 状態かどうかを調べる。

- **関数定義**

stat, module_status = 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　DeviceServer 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。

- **備考**

  module_status のキー名は、DeviceServer のサービスモジュール名（下記）になります。

- **DeviceServer サービスモジュール一覧**

サービスモジュール名	説明
COUNTER	DeviceServer 内部のシステムカウンタ、統計データベース、時系列文字列データベース、Permanent データベースの機能を提供しています。
CONVERT	郵便番号等、内部で使用する変換テーブルの機能を提供しています。
CONFIG	内部設定値の管理機能を提供しています。
MASTERS	マスターテーブルの管理機能を提供しています。
SESSION	ユーザーセッションの管理と、グローバル共有データ、セッション共有データの機能を提供しています。
ORACLE	Oracle RDBMS 上の統計データベース、Permanent データベースの機能を提供しています。
USER	ユーザー情報管理機能を提供しています。
ALARM	アラームデバイスの管理機能を提供しています。
CSVIF	アラームデバイスや他のネットワークデバイスとの TCP/IP ソケット通信機能を提供しています。
TRANSFER	複数の DeviceServer 間の同期機能を提供しています。
WEBPROXY	HTTP サーバー、WebAPI 機能を提供しています。
UIOUSB	UIOUSB デバイスの管理機能を提供しています。
SCRIPT	Lua スクリプト実行環境と、DeviceServer Lua APIライブラリ機能を提供しています。
MAIL	メールサーバー管理機能を提供しています。
XBEE	XBee 802.15.4 Series1 デバイス管理機能を提供しています。
ZB	XBee-ZB ZigBee 対応 Series2 デバイス管理機能を提供しています。
MessageBackup	内部メッセージ交換機能とバックアップ機能を提供しています。
SERIAL	シリアルデバイス管理機能を提供しています。
MQTT	MQTT ブローカとの接続管理機能を提供しています。
LAST	ダミーモジュールでサービス起動確認に使用されます。

- **制限事項**

- **使用例**

```
-----------------------------------------------------
-- Oracle RDBMS モジュールが有効な場合のみデータを登録
-----------------------------------------------------
if module_stat["ORACLE"] then
        stat = add_stat_oracle(key,tostring(temperature))
        if not stat then error() end
end
```

```
local stat,module_stat = service_module_status()
if not stat then error() end
for key,val in pairs(module_stat) do
        log_msg(string.format("module_stat[%s] = %s",key,tostring(val)),file_id)
end
```

13.23 service_module_restart()

- **機能概要**

  DeviceServer のサービスモジュールを再起動する。

- **関数定義**

  stat = service_module_restart(sevice_module [,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。
service_module:String	DeviceServer のサービスモジュール名を指定する。
timeout:Number	サービスモジュール起動完了をチェックする時のタイムアウトまでの秒数を指定する。パラメータ省略時は 60 秒が設定される。

- **備考**

  スクリプト中から DeviceServer のサービスモジュールを再起動したい場合に、リスタート対象のサービスモジュール名をパラメータに指定して使用する。サービスモジュール名とその機能については、service_module_status() ライブラリ関数の項を参照して下さい。

  運用中に通信ポートやネットワークのエラーが発生して、リトライ等では自動的に復帰できない場合に、サービスモジュールを初期化するために使用する。

- **制限事項**

- **使用例（PERIODIC_TIMER スクリプト中に記述した例）**

```
--------------------------------------------------------------------------------
-- サーバーに接続した XBee デバイスのシリアルポートで通信エラーが発生している場合には、
--- "XBEE" サービスモジュールを再起動させる
--------------------------------------------------------------------------------
if not xbee_my_serial_number() then
        if not service_module_restart("XBEE") then error() end
end
```

```
--------------------------------------------------------------------------------
-- UIOUSB モジュールがハングアップしている場合にはモジュールを再起動させる
--------------------------------------------------------------------------------
if not uio_command("version") then
        if not service_module_restart("UIOUSB") then error() end
end
```

14 文字列操作API (Lua)

スクリプト内で、文字列を操作するためのライブラリ関数です。DeviceServer 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。

14.1 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(item_#1, ..., item_#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

  item_#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
```

14.2 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**

  value = csv_to_tbl(csv_str)

- **パラメータとリターン値**

  value:Table [1..#max_item] of String
  　　CSVで構成されていた各カラムの文字列

  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
```

14.3 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**

  value = hex_to_tbl(hexstr)

- **パラメータとリターン値**

  value:Table [1..#max_item] of Number
  　　取得した数値データ(0 から 255までの整数)

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
```

```
file_id = "TMP102_READ"
--[[
●機能概要
I2C バスに接続した温度センサー(TMP102) の値を取得する
●リクエストパラメータ
------------------------------------------------------------------------------
キー値                    値                              値の例
------------------------------------------------------------------------------
device  XBee-ZB デバイス64ビットアドレスまたは NodeIdentifier          "Node1"
●リターンパラメータ
------------------------------------------------------------------------------
キー値                    値                              値の例
------------------------------------------------------------------------------
temperature               センサーから取得した摂氏温度    "12.5"
                                                          "-25.0"
]]
local slave_addr = "48"
-------------------------
-- パラメータチェック
-------------------------
if not (g_params["device"]) then
        log_msg("parameter error",file_id)
        error()
end
---------------------------------------
-- 12 bit 幅の2の補数を符号付整数に変換
```

```
----------------------------------------

function calc_2comp(val)
  if(bit_and(val,0x800) ~= 0) then
    return -1 * (bit_and(bit_not(val),0xfff) + 1)
  else
    return val
  end
end

--------------------------------------------------------------------------

-- TMP102温度レジスタの値を取得する
-- pointer register 0x00 をセットした後、2 バイトのレジスタ値を取得する

--------------------------------------------------------------------------

stat,result = zb_tdcp_safe_retry(g_params["device"],"i2c_write," .. slave_addr .. ",00,2")
if (not stat) or (result[2] == "0") then error() end

----------------------------------------

-- 温度レジスタ値から摂氏温度を計算する

----------------------------------------

local reg = {}
reg = hex_to_tbl(result[3])
local temp_int = bit_lshift(reg[1],4) + bit_rshift(reg[2],4)
local temperature = 0.0625 * calc_2comp(temp_int)
script_result(g_taskid,"temperature",string.format("%3.1f",temperature))
```

14.4 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(value_#1, ... ,value_#n)

  hexstr = tbl_to_hex(value)

- **パラメータとリターン値**

value_#n:Number	#n番目の数値データ(0 から 255 までの整数のみが有効)
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

- **備考**

  リスト中の数値が、負数や255 以上の場合はエラーになります。

- **使用例**

```
data = "010203FDFEFF"
tbl = hex_to_tbl(data)
hexdata = list_to_hex(unpack(tbl))
if data == hexdata then
   log_msg("success !!")
end
```

14.5 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**

  value = str_to_tbl(str)

- **パラメータとリターン値**

  value:Table [1..#max_item] of Number
  　　取得した数値データ(0 から 255までの整数)

  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は、文字列データ値は Shift_JIS コードで取得されます。Lua スクリプトファイル自身は UTF-8 コードで記述されていますが、この関数で得られる数値データ列は Shift_JIS コードになる点に注意してください。UTF-8 コードで数値データ列を取得したい場合には、writeUTF_hex() ライブラリ関数の使用を検討してください。

- **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
```

14.6 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(value_#1, ... ,value_#n)

  str = tbl_to_str(value)

- **パラメータとリターン値**

  value_#n:Number　　#n番目の数値データ(0 から 255 までの整数のみが有効)

value:Table [1..#max_item] of Number　　変換対象の数値データ配列

str:String　　文字列

- **備考**

リスト中の数値が、負数や255 以上の場合はエラーになります。

数値を文字に変換したもの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は、制御コード等が文字列中に入る場合があります。

数値がマルチバイト文字を構成している場合には Shift_JIS 文字列として一旦変換された後、Lua スクリプトのリターン値(str) には、それを更に UTF-8 形式に変換したものが設定されます。数値データを直接 UTF-8 形式として文字列に変換したい場合には、readUTF_hex() ライブラリ関数の使用を検討してください。

- **使用例**

```
data = "test message!!"
tbl = str_to_tbl(data)
strdata = tbl_to_str(tbl)
if data == strdata then
   log_msg("success !!")
end
```

```
stat,frame,data = zb_at_command("0013A20040A0D533","NI","")
if not stat then error() end
log_msg("cmd_data = " .. data .. " ascii = " .. tbl_to_str(hex_to_tbl(data)),file_id)
```

14.7　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)
```

14.8 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バイト分のデータがはいっているものとする

- **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)
```

14.9 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**

arr = ssv_to_tbl(ssv_str)

- **パラメータとリターン値**

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
```

14.10 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 ビット演算API (Lua)

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。DeviceServer 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。

この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" 間のみが有効になりますので注意して下さい。

15.1 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**

  str = bit_tohex(x [,n])

- **パラメータとリターン値**

  str: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**

```
str = bit_tohex(255,4)
```

15.2 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)
```

15.3 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...])

- **パラメータとリターン値**

  y:Number　　整数値

  x1, x2...:Number　　整数値

- **備考**

- **使用例**

```
y = bit_or(1,2,4,8,16,32,64)
```

15.4 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...])

- **パラメータとリターン値**

  y:Number　　整数値

  x1, x2...:Number　　整数値

- **備考**

- **使用例**

```
y = bit_and(0x1234,0xff00)
```

15.5 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...])

- **パラメータとリターン値**

  y:Number　　整数値

  x1, x2...:Number　　整数値

- **備考**

- **使用例**

```
y = bit_xor(0xffff,0xaaaa)
```

15.6 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)
```

15.7 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 日付時間API (Lua)

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。

16.1 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 までの数字。(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 までの数字)
day:Number	日を指定(1 から31 までの数字)

- **備考**

- **使用例**

```
stat,wday = day_of_week(2009,1,31)
```

16.2 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 までの数字)

- **備考**

- **使用例**

```
stat,days = days_in_month(2009,1)
```

16.3 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 までの数字)
day:Number	日を指定(1 から31 までの数字)

- **備考**

- **使用例**

```
stat,y,m,d = inc_day(100,2009,1,31)
```

16.4 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 までの数字)
day:Number	日を指定(1 から31 までの数字)

hour:Number　　時を指定(0 から23 までの数字)

min:Number　　分を指定(0 から59 までの数字)

sec:Number　　秒を指定(0 から59 までの数字)

- **備考**

- **使用例**

```
stat,y,m,d,h,mm,s = inc_second(1,2009,1,31,23,59,59)
```

16.5 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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

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 までの数字)

- **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)
```

16.6 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 = 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
str:String	日付時刻を表した文字列。"YYYY/MM/DD HH:MM:SS" 形式で指定する

- **備考**

  str で指定可能な日時のフォーマットは "YYYY/MM/DD HH:MM:SS" です。

  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。

- **使用例**

```
stat,y,m,d,h,mm,s = str_to_datetime("2011/1/1 13:1:25")
```

17 GPS API (Lua)

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。DeviceServer 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。

17.1 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、DeviceServer の動作するPC 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”)
```

17.2 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)
```

17.3　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”)
```

17.4 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.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　　dmm 形式に変換した緯度

lon_deg: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)
```

18 UIOUSB API (Lua)

DeviceServer のスクリプトやイベントハンドラ中からUIOUSB デバイスを操作するためのライブラリ関数が使用できます。 予め、頻繁に利用される機能についての関数のみがライブラリに用意されていますが、イベント関連や UIOUSB コンフィギュレーション関連のコマンドなどを含む、全てのUIOUSB コマンドは uio_command()ライブラリ関数で実行できます。UIOUSB コマンドの機能や記述方法についての詳しい内容は “UIOUSBユーザーマニュアル” を参照して下さい。

18.1 uio_do()

- **機能概要**

  UIOUSB デバイスの I/O (8bit)に指定された値を出力する。

- **関数定義**

  stat = uio_do(value)

- **パラメータとリターン値**

  value:Number　　I/O に出力する値 (0 から255 までの整数)

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

- **備考**

  出力に指定するビットは、予め出力モードに設定しておく必要があります。

- **使用例**

```
for cnt = 0,255,1 do
  if not uio_do(cnt) then error() end
end
```

18.2 uio_don()

- **機能概要**

  UIOUSB デバイスの I/O (8bit)の指定ビットに値を出力する。

- **関数定義**

  stat = uio_don(bit, flag)

- **パラメータとリターン値**

  bit:Number　　出力対象の I/O ビット (0 から7 までの整数)

  flag:Boolean　　出力を High にする時は true、Low の場合は false を指定する。

  stat:Boolean　　成功した場合は true、失敗した場合は false が返る。

- **備考**

  出力に指定するビットは、予め出力モードに設定しておく必要があります。

  bit の値は、uio_di()で取得するテーブルのインデックス (1から8)の値より 1 小さくなる点に注意してください。

- **使用例**

```
for cnt = 0,7,1 do
   if not uio_don(cnt,true) then error() end
```

```
  wait_time(100)
end
for cnt = 7,0,-1 do
  if not uio_don(cnt,false) then error() end
  wait_time(100)
end
```

18.3 uio_di()

- **機能概要**

  UIOUSB デバイスの I/O 値(8bit)を取得する。

- **関数定義**

  stat, io = uio_di()

- **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。

  io:Table[1..8] of Boolean　　ビット毎のI/O 値

  io[1]:Boolean　　I/O bit#0 の値。Highの場合は true, Low の場合はfalse になる。

  io[2]:Boolean　　I/O bit#1 の値。Highの場合は true, Low の場合はfalse になる。

  io[3]:Boolean　　I/O bit#2 の値。Highの場合は true, Low の場合はfalse になる。

  io[4]:Boolean　　I/O bit#3 の値。Highの場合は true, Low の場合はfalse になる。

  io[5]:Boolean　　I/O bit#4 の値。Highの場合は true, Low の場合はfalse になる。

  io[6]:Boolean　　I/O bit#5 の値。Highの場合は true, Low の場合はfalse になる。

  io[7]:Boolean　　I/O bit#6 の値。Highの場合は true, Low の場合はfalse になる。

  io[8]:Boolean　　I/O bit#7 の値。Highの場合は true, Low の場合はfalse になる。

- **備考**

  ビット位置(0..7) が配列要素 (1..8) に対応している点に注意してください。

- **使用例**

```
stat,io = uio_di()
if not stat then error() end
val = 0
for i = 1,8,1 do
  if io[i] then
    val = val + math.pow(2,i - 1)
  end
end
log_msg(string.format("I/O = 0x%2.2x",val))
```

18.4 uio_ad()

- **機能概要**

  UIOUSB デバイスの A/D 変換値を取得する。

- **関数定義**

  stat, ad = uio_ad()

- **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。

  ad:Table[1..4] of Number　　A/D channel毎の変換値

  　ad[1]:Number　　A/D #0 の値 (0 から 1023までの整数)

  　ad[2]:Number　　A/D #1 の値 (0 から 1023までの整数)

  　ad[3]:Number　　A/D #2 の値 (0 から 1023までの整数)

  　ad[4]:Number　　A/D #3 の値 (0 から 1023までの整数)

- **備考**

  A/D channel(#0..#3) が配列要素 (1..4) に対応している点に注意してください。

- **使用例**

```
for cnt = 1,10,1 do
  stat,ad = uio_ad()
  if not stat then error() end
  log_msg(string.format("%d %d %d %d",ad[1],ad[2],ad[3],ad[4]))
  wait_time(10)
end
```

18.5 uio_pwm()

- **機能概要**

  UIOUSB デバイスのPWM 出力の設定値の取得、または有効・無効フラグの設定。

- **関数定義**

  stat [,pwm, duty] = uio_pwm([pwm_ch, flag])

- **パラメータとリターン値**

  pwm_ch:Number　　設定対象の PWM チャネル番号 (1 または 2)

  flag:Boolean　　PWM 出力を有効にする時は true, 無効にする時は false を指定する。

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  pwm:Table[1..2] of Boolean

  　pwm[1]:Boolean　　PWM ch#1 の出力フラグ。有効な場合は true, 無効の場合は false になる。

  　pwm[2]:Boolean　　PWM ch#2 の出力フラグ。有効な場合は true, 無効の場合は false になる。

  duty:Table[1..2] of Number

  　duty[1]:Number　　PWM ch#1 のデューティ値 (0から1023 までの整数)

  　duty[2]:Number　　PWM ch#2 のデューティ値 (0から1023 までの整数)

- **備考**

  入力パラメータを省略した場合には現在の設定値がリターン値に返ります。入力パラメータを指定した場合には設定が更新されてリターン値には stat のみを返します。

- **使用例**

```
stat,pwm,duty = uio_pwm()
if not stat then error() end
for i,v in ipairs(pwm) do
   log_msg(string.format("pwm[%d] = %s",i,tostring(v)))
   wait_time(10);
end
for i,v in ipairs(duty) do
   log_msg(string.format("duty[%d] = %s",i,tostring(v)))
   wait_time(10);
end
```

18.6 uio_duty()

- **機能概要**

  UIOUSB デバイスの PWM デューティ値の変更。

- **関数定義**

  stat = uio_duty(pwm_ch, duty)

- **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。
pwm_ch:Number	設定対象の PWM チャネル番号 (1 または 2)
duty:Number	設定対象の PWM のデューティ値(0から1023 までの整数) 1023 よりも大きな値が指定された場合は 1023に設定される。

- **備考**

  現在のduty 値を取得する場合には uio_pwm() を使用して下さい。

- **使用例**

```
-----------------------------------------------------
-- UIOUSB デバイスの PWM1 にLED が接続されている時に、
-- ろうそくの光のような瞬きをするデモ
-----------------------------------------------------
function led_output(x)
 local duty = math.ceil(x * 1024 + 50) -- 完全消灯にしない
 if not uio_duty(1,duty) then error() end
```

```
 wait_time(10)
end


local xt1 = math.random()
local xt2
if not uio_pwm(1,true) then error() end
for cnt = 0,2000,1 do
 -- 間欠カオス計算 によってLED の明るさにゆらぎを与える
 led_output(xt1)
 if xt1 < 0.5 then
   xt2 = xt1 + 2 * xt1 * xt1
 else
   xt2 = xt1 - 2 * (1 - xt1) * (1 - xt1)
 end
 xt1 = xt2
end
if not uio_pwm(1,false) then error() end
```

18.7 uio_servo()

- **機能概要**

  UIOUSB デバイスの SERVO 設定値の取得もしくは設定

- **関数定義**

  stat [,servo, pos] = uio_servo([bit, 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。

  bit:Number　　出力対象の I/O ビット (0 から7 までの整数)

  flag:Boolean　　SERVO 出力を有効にする時は true, 無効にする時は false を指定する。

  servo:Table [1..8] of Boolean

  servo[1]:Boolean　　I/O bit#0 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[2]:Boolean　　I/O bit#1 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[3]:Boolean　　I/O bit#2 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[4]:Boolean　　I/O bit#3 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[5]:Boolean　　I/O bit#4 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[6]:Boolean　　I/O bit#5 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[7]:Boolean　　I/O bit#6 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  servo[8]:Boolean　　I/O bit#7 サーボ信号出力フラグ。有効な場合は true,無効の場合は false になる。

  pos:Table [1..8] of Number

  pos[1]:Number　　I/O bit#0 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

  pos[2]:Number　　I/O bit#1 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[3]:Number　　I/O bit#2 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[4]:Number　　I/O bit#3 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[5]:Number　　I/O bit#4 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[6]:Number　　I/O bit#5 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[7]:Number　　I/O bit#6 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

pos[8]:Number　　I/O bit#7 サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

- **備考**

入力パラメータを省略した場合には現在の設定値がリターン値に返ります。入力パラメータを指定した場合には設定が更新されてリターン値には stat のみを返します。

SERVO 出力に指定するビットは、予め出力モードに設定しておく必要があります。

bit の値は、uio_servo()で取得するテーブルのインデックス (1から8)の値より1小さくなる点に注意してください。

サーボは非常に電気を消費しますので、電源容量に余裕のある外部電源を使用してください。サーボ動作時の電源電圧低下やノイズの影響でUIOUSB デバイスにリセットがかかる場合がありますので、使用時には注意してください。

- **使用例**

```
stat,servo,pos = uio_servo()
if not stat then error() end
for cnt = 1,8,1 do
  log_msg(string.format("servo%d %s pos %d",cnt - 1,tostring(servo[cnt]),pos[cnt]))
end
```

18.8　uio_pos()

- **機能概要**

UIOUSB デバイスの SERVO 信号のパルス幅の設定。

- **関数定義**

stat = uio_pos(bit, pos)

- **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。

bit:Number　　出力対象の I/O ビット (0 から7 までの整数)

pos:Number　　サーボ信号のパルス幅 10μs単位(60 から240 までの整数)

- **備考**

現在のpos 値を取得する場合には uio_servo() を使用して下さい。

SERVO 出力に指定するビットは、予め出力モードに設定しておく必要があります。

bit の値は、uio_servo()で取得するテーブルのインデックス（1から8)の値より1小さくなる点に注意してください。

サーボによって設定可能な（可動可能な）パルス幅の範囲に違いがあります。サーボの仕様を確認の上、無理のかからない範囲でpos 値を設定してください。R/C 用サーボの場合の中点は 150(1.5ms) になっていると思いますので、この値を中心に +- 50 位が普通の可動範囲です。100(1.0ms) .. 200(2.0ms)

- **使用例**

```
if not uio_pos(0,150) then error() end
if not uio_servo(0,true) then error() end
wait_time(1000)
if not uio_pos(0,200) then error() end
wait_time(1000)
if not uio_pos(0,100) then error() end
wait_time(1000)
if not uio_pos(0,150) then error() end
if not uio_servo(0,false) then error() end
```

18.9 uio_command()

- **機能概要**

UIOUSB デバイスのコマンドを実行する。

- **関数定義**

stat, reply = uio_command(cmdstr [,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。
cmdstr:String	UIOUSB デバイスのコマンド文字列
timeout:Number	UIOUSB デバイス応答が、指定した値(ミリ秒)を超えてもなかった場合にはエラーとしてリトライ動作を行なう(最大2 回まで) パラメータ省略時には、100(ms) が指定されたものとする。
reply:String	コマンド実行結果のリプライ文字列。 UIOUSB デバイスが出力した、リプライ文字列の “>” よりも後ろの部分が入る

- **備考**

UIOUSB コマンドの詳しい説明は、”UIOUSBユーザーマニュアル”を参照してください。

- **使用例**

```
if not uio_command("dcfg 0") then error() end
if not uio_command("pullup 0") then error() end
if not uio_command("duty1 0") then error() end
if not uio_command("duty2 0") then error() end
```

```
if not uio_command("pwm1 1") then error() end
if not uio_command("pwm2 1") then error() end
```

18.10 uio_force_sample()

- **機能概要**

  手動サンプリングを行なって現在のI/Oポート値, A/D変換値, カウンタ値を取得する。

- **関数定義**

  stat, counter, io, ad = uio_force_sample()

- **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。

  counter:Number　　カウンタ値

  io:Table[1..8] of Boolean　　ビット毎のI/O 値

  - io[1]:Boolean　　I/O bit#0 の値。Highの場合は true, Low の場合はfalse になる。
  - io[2]:Boolean　　I/O bit#1 の値。Highの場合は true, Low の場合はfalse になる。
  - io[3]:Boolean　　I/O bit#2 の値。Highの場合は true, Low の場合はfalse になる。
  - io[4]:Boolean　　I/O bit#3 の値。Highの場合は true, Low の場合はfalse になる。
  - io[5]:Boolean　　I/O bit#4 の値。Highの場合は true, Low の場合はfalse になる。
  - io[6]:Boolean　　I/O bit#5 の値。Highの場合は true, Low の場合はfalse になる。
  - io[7]:Boolean　　I/O bit#6 の値。Highの場合は true, Low の場合はfalse になる。
  - io[8]:Boolean　　I/O bit#7 の値。Highの場合は true, Low の場合はfalse になる。

  ad:Table[1..4] of Number　　A/D channel毎の変換値

  - ad[1]:Number　　A/D #0 の値 (0 から 1023までの整数)
  - ad[2]:Number　　A/D #1 の値 (0 から 1023までの整数)
  - ad[3]:Number　　A/D #2 の値 (0 から 1023までの整数)
  - ad[4]:Number　　A/D #3 の値 (0 から 1023までの整数)

- **備考**

  ad 値は A/D channel(#0..#3) が配列要素 (1..4) に対応している点に注意してください。

  io 値は、ビット位置(0..7) が配列要素 (1..8) に対応している点に注意してください。

- **使用例**

```
stat,cnt,io,ad = uio_force_sample()
if not stat then error() end
val = 0
for i = 1,8,1 do
  if io[i] then
    val = val + math.pow(2,i - 1)
  end
end
```

```
log_msg(string.format("COUNTER = %d",cnt))
log_msg(string.format("I/O = 0x%2.2x",val))
log_msg(string.format("A/D = %d %d %d %d",ad[1],ad[2],ad[3],ad[4]))
```

19 メールAPI (Lua)

スクリプト内でメールボックスを操作して、メールの一覧やメール本文の取得や削除などを行うためのライブラリ関数です。これらのライブラリ関数を使用する場合には、予めメールサーバー(POP サーバー、SMTP サーバー)の設定と、メールアカウントの設定を DeviceServer に行ってください。

19.1 mail_send()

- **機能概要**

  メールを送信する。

- **関数定義**

  stat = mail_send(to, from, subject, mail_body_line1, ..., mail_body_line#n)

  stat = mail_send(to, from, subject, mail_body_tbl)

- **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。
to:String	メールの宛先
from:String	メールの差出名。空文字列を与えた場合は、サーバー設定プログラムの"自分のメールアドレス" 設定で指定したメールアドレスが指定される。
subject:String	メールの件名
mail_body_line1:String	メール本文1行目
mail_body_line#n:String	メール本文#n行目
mail_body_tbl:Table [1..#max_line] of String	メール本文(文字列配列で指定する場合)

- **備考**

  メール送信を行うためにサーバー設定プログラムでSMTPメールサーバーを事前に設定して下さい。

  メールは、アルファベットと日本語のみに対応しています。それ以外の文字エンコードには対応していません。メール本文及びヘッダにはアスキーまたは日本語(JIS)コードで表現できない文字は使用できません。

  メール送信時にはメールヘッダの Content-type にはメール本文の内容に関わらず常に 'text/plain' が設定され、キャラクタセットは'ISO-2022-JP'を設定します。

- **使用例**

```
body = {}
table.insert(body,"本文 1 行目")
```

```
table.insert(body,"本文 2 行目")
table.insert(body,"本文 3 行目")
stat = mail_send("宛先名 <xxxxx@aaa.bbb.ccc.ddd>","差出名 <zzzz@vvvv.wwww.xxxx>","これはテストの件名です",unpack(body))
if not stat then error() end
```

19.2 mail_retrieve_header()

- **機能概要**

  メールサーバー(POP)から、メールヘッダー覧を取得する。

- **関数定義**

  stat, timestamp, from, recipients, subject, priority, id = mail_retrieve_header()

- **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:Table [1..#max_mail] of String　　メールヘッダの TimeStamp 値

  from:Table [1..#max_mail] of String　　メールヘッダの From 値

  recipients:Table [1..#max_mail] of String　　メールヘッダの Recipients 値

  subject:Table [1..#max_mail] of String　　メールヘッダの Subject 値

  priority:Table [1..#max_mail] of String　　メールヘッダの Priority 値

  id:Table [1..#max_mail] of String　　メールヘッダの Id 値

- **備考**

  メール受信を行うために、事前にPOPメールサーバーをサーバー設定プログラムで設定して下さい。

  メールボックスにあるメールの数は、#max_mail で、テーブルサイズと同じになります。テーブルサイズは、関数で取得する全てのリターン値(timestamp, from, recipients, subject, priority, id)で同一の値になります。メールが１つも無い場合は、内容が空のテーブルを取得します。

  メールは、アルファベットと日本語のみに対応しています。それ以外の文字エンコードには対応していません。

- **使用例**

```
stat,timestamp,from,recipients,subject,priority,id = mail_retrieve_header()
if not stat then error() end
for key,val in ipairs(id) do
   log_msg("Date:" .. timestamp[key])
   log_msg("Subject:" .. subject[key])
   log_msg("From:" .. from[key])
   log_msg("--------")
end
```

19.3 mail_receive()

- **機能概要**

  メール本文を受信する

- **関数定義**

  stat, header, body = mail_receive(id [,delete_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。

  id:String　　メールID

  delete_flag:Boolean　　メール受信時にPOPサーバーからメールを削除する場合は trueを指定する。
  削除しない場合は false を指定する。パラメータ省略時は true になる。

  header:Table メールのヘッダ情報

  header["TimeStamp"]:String　　メールヘッダの TimeStamp 値

  header["From"]:String　　メールヘッダの From 値

  header["Recipients"]:String　　メールヘッダの Recipients 値

  header["Subject"]:String　　メールヘッダの Subject 値

  header["Priority"]:String　　メールヘッダの Priority 値

  header["MissedParts"]:String　　メール本文がマルチパートで構成されていて、読み出しに失敗したパートがあった場合は "Yes" が入る。それ以外は "No" が入る。

  body:Table [1..#max_line] of String　　メール本文テキスト

- **備考**

  メール受信を行うために、事前にPOPメールサーバーをサーバー設定プログラムで設定して下さい。

  メール本文(body) 中の行数は、#max_line で、テーブルサイズと同じになります。

  マルチパートで構成されたメールや、添付ファイル付きのメールは受信できません。このときには、header["MissedParts"]の値が “Yes” になります。”Yes” の場合のリターン値 body の内容は不定です。（取得に成功したメールの一部分が入っています）

  MIME エンコードが正しくメール中に指定されていないと、本文テキストがデコードできない場合があります。受信時のContent-type が ‘text/plain’ 以外の場合でも、メール本文テキストはそのままの形で取り出されて body に格納されます。

  メールは、アルファベットと日本語のみに対応しています。それ以外の文字エンコードには対応していません。

- **使用例**

```
-------------------------------------------------------------------------------
-- POP メールサーバー内の、メールリスト最後のメール内容を取り出してログに表示する
-------------------------------------------------------------------------------
stat,timestamp,from,recipients,subject,priority,id = mail_retrieve_header()
```

```
if not stat then error() end
if #id ~= 0 then
  log_msg("last id = " .. id[#id])
  stat,header,body = mail_receive(id[#id] ,true)
  if not stat then error() end
  for key,val in pairs(header) do
    log_msg(string.format("header[%s] = %s",key,val))
    wait_time(10)
  end

  if header["MissedParts"] == "No" then
    for bkey,bval in ipairs(body) do
      log_msg("body[" .. tostring(bkey) .. "]" .. bval)
      wait_time(10)
    end
  end
end
```

19.4 mail_delete()

- **機能概要**

  メールを削除する。

- **関数定義**

  stat = mail_delete(id#1, ..., id#n)

- **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。

  id#1:String　　メールID #1

  id#n:String　　メールID #n

- **備考**

  複数のメールID を指定した場合は、POPメールサーバーから指定したメールを全て削除します。

  メール操作を行うために、事前にPOPメールサーバーをサーバー設定プログラムで設定して下さい。

- **使用例**

```
--------------------------------------------------------------------------------
-- POP メールサーバー内の、メールを全て削除する
--------------------------------------------------------------------------------
stat,timestamp,from,recipients,subject,priority,id = mail_retrieve_header()
if not stat then error() end
if #id ~= 0 then
```

```
  stat = mail_delete(unpack(id))
  if not stat then error() end
end
```

20 アラームデバイスAPI (Lua)

スクリプト内でアラームデバイスを操作するためのライブラリ関数です。

アラームデバイスのシグナルステータスを更新したり、I/O ポート値の取得や変更ができます。

これらのライブラリ関数を使用する場合には、予めアラームデバイスの設定を DeviceServer に行ってください。

20.1 alarm_signal_get()

- **機能概要**

  アラームデバイスのシグナル(ランプ、ブザー)の現在の設定値を取得する。

  デバイスタイプが ALARMSIGNAL,SIGSENSOR のみ有効です。

- **関数定義**

  stat, signal = alarm_signal_get(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　　デバイス名

  signal:Table　　デバイスの現在のシグナル設定値

  signal["Red"]:Boolean　　"Red" ランプの設定。ON の場合は true, OFF の場合は false になる。

  signal["Yellow"]:Boolean　　"Yellow" ランプの設定。ON の場合は true, OFF の場合は false になる。

  signal["Green"]:Boolean　　"Green" ランプの設定。ON の場合は true, OFF の場合は false になる。

  signal["BlinkingRed"]:Boolean　　"BlinkingRed" ランプの設定。ON の場合は true, OFF の場合は false になる。

  signal["BlinkingYellow"]:Boolean　　"BlinkingYellow" ランプの設定。ON の場合は true, OFF の場合は false になる。

  signal["BlinkingGreen"]:Boolean　　"BlinkingGreen" ランプの設定。 ON の場合は true, OFF の場合は false になる。

  signal["Beep1"]:Boolean　　"Beep1" の設定。ON の場合は true, OFF の場合は false になる。

  signal["Beep2"]:Boolean　　"Beep2" の設定。ON の場合は true, OFF の場合は false になる。

- **備考**

- **使用例**

```
device_name = "SigSensor"
stat,signal = alarm_signal_get(device_name)
if not stat then error() end
```

```
for key,val in pairs(signal) do
  log_msg(string.format("signal[%s] = %s",key,tostring(val)))
end
```

20.2 alarm_signal_set()

- **機能概要**

  アラームデバイスのシグナル(ランプ、ブザー)を設定する。

  デバイスタイプが ALARMSIGNAL, SIGSENSOR のみ有効です。

- **関数定義**

  stat = alarm_signal_set(name, signal, 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。
name:String	デバイス名
signal:String	設定を行うシグナルの名前。下記の中から一つを指定する。
	"Red"
	"Yellow"
	“Green"
	"BlinkingRed"
	"BlinkingYellow"
	"BlinkingGreen"
	"Beep1"
	"Beep2"
flag:Boolean	シグナルの状態を ON にする時は true, OFF にする時は false を指定する。

- **備考**

- **使用例**

```
device_name = "SigSensor"
stat = alarm_signal_set(device_name,"Red",true)
if not stat then error() end
stat = alarm_signal_set(device_name,"Beep1",true)
if not stat then error() end
wait_time(1000)
stat = alarm_signal_set(device_name,"Red",false)
if not stat then error() end
stat = alarm_signal_set(device_name,"Beep1",false)
if not stat then error() end
```

20.3 alarm_signal_message()

- **機能概要**

  アラームデバイスのLCD に文字列を表示する。

  デバイスタイプが SIGSENSOR のみ有効です。

- **関数定義**

  stat = alarm_signal_message(name, message)

- **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	デバイス名
message:String	LCD に表示する文字列。(LCDモジュールで表示可能な ASCII 文字のみで、32 文字まで)

- **備考**

  LCD表示を元のクロック表示に戻すには、SIGSENSOR デバイスのプッシュボタン(RST_BTN)を押すか、alarm_signal_reset() を使用してください。

- **使用例**

```
device_name = "SigSensor"
stat = alarm_signal_message(device_name,"!#$%&'()=<>?_+{}This is test!!")
if not stat then error() end
```

20.4 alarm_signal_reset()

- **機能概要**

  アラームデバイスのシグナル(ランプ、ブザー)をOFF にして、LCD 表示を時刻表示(初期状態)にする。

- **関数定義**

  stat = alarm_signal_reset(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	デバイス名

- **備考**

  デバイス上のプッシュボタン(RST_BTN)を押したときと同様の動作になります。

- **使用例**

```
device_name = "SigSensor"
stat = alarm_signal_reset(device_name)
if not stat then error() end
```

20.5 alarm_signal_clock()

- **機能概要**

  アラームデバイスの時計をサーバーに合わせる。

  デバイスタイプが SIGSENSOR のみ有効です。

- **関数定義**

  stat = alarm_signal_clock(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　　デバイス名

- **備考**

  DeviceServer の動作しているPC の時計を元に、時刻合わせを行います。DeviceServer とアラームデバイス間の、ネットワークパケットの転送にかかる時刻遅延は考慮していません。

- **使用例**

```
device_name = "SigSensor"
stat = alarm_signal_clock(device_name)
if not stat then error() end
```

20.6 alarm_network_reset()

- **機能概要**

  DeviceServerに保存された、全アラームデバイスの接続エラーフラグをリセットする。

- **関数定義**

  stat = alarm_network_reset()

- **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**

  この関数をコールした後で、DeviceServer がアラームデバイスに対してネットワーク接続時にエラーを検出した場合は、ALARM_NETWORK_ERROR イベントが発生します。

- **使用例**

```
stat = alarm_network_reset()
if not stat then error() end
```

20.7 alarm_dout_get()

- **機能概要**

  アラームデバイスのDOUTPUT の値を取得する。

  デバイスタイプが SIGSENSOR, NETUIO のみ有効です。

- **関数定義**

  stat, dout = alarm_dout_get(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　　デバイス名

  dout:Number　　DOUTPUT に出力された設定値。出力ビットの値を数値に変換した値が返る。

- **備考**

  アラームデバイスに対して最後に DOUTPUT を出力したときの値が返るので、実際のポートの値は読み込みません。

- **使用例**

```
device_name = "SigSensor"
stat,dout = alarm_dout_get(device_name)
if not stat then error() end
log_msg(string.format("dout = %2.2x",dout))
```

20.8 alarm_dout_set()

- **機能概要**

  アラームデバイスのDOUTPUT の値を設定する。

  デバイスタイプが SIGSENSOR, NETUIO のみ有効です。

- **関数定義**

  stat, dout = alarm_dout_set(name, dout)

- **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　　デバイス名

  dout:Number　　DOUTPUT に出力する設定値。出力ビットの値を数値に変換した値で指定する。

- **備考**

  dout に指定可能な値は、デバイスタイプがSIGSENSOR の場合は 0 から 15 の値で、NETUIO の場合は 0から 255 までになります。

- **使用例**

```
device_name = "SigSensor"
stat = alarm_dout_set(device_name,0x0f)
if not stat then error() end
```

20.9 alarm_din_get()

- **機能概要**

アラームデバイスのDINPUT 値の取得。

デバイスタイプが SIGSENSOR, NETUIO のみ有効です。

- **関数定義**

stat, din = alarm_din_get(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　　デバイス名

din:Number　　DINPUT に入力された値。入力ビットの値を数値に変換した値が返る。

- **備考**

- **使用例**

```
device_name = "SigSensor"
stat,din = alarm_din_get(device_name)
if not stat then error() end
log_msg(string.format("din = %2.2x",din))
```

20.10　alarm_adbuff_get()

- **機能概要**

アラームデバイスのA/D 変換した値（デバイス内の変換値格納バッファ）を取得する。

デバイスタイプが SIGSENSOR, NETUIO のみ有効です。

- **関数定義**

stat, buff1, bstat1, buff2, bstat2, buff3, bstat3, buff4, bstat4 = alarm_adbuff_get(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　　デバイス名

buff1:Table [1..1024] of Number　　A/D #0 変換バッファの値 (0 から 1023までの整数)

bstat1:Table　　A/D #0 変換バッファのステータス値(以下詳細)

bstat1["Overflow"]:boolean　　最後にデバイスのバッファを読み出してからバッファの上書きが発生したかどうかを示す。

bstat1["NextWritePointer"]:Number　　次にA/D 変換を行ったときに変換値を書き込む予定のバッファのポインタを示す。

bstat1["TransferPointer"]:Number　　前回バッファを読み出した時の、バッファの変換値書き込み済みのポインタを示す。（ただし例外として、デバイスのリセット直後は変換値が未書き込みであっても 0 が返る）

bstat1["Sample"]:Number　　前回バッファを読み出してから、今回読みだすまでに新規にバッファ中

	に書き込まれた変換値の個数 +1 を示す。常に、最後に変換を行った数分の 1 が足されている点に注意。(ただし例外として、デバイスのリセット直後は 0 を返し、リセット直後から最初の一回目の読み込み動作を行うときだけ、新規にバッファ中に書き込まれた変換値の個数が返る) Overflow が発生した場合は常に 1024 を示す。
bstat1["LastValue"]:Number	バッファ中にある最後に格納した変換値。
bstat1["MeanValue"]:Number	最後に転送を行った時のLastValue 変換データと新規にバッファ中に取り込んだデータがあればそれを加えた変換データ値の平均値。
bstat1["MaxValue"]:Number	最後に転送を行った時のLastValue 変換データと新規にバッファ中に取り込んだデータがあればそれを加えた変換データ値中の最大値。
bstat1["MinValue"]:Number	最後に転送を行った時のLastValue 変換データと新規にバッファ中に取り込んだデータがあればそれを加えた変換データ値中の最小値。
buff2:Table [1..1024] of Number	A/D #1 変換バッファの値 (0 から 1023までの整数)
bstat2:Table	A/D #1 変換バッファのステータス値(構造は bstat1 と同等)
buff3:Table [1..1024] of Number	A/D #2 変換バッファの値 (0 から 1023までの整数)
bstat3:Table	A/D #2 変換バッファのステータス値(構造は bstat1 と同等)
buff4:Table [1..1024] of Number	A/D #3 変換バッファの値 (0 から 1023までの整数)
bstat4:Table	A/D #3 変換バッファのステータス値(構造は bstat1 と同等)

- **備考**

Sample 値は、Max, Min, Mean の値を計算するときの対象サンプル数に等しくなります。前回バッファの値を読み出してから新しいデータがバッファに格納されていない場合に、Max, Min, Mean の値は 0 ではなく最後にバッファに書き込んだ値を示すようになっていて、そのときのSample は常に 1 を示します。

もし、前回バッファの値を読み出してから新規にバッファに格納された値のみで Max, Min, Mean 値を計算したい場合は、バッファから読み出したデータと、Overflow, NextWritePointer, TransferPointerの値を利用して有効な値のみで計算してください。

デバイスからバッファの値を読み込む時に、デバイス -> DeviceServer ->クライアントへの転送を行っている間にも、新規にA/D 変換が行われて、バッファに書き込まれている場合があります。このため、この関数で取得した A/D channel 毎のOverflow, NextWritePointer, TransferPointerの値は、違った値になっているので注意してください。この時でも、バッファ中の同一インデックスの示す変換値は、同時刻にサンプリングされたものになっています。(厳密には同時刻ではなく、H8/3069 デバイスが持つA/D 変換機能のスキャンモードを使用しています)

- **使用例**

```
device_name = "SigSensor"
stat,buff1,bstat1,buff2,bstat2,buff3,bstat3,buff4,bstat4 = alarm_adbuff_get(device_name)
if not stat then error() end
for cntr = 1,10,1 do
  log_msg(string.format("ad1,ad2,ad3,ad4 [%d] = %d,%d,%d,%d
",cntr,buff1[cntr],buff2[cntr],buff3[cntr],buff4[cntr]),file_id)
  wait_time(10)
end
```

20.11 alarm_adrange_flag_set()

- **機能概要**

  アラームデバイスのA/D レンジチェックの有効フラグをセットする。

  デバイスタイプが SIGSENSOR, NETUIO のみ有効です。

- **関数定義**

  stat = alarm_adrange_flag_set(name, channel, high_or_low, 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。
name:String	デバイス名
channel:Number	設定を行うA/D channel 番号 1, 2, 3, 4 のいずれかを指定する。
high_or_low:Boolean	HighまたはLow のどちらのレンジチェックフラグを指定するかを示す。 true の場合は High 側、false の場合は Low 側の指定になる。
flag:Boolean	A/D レンジチェックを有効にする時は true, 無効 にする時は false を指定する。

- **備考**

  デバイス上の A/D レンジチェックを有効に設定して、各 channel 毎の制限値を事前に設定しておく必要があります。

- **使用例**

```
-------------------------------------------------
-- デバイス上の全ての A/D 変換レンジチェックフラグを
-- 有効に設定する
-------------------------------------------------
device_name = "SigSensor"
for ch = 1,4,1 do
 stat = alarm_adrange_flag_set(device_name,ch,true,true)
 if not stat then error() end
 stat = alarm_adrange_flag_set(device_name,ch,false,true)
 if not stat then error() end
end
```

20.12 alarm_sysalert_list()

- **機能概要**

  システムアラートデバイスとして登録されているアラームデバイスリストを取得する。

- **関数定義**

  stat, name, type = alarm_sysalert_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。

  name:Table [1..#max_device] of String　　デバイス名リスト

  type:Table [1..#max_device] of String　　デバイスタイプ名リスト

- **備考**

  接続エラーが発生しているデバイスはリストから除外します。

- **使用例**

```
stat,name,type = alarm_sysalert_list()
if not stat then error() end
for key,val in ipairs(name) do
 log_msg(string.format("device[%d] = %s [%s]",key,val,type[key]))
end
```

20.13 alarm_all_list()

- **機能概要**

  登録されているアラームデバイスリストを取得する。

- **関数定義**

  stat, name, type, sysalert, neterr = alarm_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。

  name:Table [1..#max_device] of String　　デバイス名リスト

  type:Table [1..#max_device] of String　　デバイスタイプ名リスト

  sysalert:Table [1..#max_device] of Boolean　　デバイスがシステムアラートデバイスとして登録されている場合は true、そうでない場合は false に設定される。

  neterr:Table [1..#max_device] of Boolean　　接続エラーが発生している場合は true、そうでない場合は false に設定される。

- **備考**

  デバイス管理プログラムで、無効に設定されているデバイスはリストから除外します。

- **使用例**

```
local stat,name,type,sysalert,neterr = alarm_all_list()
if not stat then error() end
for key,val in ipairs(name) do
   log_msg(string.format("device[%d] = %-15s[%s]¥tsysalert = %5s¥tneterr = %5s",
           key,val,type[key],tostring(sysalert[key]),tostring(neterr[key])))
end
```

21 グローバル共有データAPI (Lua)

- グローバル共有領域について

グローバル共有データは、DeviceServer 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。
グローバル共有データは、DeviceServer 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データベース API を使用してください。

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することができます。

キー・バリュー領域の他に、文字列リスト形式で扱うためのグローバル共有文字列リストも提供しています。文字列リスト自身を表す、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、DeviceServer が動作している PC 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)。

- リモート共有ライブラリ関数 xxxx_net_xxxx() について

ライブラリ関数名が xxxx_net_xxxx() のものは、リモート側の DeviceServer 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 DeviceServer 内のスクリプトから xxxx_shared_xxxx() ライブラリ関数で操作するものと同じものです。

xxxx_net_xxxx() 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"プログラムの "デフォルト・リモートホスト" 設定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、DeviceServer 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 DeviceServer 側では、デスクトッププログラム(ABDeskTop) の "許可"ボタンで設定したホスト名リストに、xxxx_net_xxxx() 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 DeviceServer 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 DeviceServer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_xxxx() ライブラリ関数を実行した DeviceServer 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_xxxx() 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。DeviceServer 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。DeviceServer 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 DeviceServer 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。

21.1 get_shared_data(), get_net_data()

- **機能概要**

  DeviceServerの共有データ領域から 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**

  stat, value = get_shared_data(key [,delete_flag])

  stat, value = get_net_data(key [,delete_flag] [,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
key:String	共有データのkey 文字列
delete_flag: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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

- **使用例**

```
stat,val = get_shared_data(“KeyName1”)
if (stat) then
  log_msg(“data = “ .. val)
end
```

21.2 set_shared_data(), set_net_data()

- **機能概要**

  DeviceServerの共有データ領域の key とそれに対応する値 value を設定する。key が既に登録済みの場合には value で指定した値に更新される。

- **関数定義**

  stat = set_shared_data(key, value [,no_global_watch])

  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。(128 文字以内の文字列)
value:String	共有データのKey に対応する値。(128 文字以内の文字列)
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。
no_global_watch:Boolean	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する。パラメータ省略時は false になる。

- **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。

DeviceServer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
```

21.3 inc_shared_data(), inc_net_data()

- **機能概要**

DeviceServerの共有データ領域の 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。
no_global_watch:Boolean	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する。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。

DeviceServer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))
```

21.4 dec_shared_data(), dec_net_data()

- **機能概要**

DeviceServerの共有データ領域の 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。
no_global_watch:Boolean	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する。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。

  DeviceServer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))
```

21.5 add_shared_strlist(), add_net_strlist()

- **機能概要**

  DeviceServerの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、新しい文字列エントリを追加する。

- **関数定義**

  stat = add_shared_strlist(channel, value [,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値(文字列)
unique:Boolean	true を指定すると、指定した文字列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。もし保持できる制限数以上に文字列エントリを追加した場合には古いものから順に自動削除される。

remote_host:String　　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

DeviceServer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
```

21.6　clear_shared_strlist(), clear_net_strlist()

- **機能概要**

DeviceServerの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_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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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
```

21.7 get_shared_strlist(), get_net_strlist()

- **機能概要**

DeviceServerの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, 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）

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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

DeviceServer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
```

21.8 remove_shared_strlist(), remove_net_strlist()

- **機能概要**

DeviceServerの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 で指定した文字列に一致するエントリを削除する。

- **関数定義**

stat = remove_shared_strlist(channel, value)

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値(文字列)
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

value で指定した文字列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
```

21.9 shift_shared_strlist(), shift_net_strlist()

- **機能概要**

DeviceServerの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文字列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, 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	取得した文字列
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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

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 を設定します。

DeviceServer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
```

21.10 pop_shared_strlist(), pop_net_strlist()

- **機能概要**

DeviceServerの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文字列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, 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	取得した文字列
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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

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 を設定します。

DeviceServer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
```

21.11 count_shared_strlist(), count_net_strlist()

- **機能概要**

DeviceServerの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_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	文字列リスト中の文字列エントリ数。 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

DeviceServer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
```

21.12 list_shared_strlist(), list_net_strlist()

- **機能概要**

DeviceServerの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_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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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
```

22　データベースAPI (Lua)

- データベース(パーマネントデータ)領域について

データベース API は　キー・バリュー・データ領域で、DeviceServer 内蔵の Firebird データベースまたは、Oracle RDBMS にデータを保存します。DeviceServer や PC の再起動を行った場合でも、最後に保存した内容が保持されています。

API の xxxx_permanent_xxxx() の名前が付いた関数は、DeviceServer 内蔵の Firebird に保存します。これらのライブラリ関数はインストール直後から常に使用することができます。

API の xxxx_oracle_xxxx() の名前が付いた関数は、外部の Oracle RDBMS に作成した表領域にデータを保存します。Oracle RDBMS 用のAPI 関数を使用する場合には、予めOracle接続機能の設定を行う必要があります。"Oracle接続機能"の章を参照してください。

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。

キー・バリュー領域の他に、文字列リスト形式で扱うためのデータベース文字列リストも提供しています。文字列リスト自身を表す、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。データベース文字列リストもキー・バリュー形式のデータベース領域と同様に、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、DeviceServer が動作している PC 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)。

- リモートデータベースライブラリ関数 xxxx_per_net_xxxx() について

ライブラリ関数名が xxxx_per_net_xxxx() のものは、リモート側の DeviceServer のデータベース領域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データベース領域のデータはリモート側の DeviceServer 内のスクリプトから xxxx_permanent_xxxx() ライブラリ関数で操作するものと同じものです。

xxxx_per_net_xxxx() 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"プログラムの "デフォルト・リモートホスト" 設定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、DeviceServer 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 DeviceServer 側では、デスクトッププログラム(ABDeskTop) の "許可"ボタンで設定したホスト名リストに、xxxx_per_net_xxxx() 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 DeviceServer 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 DeviceServer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_per_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 "UpdatePermanentParams:*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_per_net_xxxx() ライブラリ関数を実行した DeviceServer 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_per_net_xxxx() 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_permanent_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。DeviceServer 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。DeviceServer 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_per_net_xxxx() ライブラリ関数 を使用する場合には DeviceServer のスクリプトプール逼迫に注意する必要があります。xxxx_per_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。

22.1 get_permanent_data(), get_per_net_data(), get_oracle_data()

- **機能概要**

  DeviceServerデータベース領域の key に対応する値を取得する。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**

  stat, value = get_permanent_data(key [,delete_flag])

  stat, value = get_per_net_data(key [,delete_flag] [,remote_host])

  stat, value = get_oracle_data(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。
value:String	データベースの key に対応する文字列
key:String	データベースのkey 文字列 (128 文字以内)
delete_flag:Boolean	データ取得後にデータベースの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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

- **使用例**

```
stat,val = get_permanent_data(“KeyName1”)
if (stat) then
  log_msg(“data = “ .. val)
end
```

22.2 set_permanent_data(), set_per_net_data(), set_oracle_data()

- **機能概要**

DeviceServerデータベース領域に 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**

stat = set_permanent_data(key, value)

stat = set_per_net_data(key, value [,remote_host])

stat = set_oracle_data(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。
key:String	データベースのKey 文字列。(128 文字以内)
value:String	データベースのKey に対応する値。(128 文字以内)
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

データベース中のデータを消去したい場合は value に空文字列 "" を指定します。

DeviceServerを再起動してもデータベース中のデータは最後に更新した値で保持されます。

- **使用例**

```
if not set_permanent_data(“KeyName1”,"新しい値") then error() end
```

22.3 clear_permanent_data(), clear_per_net_data(), clear_oracle_data()

- **機能概要**

DeviceServerデータベース領域のデータを削除する。

- **関数定義**

stat = clear_permanent_data([key_prefix])

stat = clear_per_net_data(key_prefix [,remote_host])

stat = clear_oracle_data([key_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。
key_prefix:String	削除対象のキー名（前方一致で検索する）

パラメータ省略時または "" 空文字列指定時は全レコードが削除される
clear_per_net_data() 関数の時は必須パラメータ

remote_host:String　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

- **使用例**

```
if not clear_permanent_data() then error() end
```

22.4 inc_permanent_data(), inc_per_net_data(), inc_oracle_data()

- **機能概要**

DeviceServerデータベース領域中の 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_permanent_data(key)

stat, value = inc_per_net_data(key [,remote_host])

stat, value = inc_oracle_data(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。

value:String　データベースの key に対応するカウンタ値(文字列)

key:String　データベースのkey 文字列(128 文字以内)

remote_host:String　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

データベース中にデータが見つからないか、既存のデータが数値に変換できない場合は、 "1" がデータに設定され、value にも "1" を設定します。

- **使用例**

```
stat,data = inc_permanent_data("mykey")
if not stat then error() end
log_msg(string.format("val = %s",data))
```

22.5 dec_permanent_data(), dec_per_net_data()

- **機能概要**

DeviceServerデータベース領域中の 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)になった場合にはデ

ータベースのキーに対応するデータ自身を削除する。

- **関数定義**

stat, value = dec_permanent_data(key)

stat, value = dec_per_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(128 文字以内)
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

ライブラリ関数をコールする前のキーに対応するデータの値が "1" の場合には、デクリメント後の値が "0" になるためデータベースから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 で指定したデータベース領域のデータが見つからないとき、既存のデータが数値に変換できない場合にも同様にデータは削除されます。

- **使用例**

```
stat,data = dec_permanent_data("mykey")
if not stat then error() end
log_msg(string.format("val = %s",data))
```

22.6 add_permanent_strlist(), add_per_net_strlist(), add_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域中の channel で指定した文字列リストに新しい文字列エントリを追加する。

- **関数定義**

stat = add_permanent_strlist(channel, value [,unique])

stat = add_per_net_strlist(channel, value [,unique] [,remote_host])

stat = add_oracle_strlist(channel, value [,unique])

- **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	データベース中の文字列リスト名(128 文字以内)
value:String	文字列リストに追加する文字列(128 文字以内)
unique:Boolean	true を指定すると、指定した文字列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。
remote_host:String	リモートPC 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"プログラムの設定項目 "デフ

ォルト・リモートホスト″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。

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 = add_permanent_strlist(″my_list″,”entry1”,true)
if not stat then error() end
```

22.7 clear_permanent_strlist(), clear_per_net_strlist(), clear_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域から channel_prefix で指定した文字列リストを削除する。

- **関数定義**

stat = clear_permanent_strlist(channel_prefix)

stat = clear_per_net_strlist(channel_prefix [,remote_host])

stat = clear_oracle_strlist(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_prefix:String	共有データ中の文字列リスト名(文字列) 前方一致で一致する全ての文字列リストの channel を削除する。空文字列をパラメータに指定した場合には全ての文字列リストが削除される
remote_host:String	リモートPC 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″プログラムの設定項目 ″デフォルト・リモートホスト″ に指定した値が使用される。

- **備考**

channel_prefix には削除対象の文字列リストの channel 名を指定します。指定した文字列に前方一致する channel 名を持つ全ての文字列リストが削除されます。空文字列を指定した場合には全文字列リストが削除されますので注意してください。

- **使用例**

```
stat = clear_permanent_strlist(″my_list″)
if not stat then error() end
```

22.8 get_permanent_strlist(), get_per_net_strlist(), get_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域の 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_permanent_strlist(channel [,delete])

stat, strlist = get_per_net_strlist(channel [,delete] [,remote_host])

stat, strlist = get_oracle_strlist(channel [,delete])

- **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）
channel:String	データベース中の文字列リスト名(128 文字以内)
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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

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,strlist = get_permanent_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
```

22.9 remove_permanent_strlist(), remove_per_net_strlist(), remove_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域の channel で指定した文字列リストから、value で指定した文字列に一致するエントリを削除する。

- **関数定義**

stat = remove_permanent_strlist(channel, value)

stat = remove_per_net_strlist(channel, value [,remote_host])

stat = remove_oracle_strlist(channel, 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。
channel:String	データベース中の文字列リスト名(128 文字以内)
value:String	文字列リストから削除する文字列(128 文字以内)
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

value で指定した文字列エントリが文字列リスト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_permanent_strlist("my_list","entry1")
if not stat then error() end
```

22.10 shift_permanent_strlist(), shift_per_net_strlist(), shift_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域中の channel で指定した文字列リストの、先頭の文字列エントリを取り出して削除する。

- **関数定義**

stat, value = shift_permanent_strlist(channel [,peek_only])

stat, value = shift_per_net_strlist(channel [,peek_only] [,remote_host])

stat, value = shift_oracle_strlist(channel [,peek_only])

- **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(128 文字以内)
value:String	取得した文字列
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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

文字列リストが空(文字列リストが存在しない)の場合には、value に "" 空文字列が返ります。

- **使用例**

```
stat,val = shift_permanent_strlist("my_list")
if not stat then error() end
```

22.11 pop_permanent_strlist(), pop_per_net_strlist(), pop_oracle_strlist()

- **機能概要**

DeviceServerデータベース領域中の channel で指定した文字列リストの、最後尾の文字列エントリを取り出して削除する。

- **関数定義**

stat, value = pop_permanent_strlist(channel [,peek_only])

stat, value = pop_per_net_strlist(channel [,peek_only] [,remote_host])

stat, value = pop_oracle_strlist(channel [,peek_only])

- **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(128 文字以内)
value:String	取得した文字列
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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

文字列リストが空(文字列リストが存在しない)の場合には、value に "" 空文字列が返ります。

- **使用例**

```
stat,val = pop_permanent_strlist("my_list")
```

```
if not stat then error() end
```

22.12 count_permanent_strlist(), count_per_net_strlist(), count_oracle_strlist()

- **機能概要**

  DeviceServerデータベース領域中の 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_permanent_strlist(channel)

  stat, count = count_per_net_strlist(channel [,remote_host])

  stat, count = count_oracle_strlist(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	共有データ中の文字列リスト名(128 文字以内)
count:Numberg	文字列リスト中の文字列エントリ数。 文字列エントリが空(文字列リスト自身が存在しない)の場合には 0 が返ります。
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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,count = count_permanent_strlist("my_list")
if not stat then error() end
```

22.13 list_permanent_strlist(), list_per_net_strlist()

- **機能概要**

  DeviceServerデータベース領域に登録されている文字列リストの channel名を検索する。

  (Oracle RDMS 用のこの機能に該当する API は用意されていません)

- **関数定義**

  stat, channel_list = list_permanent_strlist([channel_prefix])

  stat, channel_list = list_per_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	

データベースに登録されている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_per_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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

  データベースに登録済みの文字列リストで使用されている 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_permanent_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
```

23　セッション共有データAPI (Lua)

セッション共有データは、DeviceServer 上で管理されているキー・バリュー・データ領域です。セッション共有データは、API をコールするときに、同一のセッショントークン文字列を指定する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。グローバル共有データとは違って、同一キー名でもセッショントークン文字列が違う場合には別の領域に保存されます。

セッション共有データは、DeviceServer が再起動された場合や、該当するセッションが削除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データベース API を使用してください

23.1　get_session_data()

- **機能概要**

  DeviceServerに登録済みの有効なログインセッション共有データから、key に対応する値を取得する。指定したキーに対応するログインセッション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。指定したログインセッションが見つからない場合はエラーとなる。

- **関数定義**

stat, value = get_session_data(session_token, key [,delete_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。
value:String	ログインセッション共有データの key に対応する文字列
session_token:String	セッショントークン文字列
key:String	ログインセッション共有データのkey 文字列
delete_flag:Boolean	データ取得後にログインセッション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。

- **備考**

セッショントークン文字列は、DeviceServer のクライアントプログラムでログイン操作を行ったときに、 ログイン成功時に DeviceServer から返される値を指定してください。

- **使用例**

```
stat,val = get_session_data(“STO269B179F24241”,“KeyName1”)
if (stat) then
  log_msg(“data = “ .. val)
end
```

23.2 set_session_data()

- **機能概要**

DeviceServerに登録済みの有効なログインセッション共有データに、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。指定したログインセッションが見つからない場合はエラーとなる。

- **関数定義**

stat = set_session_data(session_token, 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。
session_token:String	セッショントークン文字列
key:String	ログインセッション共有データKey 文字列。(128 文字以内の文字列)
value:String	ログインセッション共有データのKey に対応する値。(128 文字以内の文字列)

- **備考**

ログインセッション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。

指定したログインセッションが終了した場合（ログアウトした場合）は、該当するログインセッション共有データは破棄されます。DeviceServerを再起動した場合は、すべてのログインセッション共有データは破棄されます。

セッショントークン文字列は、DeviceServer のクライアントプログラムでログイン操作を行ったときに、ログ

イン成功時に DeviceServer から返される値を指定してください。

- **使用例**

```
if not set_session_data(“ST0269B179F24241”,“KeyName1”,”新しい値”) then error() end
```

23.3 set_all_session_data()

- **機能概要**

DeviceServerに現在ログイン中の全てのログインセッション共有データに、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**

stat = set_all_session_data(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。
key:String	共有データKey 文字列。(128 文字以内の文字列)
value:String	共有データのKey に対応する値。(128 文字以内の文字列)

- **備考**

ログインセッション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。

ログインセッションが終了した場合（ログアウトした場合）は、該当するログインセッション共有データは破棄されます。DeviceServerを再起動した場合は、すべてのログインセッション共有データは破棄されます。

このライブラリ関数実行後に、ログインを行ったログインセッションに対しては、既に設定済みのログイン共有データの内容は反映されません。

- **使用例**

```
if not set_all_session_data(“KeyName1”,”新しい値”) then error() end
```

23.4 add_session_strlist()

- **機能概要**

DeviceServerに登録済みの有効なログインセッション共有データから、channel で指定した文字列リストに新しい文字列エントリを追加する。

- **関数定義**

stat = add_session_strlist(session_token, channel, value [,unique])

- **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	セッショントークン文字列
channel:String	ログインセッション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値(文字列)

unique:Boolean　　true を指定すると、指定した文字列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。

- **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。

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。

セッショントークン文字列は、DeviceServer のクライアントプログラムでログイン操作を行ったときに、ログイン成功時に DeviceServer から返される値を指定してください。

指定したログインセッションが終了した場合（ログアウトした場合）は、該当するログインセッション共有データは破棄されます。DeviceServerを再起動した場合は、すべてのログインセッション共有データは破棄されます。

- **使用例**

```
stat = add_session_strlist(“ST0269B179F24241”,"my_list",”entry1”,true)
if not stat then error() end
```

23.5　clear_session_strlist()

- **機能概要**

DeviceServerに登録済みの有効なログインセッション共有データから、channel で指定した文字列リストを削除する。

- **関数定義**

stat = clear_session_strlist(session_token, 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。

session_token:String　　セッショントークン文字列

channel:String　　ログインセッション共有データ中の文字列リスト名(文字列)

- **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。

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。

セッショントークン文字列は、DeviceServer のクライアントプログラムでログイン操作を行ったときに、ログイン成功時に DeviceServer から返される値を指定してください。

- **使用例**

```
stat = clear_session_strlist(“STO269B179F24241”,"my_list")
if not stat then error() end
```

23.6 get_session_strlist()

- **機能概要**

DeviceServerに登録済みの有効なログインセッション共有データから、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_session_strlist(session_token, channel [,delete])

- **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）
session_token:String	セッショントークン文字列
channel:String	ログインセッション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。

- **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。

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。

セッショントークン文字列は、DeviceServer のクライアントプログラムでログイン操作を行ったときに、ログイン成功時に DeviceServer から返される値を指定してください。

指定したログインセッションが終了した場合（ログアウトした場合）は、該当するログインセッション共有データは破棄されます。DeviceServerを再起動した場合は、すべてのログインセッション共有データは破棄されます。

- **使用例**

```
stat,strlist = get_session_strlist(“STO269B179F24241”,"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
```

23.7 add_all_session_strlist()

- **機能概要**

DeviceServerに現在ログイン中の指定したセッションまたは全てのログインセッション共有データから、channel で指定した文字列リストに新しい文字列エントリを追加する。

- **関数定義**

stat = add_all_session_strlist(session_list_ch, channel, value [,unique])

- **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_list_ch	対象となるセッションのセッショントークン文字列リストが格納された共有データ中の文字列リスト名。全てのセッションを対象にする場合は空文字 “” を指定する
channel:String	ログインセッション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値(文字列)
unique:Boolean	true を指定すると、指定した文字列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。

- **備考**

session_list_ch には更新対象となるセッショントークン(複数)を保存した文字列リストの名前(channel) を指定します。セッショントークンの文字列リストは共有データ中に add_shared_strlist() コマンドで作成します。ログイン中の全てのセッションを対象にする場合には、空文字列 “” を指定して下さい。

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。

ログインセッションが終了した場合（ログアウトした場合）は、該当するログインセッション共有データは破棄されます。DeviceServerを再起動した場合は、すべてのログインセッション共有データは破棄されます。

このライブラリ関数実行後に、ログインを行ったログインセッションに対しては、既に設定済みのログイン共有データの内容は反映されません。

- **使用例**

```
if not set_all_session_data("my_list","entry1",true) then error() end
```

23.8 clear_all_session_strlist()

- **機能概要**

  DeviceServerに現在ログイン中の指定したセッションまたは全てのログインセッション共有データから、channel で指定した文字列リストを削除する。

- **関数定義**

  stat = clear_all_session_strlist(session_list_ch, 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。
session_list_ch	対象となるセッションのセッショントークン文字列リストが格納された共有データ中の文字列リスト名。全てのセッションを対象にする場合は空文字 “” を指定する
channel:String	ログインセッション共有データ中の文字列リスト名(文字列)

- **備考**

  session_list_ch には更新対象となるセッショントークン(複数)を保存した文字列リストの名前(channel) を指定します。セッショントークンの文字列リストは共有データ中に add_shared_strlist() コマンドで作成します。ログイン中の全てのセッションを対象にする場合には、空文字列 “” を指定して下さい。

  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**

```
if not clear_all_session_data("my_list") then error() end
```

24 統計データベースAPI (Lua)

- 統計データベース領域について

DeviceServer のデータベース (Firebird 又は Oracle RDMS) にセンサーからの計測値などを登録して、タイムスタンプを元に検索して取り出すことができます。集計間隔を指定して、一定期間毎のサマリを計算して出力することもできます。登録データにはデータ列で共通なキー値を指定することで、任意のデータ列を複数作成することが出来ます。

統計データベースには１つのキーに対して対応する１つの数値(整数や実数) を時系列に登録することができます。構造化されたデータや文字列を時系列に保存するときには、"時系列文字列データAPI" の章を参照してください。

DeviceServer 内蔵の Firebird データベースまたは、Oracle RDBMS を操作するための API です。ほぼ共通の機能をもつ両方の RDMS に対応する関数が DeviceServer から使用できます。

Oracle RDBMS 用のAPI 関数を使用する場合には、予めOracle接続機能の設定を行う必要があります。”Oracle接続機能”の章を参照してください。

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。

- リモートデータベースライブラリ関数 xxxx_stat_net_xxxx() について

ライブラリ関数名が xxxx_per_net_xxxx() のものは、リモート側の DeviceServer の Firebird データベース領域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データベース領域のデータはリモート側の DeviceServer 内のスクリプトから xxxx_stat_data() ライブラリ関数で操作するものと同じものです。Oracleデータベース領域へ保存するためのリモート側操作関数は用意されていません。

xxxx_stat_net_xxxx() 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"プログラムの "デフォルト・リモートホスト" 設定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、DeviceServer 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 DeviceServer 側では、デスクトッププログラム(ABDeskTop) の "許可"ボタンで設定したホスト名リストに、xxxx_stat_net_xxxx() 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 DeviceServer 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 DeviceServer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 "add_stat_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 "AddStatData:*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_stat_net_xxxx() ライブラリ関数を実行した DeviceServer 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_stat_net_xxxx() 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

"add_stat_net_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。DeviceServer 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。DeviceServer 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_stat_net_xxxx() ライブラリ関数 を使用する場合には DeviceServer のスクリプトプール逼迫に注意する必要があります。xxxx_stat_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。

24.1 add_stat_data(), add_stat_net_data(), add_stat_oracle()

- **機能概要**

  統計用データベースに、key とタイムスタンプ値に対応する値 value のデータレコードを登録する。

- **関数定義**

  stat = add_stat_data(key, value [,timestamp])

  stat = add_stat_net_data(key, value, timestamp [,remote_host])

  stat = add_stat_oracle(key, value [,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	データベースのKey 文字列(128 文字以内)
value:Number	データベースのKey に対応する値。(Double 値)
timestamp:String	データのタイムスタンプ値 'YYYY/MM/DD HH:MM:SS' 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

  add_stat_data() ライブラリ関数を使用する時に timestamp パラメータを指定する場合には、秒単位の精度で

時系列のデータレコードを作成します。timestamp パラメータを省略する場合には、CPU のシステム時刻を元に 0.001 秒の精度でデータレコードを作成します。登録済みのデータレコードを、search_stat_data() 関数を使用して個々に取り出すようにして使用する場合には、ソート順を正確にするために timestamp パラメータを省略して登録してください。summary_stat_data() 関数を使用して集計計算を行う場合には、timestamp パラメータの指定・省略による影響はありません。

add_stat_net_data() を使用する場合には、remote_host 以外のパラメータを省略することはできません。この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

DeviceServerを再起動しても統計用データベース中のデータレコードは保持されます。

- **使用例**

```
if not add_stat_data(“KeyName1”,1.234) then error() end
```

24.2 clear_stat_data(), clear_stat_net_data(), clear_stat_oracle()

- **機能概要**

統計用データベースから指定したデータレコードを削除する。

- **関数定義**

stat = clear_stat_data([key_prefix [,timestamp_to [,timestamp_from]]])

stat = clear_stat_net_data(key_prefix ,timestamp_to ,timestamp_from [,remote_host])

stat = clear_stat_oracle([key_prefix [,timestamp_to [,timestamp_from]]])

- **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_prefix:String	削除対象のキー名（前方一致で検索する） パラメータ省略時は全レコードが削除される
timestamp_to:String	削除対象のタイムスタンプ上限値 指定した日付時刻以前のレコードを削除する。 ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する パラメータ省略時はkey_prefix に一致する全レコードが削除される
timestamp_from:String	削除対象のタイムスタンプ下限値 timestamp_to の指定と組み合わせて使用して、日付時刻範囲内のレコードを削除する ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する パラメータ省略時はkey_prefix と timestamp_to に一致するレコードが削除される
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

  指定した条件に一致する統計データレコードを全て削除します。

  パラメータを全て省略した場合は、DeviceServer に登録済みの全統計データが削除されます。

  clear_stat_net_data() を使用する場合には、remote_host 以外のパラメータを省略することはできません。

  この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

- **使用例**

```
if not clear_stat_data("KEY1", "2001/12/31 23:59:59","2001/01/01 0:0:0") then error() end
```

24.3 summary_stat_data(), summary_stat_net_data(), summary_stat_oracle()

- **機能概要**

  統計データレコードの集計を行う。

- **関数定義**

```
stat, datetime, sample, total, mean, max, min = summary_stat_data(key_prefix, datetime_start, interval,
                count)
stat, datetime, sample, total, mean, max, min = summary_stat_net_data(key_prefix, datetime_start,
                interval, count [,remote_host])
stat, datetime, sample, total, mean, max, min = summary_stat_oracle(key_prefix, 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統計データ個数(整数値)
total:Table [1..#count] of Number	集計期間中に見つかった統計データの合計値(Double値)
mean:Table [1..#count] of Number	集計期間中に見つかった統計データの平均値(Double値)
max:Table [1..#count] of Number	集計期間中に見つかった統計データの最大値(Double値)
min:Table [1..#count] of Number	集計期間中に見つかった統計データの最小値(Double値)
key_prefix:String	集計対象のキー名（前方一致で検索する） 省略時は、後のパラメータで指定された期間に一致する全レコードが対象となる
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）
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **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。

datetime_start	“2001/01/01 0:0:0”
interval	3600
count	24

sample[n] が 0 の場合は、total[n], mean[n], max[n], min[n] の値は強制的に 0 になり、統計値の意味を持たない。

- **使用例**

```
stat,datetime,sample,total,mean,max,min = summary_stat_data(“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
```

24.4 search_stat_data(), search_stat_net_data(),

- **機能概要**

統計データレコードを検索する。

(Oracle RDMS 用のこの機能に該当する API は用意されていません)

- **関数定義**

```
stat, key, value, timestamp =
                search_stat_data(key_prefix, sort_ascend, datetime_from, datetime_to,
                 datetime_until [,rows [,rows_to]])
stat, key, value, timestamp =
                search_stat_net_data(key_prefix, sort_ascend, datetime_from, datetime_to,
                 datetime_until [,rows [,rows_to]] [,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:Table [1..#count] of String	検索で見つかったデータのキー名
value:Table [1..#count] of Number	検索で見つかったデータの値
timestamp: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
key_prefix:String	検索対象のキー名（前方一致で検索する） 省略時は、全てのレコードが検索対象となる
sort_ascend:Boolean	データタのタイムスタンプが昇順になるように検索結果をソートする場合には

	True を指定する。(タイムスタンプ値が古いものから順にソートされる) False を指定した場合には降順に検索結果がソートされる
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または、開始日付時刻を定する。パラメータを使用しない場合には空文字列 “” を指定する。 開始日付を指定する場合には、‘YYYY/MM/DD’ の形式で指定する 開始日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
datetime_to: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または、終了日付時刻を定する。指定した日付または日付時刻を含むデータが検索対象となる datetime_to パラメータは datatime_until パラメータと同時に使用することはできません。パラメータを使用しない場合には空文字列 “” を指定する。 終了日付を指定する場合には、‘YYYY/MM/DD’ の形式で指定する 終了日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
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 datetime_until パラメータは datatime_to パラメータと同時に使用することはできません。パラメータを使用しない場合には空文字列 “” を指定する。 終了日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
rows:Number	検索結果の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検索条件に一致する全てのレコードがリターン値に返ります。ただし、検索結果として取得可能なレコード数は 最大 5000 までです。これを超えた場合には、stat には false が返って、リターン値には 5000 レコード分のデータが格納されます。 rows パラメータを指定して rows_to パラメータを省略した場合には、rows パラメータは検索結果の総レコード数を指定することになります。このとき sort_ascend パラメータによって昇順または降順に並べられた最初のレコードから rows で指定したレコード番号までのデータをリターン値に返します。レコード番号は 1 から始まる点に注意して下さい。 rows パラメータと rows_to パラメータの両方を指定した場合に rows パラメータは、検索結果のデータレコード中からリターン値に返す最初のレコード番号を指定することになります。
rows_to:Number	rows パラメータと同時に使用して、検索結果に取得する最後のレコード番号を指定する。たとえば rows に 10 を指定して rows_to に 20 を指定すると、検索結果のレコード番号が 10 から 20 までのデータがリターン値に返ります。検索結果として取得可能なレコード数は 最大 5000 までです。これを超えた場合には、stat には false が返って、リターン値には 5000 レコード分のデータが格納されます。 rows パラメータを指定しない場合には、rows_to パラメータを指定することはできません。
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

例1：2001/1/1 の データを全て取得する場合には、パラメータを下記の様に指定する。データは昇順で取得する

key_prefix	“”
sort_ascend	true
datetime_from	“2001/01/01”
datetime_to	“2001/01/01”
datetime_until	“”

例2：キー名が “TEST” で始まる直近のデータを 100 レコード分取得する。ただし、現在時刻は (2012/11/21 10:10:20) とする

key_prefix	“TEST”
sort_ascend	false
datetime_from	“”
datetime_to	“”
datetime_until	“2012/11/21 10:10:20”
rows	100

例3：キー名が “TEST” で始まるデータで、2001/1/1 から、2001/1/2 までのデータを 10 レコードづつ複数回にわたって全て取得する。

key_prefix	“TEST”
sort_ascend	true
datetime_from	“2001/01/01”
datetime_to	“2001/01/02”
datetime_until	“”
rows	1 (続けてコールする時に、11, 21, 31 ...の順に指定する)
rows_to	10(続けてコールする時に、20, 30, 40 ...の順に指定する)

sort_ascend = false を指定する場合の注意

add_stat_data() ライブラリ関数でデータを登録した時に timestamp パラメータを指定した場合には、ソートの精度は秒単位になります。timestamp パラメータを省略していた場合には、内部でミリ秒単位までタイムスタンプが記録されていますので、0.001 秒の精度でソートが可能です。同じキーに対するデータを 0.001秒よりも短い間隔で複数登録した場合には、降順(sort_ascend = false)ソート時にデータの順序が前後することがあります。

- **使用例**

```
stat,key,value,timestamp = search_stat_data(“KEY”,true,”2001/01/01 0:0:0”,””,””)
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
```

24.5 key_list_stat_data(), key_list_stat_net_data()

- **機能概要**

  統計データベース中のデータレコードで使用しているキー名を検索する。

  (Oracle RDMS 用のこの機能に該当する API は用意されていません)

- **関数定義**

  stat, key_list = key_list_stat_data([key_prefix])

  stat, key_list = key_list_stat_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

  　　統計データベースに登録されているデータレコードで使用しているキー名リスト

  key_prefix:String　　検索対象のキー名（前方一致で検索する）

  　　省略時は全てのレコードが検索対象となる

  remote_host:String　　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

  統計データベースに登録済みのレコードで使用されているキー名リストを取得する。

  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キー名のみが検索対象になります。

  key_list_stat_net_data() を使用する場合には、remote_host 以外のパラメータを省略することはできません。この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

- **使用例**

```
stat,key_list = key_list_stat_data("SENSOR")
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
```

25　時系列文字列データAPI (Lua)

- 時系列データベース領域について

DeviceServer のデータベース（Firebird）に任意の文字列を登録して、タイムスタンプを元に検索して取り出すことができます。データベースに登録する文字列データには、JSON 形式の構造化された複数のセンサーデータを格納することもできます。

この API では任意のキー文字列に対して、値となる文字列値を任意の数だけ登録することができます。データ登録時にタイムスタンプが記録されますので、後から検索期間やキー文字列を指定してデータを取り出すことが簡単にできます。検索時に、取り出すデータのソート（昇順・降順）の指定ができますのでセンサー値などを時系列に処理することが簡単にできます。

一つのキーに対して数値(整数、実数)を１つだけ時系列に保存したい場合には、集計機能などが利用できる "統計データベースAPI" の章を参照してください。

DeviceServer 内蔵の Firebird データベースを使用していますので、インストール直後から時系列文字列データ API を利用することができます。

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できます。

- リモート時系列データベースライブラリ関数 xxxx_series_net_xxxx() について

ライブラリ関数名が xxxx_series_net_xxxx() のものは、リモート側の DeviceServer のデータベース領域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データベース領域のデータはリモート側の DeviceServer 内のスクリプトから xxxx_series_xxxx() ライブラリ関数で操作するものと同じものです。

xxxx_series_net_xxxx() 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"プログラムの "デフォルト・リモートホスト" 設定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、DeviceServer 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 DeviceServer 側では、デスクトッププログラム(ABDeskTop) の "許可"ボタンで設定したホスト名リストに、xxxx_series_net_xxxx() 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 DeviceServer 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 DeviceServer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 "add_series_net_str:*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 "AddSeriesString:*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_series_net_xxxx() ライブラリ関数を実行した DeviceServer 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_series_net_xxxx() 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 "add_series_net_str:*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。DeviceServer 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。DeviceServer 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_series_net_xxxx() ライブラリ関数 を使用する場合には DeviceServer のスクリプトプール逼迫に注意する必要があります。xxxx_series_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。

25.1 add_series_str(), add_series_net_str(),

- **機能概要**

  時系列文字列データベースに、key とタイムスタンプ値に対応する値 value のデータレコードを登録する。

- **関数定義**

  stat = add_series_str(key, value [,timestamp])

  stat = add_series_net_str(key, value, timestamp [,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	データベースのKey 文字列(128 文字以内)
value:String	Key に対応するデータ文字列(1000 文字以内)
timestamp:String	データのタイムスタンプ値 'YYYY/MM/DD HH:MM:SS' 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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。

add_series_str() ライブラリ関数を使用する時に timestamp パラメータを指定する場合には、秒単位の精度で時系列のデータレコードを作成します。timestamp パラメータを省略する場合には、CPU のシステム時刻を元に 0.001 秒の精度でデータレコードを作成します。登録済みのデータレコードを、search_series_str() 関数を使用して個々に取り出すようにして使用する場合には、ソート順を正確にするために timestamp パラメータを省略して登録してください。

add_series_net_str() を使用する場合には、remote_host 以外のパラメータを省略することはできません。この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

DeviceServerを再起動しても時系列データベース中のデータレコードは保持されます。

- **使用例**

```
if not add_series_str('MySensorData1',
'[{"id":"SENSOR_TP_Device4","current_value": "11"},{"id":"SENSOR_IR_Device4","current_value":
"0"}]') then error() end
```

25.2 clear_series_str(), clear_series_net_str()

- **機能概要**

時系列文字列データベースから指定したデータレコードを削除する。

- **関数定義**

stat = clear_series_str([key_prefix [,timestamp_to [,timestamp_from]]])

stat = clear_series_net_str(key_prefix ,timestamp_to ,timestamp_from [,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_prefix:String	削除対象のキー名（前方一致で検索する） パラメータ省略時は全レコードが削除される
timestamp_to:String	削除対象のタイムスタンプ上限値

指定した日付時刻以前のレコードを削除する。
'YYYY/MM/DD HH:MM:SS' の形式で指定する
パラメータ省略時はkey_prefix に一致する全レコードが削除される

timestamp_from:String　削除対象のタイムスタンプ下限値
timestamp_to の指定と組み合わせて使用して、日付時刻範囲内のレコードを削除する
'YYYY/MM/DD HH:MM:SS' の形式で指定する
パラメータ省略時はkey_prefix と timestamp_to に一致するレコードが削除される

remote_host:String　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

指定した条件に一致する時系列文字列データレコードを全て削除します。
パラメータを全て省略した場合は、DeviceServer に登録済みの全時系列文字列データが削除されます。

clear_series_net_str() を使用する場合には、remote_host 以外のパラメータを省略することはできません。
この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

- **使用例**

```
if not clear_series_str("MySensorData1","2001/12/31 23:59:59","2001/01/01 0:0:0") then error() end
```

25.3 search_series_str(), search_series_net_str()

- **機能概要**

時系列文字列データレコードを検索する。

- **関数定義**

```
stat, key, value, timestamp =
                search_series_str(key_prefix, sort_ascend, datetime_from, datetime_to,
                 datetime_until [,rows [,rows_to]])
stat, key, value, timestamp =
                search_series_net_str(key_prefix, sort_ascend, datetime_from, datetime_to,
                 datetime_until [,rows [,rows_to]] [,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:Table [1..#count] of String　検索で見つかったデータのキー名
value:Table [1..#count] of String　検索で見つかった文字列データ

timestamp: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
key_prefix:String	検索対象のキー名（前方一致で検索する） 省略時は、全てのレコードが検索対象となる
sort_ascend:Boolean	データのタイムスタンプが昇順になるように検索結果をソートする場合には True を指定する。(タイムスタンプ値が古いものから順にソートされる) False を指定した場合には降順に検索結果がソートされる
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または、開始日付時刻を定する。パラメータを使用しない場合には空文字列 “” を指定する。 開始日付を指定する場合には、‘YYYY/MM/DD’ の形式で指定する 開始日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
datetime_to: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または、終了日付時刻を定する。指定した日付または日付時刻を含むデータが検索対象となる datetime_to パラメータは datatime_until パラメータと同時に使用することはできません。パラメータを使用しない場合には空文字列 “” を指定する。 終了日付を指定する場合には、‘YYYY/MM/DD’ の形式で指定する 終了日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
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 datetime_until パラメータは datatime_to パラメータと同時に使用することはできません。パラメータを使用しない場合には空文字列 “” を指定する。 終了日付時刻を指定する場合には ‘YYYY/MM/DD HH:MM:SS’ の形式で指定する
rows:Number	検索結果の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検索条件に一致する全てのレコードがリターン値に返ります。ただし、検索結果として取得可能なレコード数は 最大 5000 までです。これを超えた場合には、stat には false が返って、リターン値には 5000 レコード分のデータが格納されます。 rows パラメータを指定して rows_to パラメータを省略した場合には、rows パラメータは検索結果の総レコード数を指定することになります。このとき sort_ascend パラメータによって昇順または降順に並べられた最初のレコードから rows で指定したレコード番号までのデータをリターン値に返します。レコード番号は 1 から始まる点に注意して下さい。 rows パラメータと rows_to パラメータの両方を指定した場合に rows パラメータは、検索結果のデータレコード中からリターン値に返す最初のレコード番号を指定することになります。
rows_to:Number	rows パラメータと同時に使用して、検索結果に取得する最後のレコード番号を指定する。たとえば rows に 10 を指定して rows_to に 20 を指定すると、検索結果のレコード番号が 10 から 20 までのデータがリターン値に返ります。検索結果として取得可能なレコード数は 最大 5000 までです。これを超えた場合には、stat には false が返って、リターン値には 5000 レコード分のデータが格納されます。

rows パラメータを指定しない場合には、rows_to パラメータを指定することはできません。

remote_host:String　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

例1：2001/1/1 の データを全て取得する場合には、パラメータを下記の様に指定する。データは昇順で取得する

key_prefix	“”
sort_ascend	true
datetime_from	“2001/01/01”
datetime_to	“2001/01/01”
datetime_until	“”

例2：キー名が “TEST” で始まる直近のデータを 100 レコード分取得する。ただし、現在時刻は (2012/11/21 10:10:20) とする

key_prefix	“TEST”
sort_ascend	false
datetime_from	“”
datetime_to	“”
datetime_until	“2012/11/21 10:10:20”
rows	100

例3：キー名が “TEST” で始まるデータで、2001/1/1 から、2001/1/2 までのデータを 10 レコードづつ複数回にわたって全て取得する。

key_prefix	“TEST”
sort_ascend	true
datetime_from	“2001/01/01”
datetime_to	“2001/01/02”
datetime_until	“”
rows	1 (続けてコールする時に、11, 21, 31 ...の順に指定する)
rows_to	10(続けてコールする時に、20, 30, 40 ...の順に指定する)

sort_ascend = false を指定する場合の注意

add_series_str() ライブラリ関数でデータを登録した時に timestamp パラメータを指定した場合には、ソートの精度は秒単位になります。timestamp パラメータを省略していた場合には、内部でミリ秒単位までタイムスタンプが記録されていますので、0.001 秒の精度でソートが可能です。同じキーに対するデータを 0.001秒よりも短い間隔で複

数登録した場合には、降順(sort_ascend = false)ソート時にデータの順序が前後することがあります。

- **使用例**

```
stat,key,value,timestamp = search_series_str(“MySensor1”,true,”2001/01/01 0:0:0”,””,””)
if not stat then error() end
for key,val in ipairs(value) do
  log_msg(val,file_id)
end
```

25.4 key_list_series_str(), key_list_series_net_str()

- **機能概要**

時系列文字列データベース中のデータレコードで使用しているキー名を検索する。

- **関数定義**

stat, key_list = key_list_series_str([key_prefix])

stat, key_list = key_list_series_net_str(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

　　時系列文字列データベースに登録されているデータレコードで使用しているキー名リスト

key_prefix:String　　検索対象のキー名（前方一致で検索する）

　　省略時は全てのレコードが検索対象となる

remote_host:String　　リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

時系列文字列データベースに登録済みのレコードで使用されているキー名リストを取得する。

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キー名のみが検索対象になります。

key_list_series_net_data() を使用する場合には、remote_host 以外のパラメータを省略することはできません。この場合でも、パラメータに""空文字列を指定することで省略時と同様の動作させることができます。

- **使用例**

```
stat,key_list = key_list_series_str("SENSOR")
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
```

25.5 count_series_str(), count_series_net_str()

- **機能概要**

  DeviceServerデータベース領域中の 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_series_str(key)

  stat, count = count_series_net_str(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。
key:String	データベースのKey 文字列(128 文字以内) ここで指定する key文字列 は完全一致で検索します
count:Numberg	時系列文字列データベース中のキーに一致するデータレコード数。
remote_host:String	リモートPC 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"プログラムの設定項目 "デフォルト・リモートホスト" に指定した値が使用される。

- **備考**

- **使用例**

```
stat,count = count_series_str("MySensor1")
if not stat then error() end
```

26 スクリプト実行API (Lua)

スクリプト実行ライブラリ関数は、DeviceServer に設定した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することができます。任意のスクリプトパラメータを起動時に指定できます。

script_exec2(),script_rexec2() ライブラリ関数を使用すると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ができます。

ライブラリ関数を使用して起動したスクリプト中からも、再びこのライブラリ関数を使用してスクリプトをコールすることもできます。このときスクリプトプールのエントリを複数同時に使用して実行しています。

script_rexec(), script_rexec2(), script_fork_rexec() ライブラリ関数は、別の CPU で動作している DviceServer のスクリプトを実行できます。script_net_list() ライブラリ関数は、リモート側のスクリプトファイ

ルリストを取得できます。これらのライブラリ関数を使用する場合にはリモート側の DeviceServer 側で、デスクトッププログラム(ABDeskTop) の "許可"ボタンで設定したホスト名リストに、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 DeviceServer 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 DeviceServer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 DeviceServer 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()ライブラリ関数などの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 "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、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。DeviceServer 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。DeviceServer 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、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 DeviceServer 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。

26.1 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 “” を渡すこと。

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 “” を渡すこと。

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。

- **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。

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は "サーバー設定"プログラムの "同時実行可能な最大スクリプト数" で設定されています。（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
```

26.2　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 "" を渡すこと。
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 "" を渡すこと。
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。

- **備考**

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 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。

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は "サーバー設定"プログラムの "同時実行可能な最大スクリプト数" で設定されています。（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
```

26.3 script_rexec()

- **機能概要**

リモートPC の DeviceServer 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PC の IP アドレスまたはホスト名

"" 空文字列を指定した場合には、"サーバー設定"プログラムの設定項目 "デフォルト・リモートホスト" 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 "" を渡すこと。

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 "" を渡すこと。

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。

- **備考**

リモート PC で動作している DeviceServer 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PC で動作している DeviceServer のマスタ(HostsEquiv)には、呼び出し元の PC 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、デスクトッププログラム (ABDeskTop) の "許可" ツールボタンで設定できます。

name パラメータで指定するスクリプト名は、リモートPC側の DeviceServer 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。

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は "サーバー設定"プログラムの "同時実行可能な最大スクリプト数" で設定されています。（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
```

26.4 script_rexec2()

- **機能概要**

リモートPC のDeviceServer 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PC の IP アドレスまたはホスト名 "" 空文字列を指定した場合には、"サーバー設定"プログラムの設定項目 "デフォルト・リモートホスト" 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 "" を渡すこと。
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 "" を渡すこと。
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。

- **備考**

リモート PC で動作している DeviceServer 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PC で動作している DeviceServer のマスタ(HostsEquiv)には、呼び出し元の PC 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、デスクトッププログラム (ABDeskTop) の "許可" ツールボタンで設定できます。

name パラメータで指定するスクリプト名は、リモートPC側の DeviceServer 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 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。

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は "サーバー設定"プログラムの "同時実行可能な最大スクリプト数" で設定されています。（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
```

26.5 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（必ずグローバル変数 g_taskidを指定する）
key:String	リターン値のキー名。(128 文字以内の文字列)
value:String	リターン値のキーに対応する値。(128 文字以内の文字列)

- **備考**

パラメータエラーや、共有データの更新に失敗した場合に、エラーが発生するまでにスクリプト中で script_result() 関数でセットした値が、DeviceServer の共有データに残る場合があります。

- **使用例**

```
if not script_result(g_taskid,"key1","val1") then error() end
if not script_result(g_taskid,"リターン#2","値#2") then error() end
```

26.6 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 “” を渡すこと。

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 “” を渡すこと。

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。

- **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）

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は ″サーバー設定″プログラムの ″同時実行可能な最大スクリプト数″ で設定されています。（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
```

26.7　script_fork_rexec()

- **機能概要**

リモートPC のDeviceServer 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PC の IP アドレスまたはホスト名

"" 空文字列を指定した場合には、"サーバー設定"プログラムの設定項目 "デフォルト・リモートホスト" 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 "" を渡すこと。

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 "" を渡すこと。

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。

- **備考**

リモート PC で動作している DeviceServer 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PC で動作している DeviceServer のマスタ(HostsEquiv)には、呼び出し元の PC 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、デスクトッププログラム (ABDeskTop) の "許可" ツールボタンで設定できます。

name パラメータで指定するスクリプト名は、リモートPC側の DeviceServer 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）

DeviceServer で同時実行可能なスクリプトの数には制限があります。この値は "サーバー設定"プログラムの "同時実行可能な最大スクリプト数" で設定されています。（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
```

26.8 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
```

26.9 script_task_list()

- **機能概要**

  現在 DeviceServer 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
```

26.10 script_file_list(), script_net_list()

- **機能概要**

DeviceServer 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PC 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"プログラムの設定項目 "デフォルト・リモートホスト" 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)

DeviceServer の "C:¥Program Files¥AllBlueSystem¥Scripts" フォルダ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、
“C:¥Program Files¥AllBlueSystem¥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
```

26.11 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使用可能な、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。DeviceServer 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。このときログには以下の様なメッセージが記録されます。

```
2014/11/05 21:19:24 eagle      LUASessionPool      0 SelectFreeLUA: could not find the free index.
```

このときに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が再開されます。

DeviceServer 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、"サーバー設定プログラム"中のパフォーマンスタブ内にある、"同時実行可能なスクリプト数" で設定されています。この値はスタンダードライセンスの場合には 16 で固定ですが、エンハンスライセンスの場合には任意の値に設定できます。

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**

```
file_id = "RELAY_SERVER_UPLINK"
--[[
●機能概要
リレーサーバーに配信用のデータをアップロードする
●スクリプト中の変数に初期設定が必要な項目
--------------------------------------------------------------------------------
変数名                          値                                  値の例
--------------------------------------------------------------------------------
host    リレーサーバーが動作している PC のホスト名          "localhost"
        リレーサーバーは node.js プログラムで起動されたrelay_server.js ファイルで実行されます。
port    リレーサーバーのポート番号                          9080
●リクエストパラメータ
--------------------------------------------------------------------------------
キー値                          値                                  値の例
--------------------------------------------------------------------------------
<RelayServerParamKey#1>     <RelayServerParamValue#1>     "ADO" "-1"
```

```
..
..
<RelayServerParamKey#n>      <RelayServerParamValue#n>
リレーサーバーで配信するメッセージ中に格納するデータ値を任意の数だけ指定できる。
このスクリプトを起動する時に指定した全てのパラメータ(キー値と値)はそのまま
リレーサーバーに渡されて、その後 WebSocketで接続している全てのクライアントに配信される。
●リターンパラメータ
-------------------------------------------------------------------------------
キー値                          値                              値の例
-------------------------------------------------------------------------------
●備考
スクリプト中で実行している tcp_send_recv_data() 関数は、相手側のサーバーがダウン
しているときにネットワークエラー検出までに時間がかかることが想定されます。
このとき、イベントハンドラなど別スレッドで同時にこのスクリプトを続けて起動すると、
スクリプトプールの未使用エントリ数がなくなり、システム全体の動作に影響する可能性
があります。これを防ぐために、スクリプトプールの未使用エントリ数が一定以下になっ
た場合には、このスクリプトの実行をすぐに中止させるようにしています

中止させるためのフラグは abort_key のキー名で設定されたグローバル共有変数を使用
しています。一度このフラグが設定されるとこのスクリプトは直ぐに終了して、ログに
*ABORT* が記録されるようになります。再びこのスクリプトを実行可能にするためには、
DeviceServerを再起動させるか abort_key に設定されたグローバル共有変数を削除して
ください。このとき PERIODIC_TIMER.lua 中に dec_shared_data() 関数を使用すると
簡単に一定期間後にグローバル共有変数を削除することができます。このスクリプトでは
中止フラグの値に "60" を設定していますので、下記のスクリプトをPERIODIC_TIMER.lua 中
に記述すると、60分後に中止フラグが自動的に削除されます。

dec_shared_data("$ABORT_RELAY_SERVER_UPLINK")
●変更履歴
2014/11/8          初版作成
ABS-9000 DeviceServer        copyright(c) All Blue System
]]
local host = "localhost"
local port = 9080
-------------------------------------------------------------------------------------------------------
-- リレーサーバー接続が失敗する場合にスクリプトプールの逼迫を防ぐため、スクリプト実行中止の判断をする
-------------------------------------------------------------------------------------------------------
local abort_key = "$ABORT_" .. file_id
local stat, val = get_shared_data(abort_key)
```

```
if val ~= "" then
        log_msg("*ABORT*",file_id)
        return
end
if script_free_count() < 5 then
        if not set_shared_data(abort_key,"60") then error() end
        log_msg("*ABORT SET*",file_id)
        return
end
-----------------------------------------------------------------------------
-- スクリプトパラメータで渡されたキーと値を JSON 文字列に変換する
-----------------------------------------------------------------------------
local json_str = '{'
local first = true
for key,val in pairs(g_params) do
        if first then
                first = false
        else
                json_str = json_str .. ","
        end
        json_str = json_str .. string.format('"%s":"%s"',key,val)
end
json_str = json_str .. '}'
-----------------------------------------
-- リレーサーバーにメッセージを送信する
-----------------------------------------
log_msg(json_str,file_id)
local nstat,rdata = tcp_send_recv_data(host,port,json_str,2,2)
```

27 XBee API (Lua)

XBEE サービスモジュールで管理している XBee 802.15.4 (Series1)デバイスを操作するためのライブラリ関数です。API で操作するXBee デバイスは、予め “XBee デバイス管理”プログラムでマスターに登録して下さい。

27.1 xbee_all_list()

- **機能概要**

  “XBee デバイス管理プログラム” でマスターに登録された、全ての XBee デバイスリストを取得する。

- **関数定義**

stat, serial, addr16, name, localdevice = xbee_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。

serial:Table [1..#max_device] of String　　XBeeデバイスのSerial Number(64bitアドレス)16進数表記の文字列

addr16:Table [1..#max_device] of String　　XBeeデバイスのSourceAddress(16bitアドレス)16進数表記の文字列

name:Table [1..#max_device] of String　　XBeeデバイスの Node Identification 文字列

localdevice:Table [1..#max_device] of Boolean　　XBeeデバイスが DeviceServer 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**

xbee_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 デバイス管理プログラム” の “探索＆登録” ツールボタンを使用してください。

- **使用例**

```
stat,serial,addr16,name,localdevice = xbee_all_list()
if not stat then error() end
for key,val in ipairs(serial) do
  log_msg(string.format("device[%d] serial = %s addr16 = %s name = %s is_local
              = %s",key,val,addr16[key],name[key],tostring(localdevice[key])),file_id)
end
```

27.2　xbee_transmit_data()

- **機能概要**

指定したXBee デバイスにデータを送信する。

- **関数定義**

stat = xbee_transmit_data(device, rf_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。

device:String　　送信先の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デバイスの Node Identification 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

XBeeデバイスの SourceAddress(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

rf_data:String　　送信するデータ。16進数表記の文字列で指定する。

- **備考**

rf_data に指定可能なデータ数は、XBee デバイスの仕様によって100 バイトまでに制限されます。(ファーム

ウエアバージョン “10CD”)

device に、DeviceServer の COM ポートに接続したXBeeデバイス(送信元)を指定すると、XBee デバイスでパケットを受信することができませんので注意してください。(ファームウエアバージョン “10CD”)

- **使用例**

```
stat = xbee_transmit_data("Device2",list_to_hex(unpack(str_to_tbl("Hello World!"))))
if not stat then error() end
```

27.3 xbee_dio()

- **機能概要**

指定したXBeeデバイスの DIO コンフィギュレーションを DO Low または DO High に設定する。

- **関数定義**

stat = xbee_dio(device, bit, 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。
device:String	送信先の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デバイスの Node Identification 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
	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、DeviceServer に接続したXBeeデバイスが対象
bit:Number	XBeeデバイスのDIO ビット番号を指定する。(0から7までの整数)
flag:boolean	true を指定すると、DO High に設定され、false を指定すると DO Low に設定する。

- **備考**

device がリモート XBee デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。またコマンドオプションは 0x02(Apply Change on remote) になります。

- **使用例**

```
stat = xbee_dio("Device1",0,false)
if not stat then error() end
```

27.4 xbee_duty()

- **機能概要**

指定したXBeeデバイスの PWM Output Level を設定する。

- **関数定義**

stat = xbee_duty(device, channel, duty)

- **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。
device:String	送信先の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。
	Beeデバイスの Node Identification 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
	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、DeviceServer に接続したXBeeデバイスが対象
channel:Number	XBeeデバイスのPWM チャンネル番号を指定する。(0 または 1)
duty:Number	PWM Output レベルを 0 から 1023 の間の整数で指定する。

- **備考**

PWM 出力を行うためには、予め XBee デバイスの PWM Configuration の値を 2 (PWM Output) に設定する必要があります。“XBee デバイス管理プログラム” から PWM設定を操作することができます。

device がリモート XBee デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。またコマンドオプションは 0x02(Apply Change on remote) になります。

- **使用例**

```
stat = xbee_duty("Device1",1,1023)
if not stat then error() end
```

27.5 xbee_force_sample()

- **機能概要**

指定したXBeeデバイスの有効な入力データを全て取得する。

- **関数定義**

stat, adc, dio = xbee_force_sample(device)

- **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。
device:String	送信先の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デバイスの Node Identification 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
	XBeeデバイスの Source Address(16bitアドレス)16進数表記の文字列
	“” 空文字列は、DeviceServer に接続したXBeeデバイスが対象(備考参照の事)
adc:Table	XBeeデバイスのA/D 変換入力値。
adc[“<bit_number”]:String	0から1023までの整数値。文字列で入る
	<bit_number> にはA/D 変換を行ったDIO ビット番号が文字列で入る。
dio:Table	XBeeデバイスのDI 入力値。
dio[“<bit_number”]:String	”1” で High, “0” で Low を示す。
	<bit_number> にはDI入力対象の DIO ビット番号が文字列で入る。

- **備考**

XBee デバイスの DIO 設定で、少なくとも一つ以上の DIO が 入力(DIN または ADC)モードになっている必要があります。”XBee デバイス管理プログラム” から DIOや ADC 設定を操作することができます。

device に DeviceServer の COM ポートに接続した XBee デバイスを指定することもできますが、この場合にはリターン値(adc, dio) に サンプル値は設定されません。その代わりに I/Oイベント(XBEE_IO_DATA) が発生して、XBEE_IO_DATA イベントハンドラ中でサンプリングデータを取り込みます。これは、XBeeデバイスの、レスポンスフレームデータの内容に対応しています。(ファームウエアバージョン “10CD”)

device にブロードキャストアドレスを指定することはできません。

device がリモート XBee デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。またコマンドオプションは 0x02(Apply Change on remote) になります。

- **使用例**

```
stat,adc,dio = xbee_force_sample("Device1")
if not stat then error() end
for key,val in pairs(adc) do
  log_msg(string.format("adc[%s] = %s",key,val),file_id)
end
for key,val in pairs(dio) do
  log_msg(string.format("dio[%s] = %s",key,val),file_id)
end
```

27.6 xbee_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, cmd_data = xbee_at_command(device, at_cmd, param_data)

stat, frame, cmd_data = xbee_at_command(device, at_cmd, param_data, cmd_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。

frame:Table [1..#max_frame] of String

　　レスポンスフレームデータ。16進数表記の文字列

cmd_data:String　　frame リターンパラメータ中の ATコマンドレスポンスデータ部分のみを取り出したデータ。16進数表記の文字列

　　param_data に ""(空文字列)を指定して現在の設定値を取得するように指示したときに cmd_data にデータが格納されます。param_data に更新データを指定した場

	合や、レスポンスにマルチフレームデータを受信した場合には cmd_data には""(空文字列)が設定されます。
device:String	送信先の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デバイスの Node Identification 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 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、DeviceServer に接続したXBeeデバイスが対象
at_cmd:String	XBeeデバイスに送信する AT コマンド文字列(2 文字の大文字ASCII)
param_data: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。”” 空文字列を指定した場合にはATコマンドパラメータは設定されない。
cmd_option:String	リモートXBee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**

device 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 XBee デバイスにコマンドを送信します。全ての ATコマンドは、DeviceServer の COM ポートに接続された XBee デバイス(ローカル XBee 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。

device がリモート XBee デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。

cmd_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。

- **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 デバイス に対する “ND” コマンドのみです。

- **使用例**

```
stat,frame = xbee_at_command("Device2","MY","0002")
if not stat then error() end
for key,val in ipairs(frame) do
  log_msg(string.format("frame[%d] = %s",key,val),file_id)
end
```

27.7 xbee_tdcp_command()

- **機能概要**

指定したXBee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に任意の

TDCPコマンドを送信する。

- **関数定義**

stat, result = xbee_tdcp_command(device, 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。

result:Table [1..#max_frame] of String　　コマンドリプライデータ文字列

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。

device:String　　送信先の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。

Beeデバイスの Node Identification 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

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

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 が指定されたものとする。

- **備考**

stat には DeviceServerからコントローラデバイスへのアクセスに成功した場合には true が設定されて、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 false を設定します。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 配列の内容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”)

device に、DeviceServer の COM ポートに接続したXBeeデバイス(送信元)を指定すると、XBee デバイスでパケットを受信することができませんので注意してください。(ファームウエアバージョン “10CD”)

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、DeviceServe 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**

device 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**

```
stat,result = xbee_tdcp_command("Device2","port_read")
if not stat then error() end
for key,val in ipairs(result) do
  log_msg(string.format("result[%d] = %s",key,val),file_id)
end
```

27.8 xbee_tdcp_safe_retry()

- **機能概要**

指定したXBee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に任意のTDCPコマンドを送信する。コマンドリプライデータの取得に失敗した場合に自動でリトライ動作を行う。

- **関数定義**

stat, result = xbee_tdcp_safe_retry(device, tdcp_cmd [,max_retry])

- **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 [1..#max_frame] of String　　コマンドリプライデータ文字列
　　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。

device:String　　送信先の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。
　　Beeデバイスの Node Identification 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
　　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

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

max_retry:Number　　リプライデータの取得に失敗した時に、リトライ動作を行う最大回数を指定する
　　パラメータ省略時には、2 が指定されたものとする。

- **備考**

xbee_tdcp_command() と同様に TDCP コマンドを実行しますが、リプライデータ(TDCP リプライパケット)の受信に失敗した場合にはコマンド実行のリトライを行います。このため、複数回 TDCP デバイス側で同一コマンドが実行される可能性がある点に注意する必要があります。

TDCP デバイスにリクエストを送信してリプライを受信するまで、他スレッドの xbee_tdcp_safe_retry() コマ

ンド実行は待ち状態になります。(xbee_tdcp_command() コマンドでは同時に実行することができます)

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”)

device に、DeviceServer の COM ポートに接続したXBeeデバイス(送信元)を指定すると、XBee デバイスでパケットを受信することができませんので注意してください。(ファームウエアバージョン “10CD”)

stat と result に設定される値については xbee_tdcp_command() コマンドの説明も参照してください。

- **制限事項**

- **使用例**

```
stat,result = xbee_tdcp_safe_retry("Device2","port_read")
if not stat then error() end
for key,val in ipairs(result) do
  log_msg(string.format("result[%d] = %s",key,val),file_id)
end
```

27.9 xbee_tdcp_far_command()

- **機能概要**

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コマンドを送信する。複数のTDCPデバイスを途中に経由してコマンドを転送して実行することができる。コマンドリプライデータの取得に失敗した場合に自動でリトライ動作を行う。

- **関数定義**

stat, result = xbee_tdcp_far_command(tdcp_cmd, no_result, device1 [,device2 [,device3]])

- **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 [1..#max_frame] of String　　コマンドリプライデータ文字列

　　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。

no_result:Boolean　　true を指定すると、コマンドリプライデータ(result) の受信を省略する。

device1, device2, device3:String

　　送信先のXBeeデバイスまたは途中に経由する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。

　　Beeデバイスの Node Identification 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

　　XBeeデバイスの Source Address(16bitアドレス)16進数表記の文字列

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

- **備考**

xbee_tdcp_command(), xbee_tdcp_safe_retry() と同様に TDCP コマンドを実行しますが、途中に複数のデバイスを経由して転送することができるため、単体のXBee デバイスの通信範囲を超えた遠距離にあるリモートデバイスの操作が可能になります。

リプライデータ(TDCP リプライパケット)の受信に失敗した場合にはコマンド実行のリトライを行います。このため、複数回 TDCP デバイス側で同一コマンドが実行される可能性がある点に注意する必要があります。リトライの最大回数は 2 で固定です。

device1, device2, device3 には番号の低い順番から、（もし必要なら複数の）途中に経由するXBee デバイスとコマンド実行対象のXBee デバイスを指定します。パラメータで指定した最後(パラメータで一番右に指定したデバイス)で tdcp_cmd で指定したコマンドが実行されます。例えば、device1 と device2 を指定した場合には、サーバーから device1 を経由してコマンドが device2 に送信されて、device2 でコマンド実行されます。no_result が false の場合には、リプライパケットが device1 経由でサーバーに戻されてresult パラメータに格納されます。

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60バイト程度までに制限されます。これは、途中に経由するデバイスのルーティング情報を含めてコマンド文字列を送信するために、XBee で本来送信可能なパケットサイズ(100 bytes) よりも少なくなるためです。

device<n> に、DeviceServer の COM ポートに接続したXBeeデバイス(送信元)を指定すると、XBee デバイスでパケットを受信することができませんので注意してください。(ファームウエアバージョン “10CD”)

- **制限事項**

コマンド実行が行なわれる TDCP デバイスのファームウエアバージョンは 2.00 以降である必要があります。途中に経由する TDCP デバイスのバージョンは問いません。

request_tdcp_command() と比べてこの関数の実行時間は長くなります(2秒程度)。これは、リプライを受信する場合には(no_result=false)、リプライパケットリダイレクト用の設定コマンドが送信された後に、リクエストコマンドが送信され、その後リプライパケット自身も途中に複数のデバイスを経由して送信されてくるためです。

- **使用例**

```
stat,result = xbee_tdcp_far_command("adc_read",false,"Device4","Device2","Device1")
if not stat then error() end
for key,val in ipairs(result) do
  log_msg(string.format("result[%d] = %s",key,val),file_id)
end
```

27.10 xbee_my_serial_number()

- **機能概要**

  DeviceServer の COM ポートに接続されたXBeeデバイスのシリアル番号(64bit Adress)と 16bitアドレスを取得する。

- **関数定義**

  stat, serial_number, addr16 = xbee_my_serial_number()

- **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。
serial_number:String	DeviceServer の COM ポートに接続されたXBee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
addr16:String	DeviceServer の COM ポートに接続されたXBeeデバイスの16ビットアドレス 16進数表記の文字列

- **備考**

  16 ビットアドレスは、この関数実行時にCOM ポートに接続されたXBee デバイスに AT コマンド(“MY”)を使用して値を取得しますので、XBee デバイス管理プログラムでマスターに未登録の場合でも実行可能です。

- **制限事項**

- **使用例**

```
stat,server_addr,addr16 = xbee_my_serial_number()
log_msg("server address = “ .. server_addr,file_id)
```

27.11 xbee_find_device()

- **機能概要**

  指定したXBeeデバイスの Source Address(16bit)とSerialNumber(64bit), NodeIdentifier 文字列を取得する。

- **関数定義**

  stat, addr16, serial_number, name, localdevice = xbee_find_device(device)

- **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。
device: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デバイスの Node Identification 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 XBeeデバイスの Serial Number(64bitアドレス)16進数表記の文字列
addr16:String	XBeeデバイスの Source Address(16bitアドレス)16進数表記の文字列
serial_number:String	XBeeデバイスの SerialNumber(64bitアドレス)16進数表記の文字列
name:String	XBeeデバイスの Node Identification 文字列
localdevice:Boolean	XBeeデバイスが DeviceServer 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**

  XBee デバイス管理プログラム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。

  device に指定した文字列は、マスター中の NodeIdentifier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) を検索します。

- **制限事項**

- **使用例**

```
stat,addr16,serial,name,localdevice = xbee_find_device("Device1")
```

28 XBee-ZB API (Lua)

ZB サービスモジュールで管理している XBee-ZB(Series2)デバイスを操作するためのライブラリ関数です。APIで操作するXBee-ZB デバイスは、予め “ZBデバイス管理”プログラムでマスターに登録して下さい。

28.1 zb_all_list()

- **機能概要**

  “ZBデバイス管理プログラム” でマスターに登録された、全ての XBee-ZB デバイスリストを取得する。

- **関数定義**

  stat, serial, netaddr, name, type, id, localdevice = zb_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。

  serial:Table [1..#max_device] of String　　XBee-ZBデバイスのSerial Number(64bitアドレス)
  16進数表記の文字列

  netaddr:Table [1..#max_device] of String　　XBee-ZBデバイスのNetwork Address(16bitアドレス)
  16進数表記の文字列

  name:Table [1..#max_device] of String　　XBee-ZBデバイスの Node Identification 文字列

  type:Table [1..#max_device] of String　　XBee-ZB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"

  id:Table [1..#max_device] of String　　XBee-ZBデバイスの　DeviceTypeID(32bit)
  16進数表記の文字列

  localdevice:Table [1..#max_device] of Boolean
  XBee-ZBデバイスが DeviceServer 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**

  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、“ZB デバイス管理プログラム”の“探索＆登録”ツールボタンを使用してください。

- **使用例**

```
stat,serial,netaddr,name,type,id,localdevice = zb_all_list()
if not stat then error() end
for key,val in ipairs(serial) do
  log_msg(string.format("device[%d] serial = %s netaddr = %s name = %s is_local = %s",key,val,
                        netaddr[key],name[key],tostring(localdevice[key])),file_id)
end
```

28.2 zb_transmit_data()

- **機能概要**

  指定したXBee-ZB デバイスにデータを送信する。

- **関数定義**

  stat = zb_transmit_data(device, rf_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。
device:String	送信先のXBee-ZB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-ZBデバイスの Node Identification 文字列 XBee-ZBデバイスの Serial Number(64bitアドレス)16進数表記の文字列 XBee-ZBデバイスの Network 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、DeviceServer に接続したXBee-ZBデバイスが対象
rf_data:String	送信するデータ。16進数表記の文字列で指定する。

- **備考**

  rf_data に指定可能なデータ数はXBee-ZB デバイスの仕様によって255バイトまでに制限されます。フラグメンテーションによって複数回パケットが送信されるのを防ぐためには 84バイト、"Encryption enabled" 指定時には66バイト以内にしてください。

- **使用例**

```
stat = zb_transmit_data("Node2",list_to_hex(unpack(str_to_tbl("Hello World!"))))
if not stat then error() end
```

28.3 zb_dio()

- **機能概要**

指定したXBee-ZBデバイスの DIO コンフィギュレーションを DO Low または DO High に設定する。

- **関数定義**

stat = zb_dio(device, bit, flag)

- **パラメータとリターン値**

stat:Boolean	成功功した場合は true, 失敗した場合は false が返る。
device:String	送信先のXBee-ZB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-ZBデバイスの Node Identification 文字列
	XBee-ZBデバイスの Serial Number(64bitアドレス)16進数表記の文字列
	XBee-ZBデバイスの Network 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、DeviceServer に接続したXBee-ZBデバイスが対象
bit:Number	XBee-ZB デバイスのDIO ビット番号を指定する。
	指定可能な値は 0, 1, 2, 3, 4, 5, 10, 11, 12
flag:boolean	true を指定すると、DO High に設定され、false を指定すると DO Low に設定する。

- **備考**

device がリモート XBee デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。またコマンドオプションは 0x02(Apply Change on remote) になります。

- **使用例**

```
stat = zb_dio("Node1",0,false)
if not stat then error() end
```

28.4 zb_force_sample()

- **機能概要**

指定した XBee-ZBデバイスの有効な入力データを全て取得する。

- **関数定義**

stat, adc, dio = zb_force_sample(device)

- **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。
device:String	送信先のXBee-ZB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-ZBデバイスの Node Identification 文字列
	XBee-ZBデバイスの Serial Number(64bitアドレス)16進数表記の文字列
	XBee-ZBデバイスの Network Address(16bitアドレス)16進数表記の文字列
	“” 空文字列は、DeviceServer に接続したXBee-ZBデバイスが対象
adc:Table	XBeeデバイスのA/D 変換入力値。
adc[“<bit_number”]:String	0から1023までの整数値。文字列で入る
	<bit_number> にはA/D 変換を行ったDIO ビット番号が文字列で入る。

dio:Table	XBeeデバイスのDI 入力値。
dio[“<bit_number”]:String	”1” で High, “0” で Low を示す。 <bit_number> にはDI入力対象の DIO ビット番号が文字列で入る。

- **備考**

XBee-ZB デバイスの DIO 設定で、少なくとも一つ以上の DIO が 入力(DIN または ADC)モードになっている必要があります。”ZB デバイス管理プログラム” から DIOや ADC 設定を操作することができます。

device にブロードキャストアドレスを指定することはできません。

device がリモート XBee-ZB デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。またコマンドオプションは 0x02(Apply Change on remote) になります。

- **使用例**

```
stat,adc,dio = zb_force_sample("Device1")
if not stat then error() end
for key,val in pairs(adc) do
  log_msg(string.format("adc[%s] = %s",key,val),file_id)
end
for key,val in pairs(dio) do
  log_msg(string.format("dio[%s] = %s",key,val),file_id)
end
```

28.5 zb_at_command()

- **機能概要**

指定した XBee-ZB デバイスに任意の ATコマンドを送信する。

- **関数定義**

stat, frame, cmd_data = zb_at_command(device, at_cmd, param_data [,no_result])

stat, frame, cmd_data = zb_at_command(device, at_cmd, param_data, cmd_option [,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。
frame:Table [1..#max_frame] of String	レスポンスフレームデータ。16進数表記の文字列
cmd_data:String	frame リターンパラメータ中の ATコマンドレスポンスデータ部分のみを取り出したデータ。16進数表記の文字列 param_data に ""(空文字列)を指定して現在の設定値を取得するように指示したときに cmd_data にデータが格納されます。param_data に更新データを指定した場合や、レスポンスにマルチフレームデータを受信した場合には cmd_data には""(空文字列)が設定されます。
device:String	送信先のXBee-ZB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。

	ZBee-ZB デバイスの Node Identification 文字列 XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列 XBee-ZB デバイスの Network 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、DeviceServer に接続した XBee-ZB デバイスが対象
at_cmd:String	XBee-ZB デバイスに送信する AT コマンド文字列(2 文字の大文字ASCII)
param_data: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。”” 空文字列を指定した場合にはATコマンドパラメータは設定されない。
no_result:Boolean	コマンドリプライデータ(frame) の受信を省略する。パラメータ省略時には、false が指定されたものとする。
cmd_option:String	リモートXBee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**

device 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 XBee-ZB デバイスにコマンドを送信します。全ての ATコマンドは、DeviceServer の COM ポートに接続された XBee-ZB デバイス(ローカル XBee 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。

device がリモート XBee-ZB デバイスの場合には、送信されるリモートコマンド(API フレーム中) の FrameID は常に 0x01 になります。

cmd_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コマンドに "ND" を指定した場合は、コマンド実行終了まで10秒程度かかります。この時間は、"サーバー設定" プログラムの "XBee-ZB" 設定タブ中にある "タイムアウト検出時間" と同じになります。

- **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 デバイス に対する “ND” コマンドのみです。

- **使用例**

```
stat,frame = zb_at_command("Node2","MY","")
if not stat then error() end
for key,val in ipairs(frame) do
  log_msg(string.format("frame[%d] = %s",key,val),file_id)
end
```

28.6 zb_tdcp_command()

- **機能概要**

  指定した 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に任意の TDCPコマンドを送信する。

- **関数定義**

  stat, result = zb_tdcp_command(device, 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。

  result:Table [1..#max_frame] of String　　コマンドリプライデータ文字列

  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。

  device:String　　送信先の XBee-ZB 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-ZB デバイスの Node Identification 文字列

  XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列

  XBee-ZB デバイスの Network 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

  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 が指定されたものとする。

- **備考**

  stat には DeviceServerからコントローラデバイスへのアクセスに成功した場合には true が設定されて、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 false を設定します。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 配列の内容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-ZB デバイスの仕様によって255 - 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。フラグメンテーションによって複数回パケットが送信されるのを防ぐためには 75 バイト、"Encryption enabled" 指定時には 57 バイト以内にしてください。

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、DeviceServe 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**

device 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**

```
stat,result = zb_tdcp_command("Node2","port_read")
if not stat then error() end
for key,val in ipairs(result) do
  log_msg(string.format("result[%d] = %s",key,val),file_id)
end
```

28.7 zb_tdcp_safe_retry()

- **機能概要**

指定した 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に任意の TDCPコマンドを送信する。コマンドリプライデータの取得に失敗した場合に自動でリトライ動作を行う。

- **関数定義**

stat, result = zb_tdcp_safe_retry(device, tdcp_cmd [,max_retry])

- **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 [1..#max_frame] of String　　コマンドリプライデータ文字列
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。

device:String　　送信先の XBee-ZB 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-ZB デバイスの Node Identification 文字列
XBee-XB デバイスの Serial Number(64bitアドレス)16進数表記の文字列
XBee-ZB デバイスの Network 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

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

max_retry:Number　　リプライデータの取得に失敗した時に、リトライ動作を行う最大回数を指定する
パラメータ省略時には、2 が指定されたものとする。

- **備考**

zb_tdcp_command() と同様に TDCP コマンドを実行しますが、リプライデータ(TDCP リプライパケット)の受信に失敗した場合にはコマンド実行のリトライを行います。このため、複数回 TDCP デバイス側で同一コマンド

が実行される可能性がある点に注意する必要があります。

TDCP デバイスにリクエストを送信してリプライを受信するまで、他スレッドの zb_tdcp_safe_retry() コマンド実行は待ち状態になります。(zb_tdcp_command() コマンドでは同時に実行することができます)

tdcp_cmd に指定可能な文字列長はXBee-ZB デバイスの仕様によって255 - 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。フラグメンテーションによって複数回パケットが送信されるのを防ぐためには 75 バイト、"Encryption enabled" 指定時には 57 バイト以内にしてください。

stat と result に設定される値については zb_tdcp_command() コマンドの説明も参照してください。

- **制限事項**

- **使用例**

```
stat,result = zb_tdcp_safe_retry("Node2","port_read")
if not stat then error() end
for key,val in ipairs(result) do
  log_msg(string.format("result[%d] = %s",key,val),file_id)
end
```

28.8 zb_my_serial_number()

- **機能概要**

DeviceServer の COM ポートに接続されたXBeeデバイスのシリアル番号(64bit Adress)を取得する。

- **関数定義**

stat, serial_number = zb_my_serial_number()

- **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。

serial_number:String　　DeviceServer の COM ポートに接続されたXBee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**

- **制限事項**

- **使用例**

```
stat,server_addr = zb_my_serial_number()
log_msg("server address = " .. server_addr,file_id)
```

28.9 zb_find_device()

- **機能概要**

  指定した XBee-ZB デバイスの Network Address(16bit)とSerialNumber(64bit), NodeIdentifier 文字列, デバイスタイプ, DeviceTypeIDを取得する。

- **関数定義**

  stat, netaddr, serial, name, type, id, localdevice = zb_find_device(device [,try_resolve])

- **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。
device:String	検索対象の XBee-ZB 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-XB デバイスの Node Identification 文字列 XBee-ZB デバイスの Network Address(16bitアドレス)16進数表記の文字列 XBee-ZB デバイスの Serial Number(64bitアドレス)16進数表記の文字列
try_resolve:Boolean	true を指定すると、マスターファイル中にデバイスが見つからなかった場合にはサーバーPCに接続した XBee-ZB デバイスで DN (Resolves an Node Identifier)コマンドを実行してアドレス解決を試みる。このパラメータを省略した場合には false が設定される。
netaddr:String	XBee-ZB デバイスの Network Address(16bitアドレス)16進数表記の文字列
serial:String	XBee-ZB デバイスの SerialNumber(64bitアドレス)16進数表記の文字列
name:String	XBee-ZB デバイスの Node Identification 文字列
type:String	XBee-ZB 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"
id:String	XBee-ZB デバイスの DeviceTypeID(32bit)16進数表記の文字列
localdevice:Boolean	XBee-ZB デバイスが DeviceServer 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**

  "ZB デバイス管理プログラム" でマスターに登録済みの XBee-ZB 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。

  Network Address を指定して検索することもできますが、このときは DeviceServer のアドレス変換キャッシュに格納されている最新の 16ビットアドレスを対象に検索します。FFFE 等の不明アドレスを指定すると最初に一致したデバイスを見つけます。

  device に指定した文字列は、マスター中の 64ビットアドレス値または16ビットアドレス値として先に検索され、見つからなかった場合に NodeIdentifier として検索します。

- **制限事項**

  try_resole を true に指定して、マスターに未登録の XBee-ZB デバイスを検索して結果を取得した場合には、

type と id, localdevice のリターン値は設定されません。

- **使用例**

```
stat,netaddr,serial,name,type,id,localdevice = zb_find_device("Node1")
```

29 ネットワーク通信 API (Lua)

DeviceServer 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、"WebAPI 実行用 API"の章を参照して下さい。また、DeviceServer の機能を WebAPI 経由でアクセスしたい場合には、"WebAPI" の章を参照して下さい。

29.1 udp_send_data()

- **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_data(host, port, data [,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
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 0 が設定される 0: Shift-JIS 1: UTF-8 2: UCS2 (Unicode文字で "¥uxxxx" エンコード表記される)

- **備考**

data で指定された文字列をDeviceServerから指定されたホストへデータパケットを作成して送信します。

host に空文字列 "" を指定した場合は "localhost" が送信先になります。

- **使用例**

```
for cnt = 1,10,1 do
   stat = udp_send_data("localhost",8091,"TestMessage #" .. tostring(cnt))
   if not stat then error() end
end
```

29.2 tcp_send_data()

- **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_data(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

  0: Shift-JIS

  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DeviceServer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_data() 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_data("localhost",8091,"TestMessage #" .. tostring(cnt),1,2,10)
   if not stat then error() end
end
```

29.3 tcp_send_recv_data()

- **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_data(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 0: Shift-JIS 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DeviceServer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、DeviceServer 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_data() 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_data() 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_data("localhost",8091,"TestMessage #" .. tostring(cnt),1,2)
   if not stat then error() end
end
```

29.4 perform_csvif

- **機能概要**

指定されたホストとポートに対して CSVIF パケットを送信した後、応答パケットを受信する

- **関数定義**

reply_tbl = perform_csvif(packet_type, title_list_csv, data_list_csv [,target_host, target_port])

reply_tbl = perform_csvif(packet_type, param_tbl [,target_host, target_port])

- **パラメータとリターン値**

reply_tbl:Table of String　　CSVIF リプライパケットで受信した値(キーと値のペア)が入る。

packet_type:String　　CSVIF パケットタイプ文字列

title_list_csv:String　　CSVIF パラメータタイトルリスト(CSV 形式)

data_list_csv:String　　CSVIF パラメータデータリスト(CSV 形式)

target_host:String　　送信先ホスト名。省略時は "localhost"が指定される

target_port:Number　　送信先ポート番号。省略時は 27102 が指定される

param_tbl:Table of String

CSVIF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、CSVIFパラメータのキーと値にそれぞれ対応する。

- **備考**

外部アプリケーションやサーバーに対して、イベント情報やリクエスト要求を行うときに使用します。通信には、CSVIF プロトコル(TCP/IP) でキーと値からなる複数のパラメータとパケットタイプ文字列を送信して、サーバー側で処理した後リプライデータを受信します。TCP/IP のストリームソケットを使用しますので、サーバー側が確実にデータを受信したことを確認できます。

単純なイベント通知でリプライ情報が必要ない場合や、高速に通信を行いたい場合には、この関数の代わりにデータグラム通信を行う udp_send_data() 関数の使用を検討してください。

CSVIF プロトコルで、リクエストパケットを受信する側のアプリケーションでは、CSVIF プロトコルの仕様に基づいたサーバー機能を実行している必要があります。パケットタイプ文字列やリクエストパラタメータ、リプライパケットのデータ内容は、実装する CSVIF サーバー側で自由に決めることができます。

data_list_csv または、param_tbl に入っている日本語文字列は、Lua スクリプトファイル中に記述された UTF-8 から、Shift_JIS コードに変換された後CSVIF リクエストパケットに格納されます。また反対に、CSVIF リプライパケットで受信した Shift_JIS コードは UTF-8 コードに変換された後、reply_tbl に格納されます。

CSVIF パケットと通信プロトコルの詳細は、**"DeviceServer CSVIF インターフェイスマニュアル"** を参照して下さい。

- **制限事項**

- **使用例(DeviceServer 内蔵の CSVIF サーバーに、スクリプト実行を行う SCRIPT_EXEC パケットを送信)**

```
local req_type = "SCRIPT_EXEC"
local req_title = list_to_csv("SessionToken","ScriptName","Key_1","Value_1","Key_2","Value_2")
local req_data = list_to_csv("1234","PARAM_ECHO","キー値１","値１","Key2","Val2")
rst_tbl = perform_csvif(req_type,req_title,req_data)
  if (rst_tbl["ResultStatus"] == "Success") then
    for key,val in pairs(rst_tbl) do
      log_msg(string.format("CSVIF reply[%s] = %s",key,val),file_id)
    end
  else
    log_msg("*ERROR* perform_csvif failed",file_id)
end
```

29.5 scratch_send()

- **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。ストリームの先頭に、データ長を格納した４バイトのデータを付加して送信する

- **関数定義**

  stat = scratch_send(host, 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

  data:String　　送信する文字列データ

  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**

この関数は SCRATCH プログラム(http://scratch.mit.edu/about/) の IDE (ver1.4) に対して "Remote Sensors Protocol" でブロードキャストメッセージを送信するときに使用できます。

data で指定された文字列をDeviceServer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。ポート番号は 42001 で固定です。

データストリームの先頭に、data パラメータで指定したデータの長さを格納した４バイトのバイナリデータを BigEndian 形式で付加して送信します。

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。

- **使用例**

```
for cnt = 1,100,1 do
  if not scratch_send("localhost",'sensor-update "data1" ' .. tostring(600 + cnt) .. ' "note" ' ..
          tostring(100 + cnt) .. ' "ADC" ' .. tostring(cnt)) then error() end
  if not scratch_send("localhost",'broadcast "my_message"') then error() end
  wait_time(100)
end
```

30 マスターファイルAPI (Lua)

DeviceServer 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、DeviceServer 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 DeviceServer をインストールしたディレクトリの中にファイル名 (C:¥Program Files¥AllBlueSystem¥masters.xml または C:¥Program Files (x86)¥AllBlueSystem¥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、データベースAPIや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・削除などを手動で更新します。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 DeviceServerに変更を通知する必要があります。（詳しくは master_reload() 関数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>
        <XBeeDevice>
            <Description>XBee device master</Description>
            <List>
                <Item>
                    <SerialNumber>0013A200404AC39C</SerialNumber>
                    <MyAddress16>0A01</MyAddress16>
                    <NodeIdentifier>Device1</NodeIdentifier>
                    <RSSI>29</RSSI>
                </Item>
                <Item>
                    <SerialNumber>0013A200404AC397</SerialNumber>
                    <MyAddress16>0B02</MyAddress16>
                    <NodeIdentifier>Device2</NodeIdentifier>
                    <RSSI>41</RSSI>
                </Item>
                <Item>
                    <SerialNumber>0013A20040558026</SerialNumber>
                    <MyAddress16>0D04</MyAddress16>
                    <NodeIdentifier>Device4</NodeIdentifier>
                    <RSSI>48</RSSI>
                </Item>
                <Item>
                    <SerialNumber>0013A200404AC398</SerialNumber>
                    <MyAddress16>0C03</MyAddress16>
                    <NodeIdentifier>Device3</NodeIdentifier>
                    <RSSI></RSSI>
                </Item>
            </List>
        </XBeeDevice>
    </Master>
</Document>
```

⚠ マスターファイル中の文字コードは Shift_JIS 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、Windows 標準の Shift_JIS コードを使用して下さい。

これに対して、DeviceServer のスクリプトファイル中に日本語を使用する場合には UTF-8N を使用します。

上記のマスターファイル例では、３つのマスター(“Sample”, ”HostsEquiv”, ” XBeeDevice”)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。例えば、”XbeeDevice” マスターには ４つのアイテムが登録されていて、各々のアイテム内に４つのフィールド (“SerialNumber”, ” MyAddress16”, ”NodeIdentifier”, ”RSSI”) が定義されています。

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。

30.1 master_reload()

- **機能概要**

  DeviceServer 内で保持しているマスターファイル(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**

  デバイスサーバー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、DeviceServer 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
```

30.2 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**

デバイスサーバー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
```

30.3 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**

デバイスサーバー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
```

30.4 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。

30.5　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)

  stat = master_item_delete(master_name, idx)

- **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。

- **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
```

30.6 master_item_list()

- **機能概要**

  マスターにある全アイテムエントリの、最初から３つ目までのチャイルドタグの値リストを取得する。

- **関数定義**

  stat, item_tag1, item_tag2, item_tag3 = 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) に入るリストの長さは、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、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３未満の場合は、対応するitem_tag<n>リストに、空文字 ‘’ が設定されます。

- **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
```

30.7 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
```

31 WebAPI(HTTP)クライアント API (Lua)

DeviceServer 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。

31.1 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。

```
json = require('json')
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 = 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

31.2 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
```

31.3 g_json.decode()

- **機能概要**

  JSON 文字列を Lua のテーブルに変換する。

- **関数定義**

  tbl = g_json.decode(json_str)

- **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が入る。

  json_str:String　　JSON 文字列。

- **備考**

DeviceServer 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/

"C:¥Program Files¥AllBlueSystem¥Scripts¥preload¥" または "C:¥Program Files
(x86)¥AllBlueSystem¥Scripts¥preload¥"フォルダ中のスクリプトファイルで設定されています。
```

この関数は、日本語などのマルチバイト文字をJSON 文字列に変換する場合に、"¥uxxxx" 形式のエンコードフォマットを自動的に UTF-8 にデコードしません。このため、DeviceServer 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
```

32 シリアルデバイスAPI (Lua)

DeviceServer 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。

DeviceServer 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。

32.1 serial_all_list()

- **機能概要**

  DeviceServer 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 DeviceServer 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、”サーバー設定プログラム” を使用して下さい。

- **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
```

32.2 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 (“COMxx”) 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 (“COMxx”)

- **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(COMxx) 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")
```

32.3 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, count = 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

count: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。

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します。
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ます。1つの FIRMATA パケット毎にバッファにまとめて格納します。
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します。

- **使用例**

```
local com = "COM1"
-----------------------------------------------------------------
-- 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
```

32.4 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の個数を取得する。

- **関数定義**

  stat, count = serial_available(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。
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。

- **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。

  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。

- **使用例**

```
local com = "COM1"
------------------------------------------------------------------
-- 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
```

32.5 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制御コードを追加する。省略時は 0 が指定される 0: 文字列の終端に制御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制御(改行)コードを付加してから送信します。

- **使用例**

```
if not serial_print("COM1","Hello World!!") then error() end
```

32.6 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("COM1","cmd param1 param2")
if stat then
  log_msg("result = " .. reply,"COMMAND_TEST")
else
  error()
end
```

32.7 serial_write()

- **機能概要**

パラメータで指定したシリアルポートに、バイナリデータや FIRMATA パケットデータを送信する。
FIRMATA プロトコルを使用して、sysex タイプのリプライデータパケットの受信をすることができる。

- **関数定義**

stat [,sysex_reply_data] = serial_write(com_port, data [,sysex_reply_command_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。
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。

data: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_data: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_byte: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。

- **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。

data 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、data パラメ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_byte 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_data 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。

- **使用例**

```
if not serial_write(“COM1”,”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)
```

32.8　twe_print()

- **機能概要**

パラメータで指定したシリアルポートに文字列を送信する。TWEワイヤレスデバイスで使用するアスキー形式の

文字列を送信するときに、リプライデータパケットの受信をすることができる。

- **関数定義**

stat [,twe_reply_data] = twe_print(com_port, str [,twe_reply_byte [,retry_mode]])

- **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。TWEワイヤレスデバイスでサポートされているアスキー形式の文字列で記述する。
twe_reply_data:String	twe_reply_byte パラメータを指定した場合に、TWEワイヤレスデバイスから受信したリプライパケットのアスキー形式の文字列。
twe_reply_byte:String	str で指定するアスキー形式のデータパケットが TWEワイヤレスデバイスでサポートしている Request-Reply タイプのコマンドデータの場合に、このパラメータで指定した ステータスバイトに一致するリプライデータパケットを受信する。2桁(1バイト)16進数の文字列形式で指定する。
retry_mode:Boolean	twe_reply_byte パラメータと共に指定可能で true に設定した場合、500ms以内にリプライパケットを受信出来なかった場合には、str パラメータで指定したデータを再送信する。省略時は true が設定される。

- **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。

TWEワイヤレスデバイスに送信する場合に、動作中のファームウエアでサポートしているアスキー形式のデータフォーマットに従って、str パラメータを作成してください。TWEワイヤレスデバイスに送信するときには、自動的に文字列の終端に CR(0x0A)コードを付加してから送信します。

TWEワイヤレスデバイスに対して、Query -> Reply タイプのパケットを送・受信する場合には、Query コマンドデータを送信するときに、twe_reply_byte パラメータを指定します。これはリプライで受信予定のデータパケット中のステータス(コマンド)バイト値を指定することで、この値に一致したデータパケットを受信するまで関数内部でウェイトします。リプライデータパケットを受信したら、その内容が twe_reply_data 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。twe_reply パラメータを指定する場合には、リモート側の応答速度が遅い場合にリトライが発生するのを防ぐために、できるだけ連続モード(子機連続モード)で動作させてください。

retry_mode が trueの場合には(デフォルト設定) Queryパケット送信後、500ms以内に Replyを受信出来なかっ

た場合には Query パケットを再送信します。最大待ち時間の 5 秒に達するまでこれを繰り返します。この関数では Query 送信時またはReply受信時の、どちらが失敗しているかの判別はできません。このため retry_mode が true の場合には受信側のモジュールで複数回、同一 Query パケットを受信する可能性がある点に注意してください。受信側のモジュールで同一パケットを受信することにより問題が発生する場合には、retry_mode パラメータを false に設定してください。

TWEワイヤレスデバイスの Query->Reply タイプのデータ送・受信では複数デバイスに対して同時にリクエストデータを送信して、それぞれのデバイスから複数のリプライデータを受信することができますが、この関数では最初に受信したリプライデータを一つだけ返します。

- **使用例**

```
--------------------------------------
-- 子デバイス ID:1に DO#1 Low 出力
-- ワイヤレスデバイスでは "超簡単！TWEアプリ" が動作中の場合
--------------------------------------
local stat = twe_print("COM10",":0180010101FFFFFFFFFFFFFFFX")
if not stat then error() end
```

```
-------------------------------------------------------------------------------------------
-- 子デバイス ID:1に接続したI2C EEPROM デバイス(24LC256) にレジスタアドレス 0x0000 書き込み
-- ワイヤレスデバイスでは "超簡単！TWEアプリ" が動作中の場合
-------------------------------------------------------------------------------------------
local stat = twe_print("COM10",":0188090150000100X")
if not stat then error() end
-------------------------------------------------------------------------------------------
-- 子デバイス ID:1に接続したI2C EEPROM デバイス(24LC256)から10バイト読み込み
-- このときリプライパケットで受信予定のステータスバイトは 0x89
-------------------------------------------------------------------------------------------
local stat,reply_data = twe_print("COM10",":0188090250000AX","89")
if not stat then error() end
log_msg("reply = " .. reply_data)
```

32.9 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 = "COM4"
-----------------------------------------------------------------
-- 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
```

32.10 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

32.11 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
```

32.12 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
```

33 FAX送信API (Lua)

DeviceServer の SERIAL モジュールで、テキストデータを FAX で送信することができます。

FAX 送信は PC にセットアップされた Microsoft FAX サービスを使用します。予め Windows コンポーネントのインストールで FAX サービス機能をインストールした後、FAX を構成して送信可能な状態にセットアップしておいて下さい。DeviceServer ではテキストデータをローカルコンピュータで動作している FAX サービスにサブミットします。FAX 送信のリトライや取り消しなど、FAX 送信ジョブの管理は Windows の FAX コンソールを使用します。

33.1 fax_submit()

- **機能概要**

  テキストデータをFAX で送信する。

- **関数定義**

  stat = fax_submit(recipient_fax_number, fax_body)

  stat = fax_submit(header, fax_body)

- **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。

  header:Table of String　FAX の送信時ヘッダ情報を格納したテーブル。下記のキー名で送信時のパラメータが指定できる。header["RecipientFax"] のみが必須パラメータで、それ以外のパラメータは、FAX のジョブ管理ステータスに表示される情報としてのみ使用されます。

  - header["RecipientFax"]:String　送信先の FAX 番号 (必須パラメータ)
  - header["RecipientName"]:String　送信先名前
  - header["Subject"]:String　件名
  - header["DocumentName"]:String　ドキュメント名
  - header["SenderTel"]:String　送信元電話番号
  - header["SenderFax"]:String　送信元FAX 番号
  - header["SenderName"]:String　送信元名前

  recipient_fax_number:String　header パラメータを使用しない場合に送信先 FAX 番号を指定する

  fax_body:Table [1..#max_line] of String　FAX 本文を文字列テーブルで指定する。

- **備考**

FAX 送信を行うために 事前にWindows コンポーネントの Microsoft FAX サーバーをインストールして、FAX を構成して下さい。

DeviceServer が動作している PC のファイル拡張子 (.txt) に関連づけられているプログラムが、Windows インストール時の ”メモ帳” アプリケーションから変更されている場合には正常に FAX 送信できない場合があります。この場合には、Windows インストール時のデフォルト設定に戻してから実行して下さい。

この関数では送信状(カバーシート)を添付することはできません。必要に応じて FAX 本文内に記述してください。

- **使用例**

```
header = { RecipientFax = “0123456789”, Subject = “監視システム警報FAX” }
body = {}
table.insert(body,”FAX 本文 1 行目”)
table.insert(body,”FAX 本文 2 行目”)
table.insert(body,”FAX 本文 3 行目”)
if not fax_submit(header,body) then error() end
```

34 MQTT クライアントAPI (Lua)

DeviceServer 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。

”サーバー設定プログラム” で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。

34.1 mqtt_all_list()

- **機能概要**

DeviceServer 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 DeviceServer 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、”サーバー設定プログラム” を使用して下さい。

- **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)
```

34.2 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
```

34.3 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" プログラムで MQTTサービスモジュールに登録しておく必要があります。

DeviceServer 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" プログラムで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
```

34.4 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" プログラムで MQTTサービスモジュール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。

  DeviceServer 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")
```

34.5 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" プログラムで MQTT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、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。

  DeviceServer 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" プログラムで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
```

34.6 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" プログラムで MQTT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、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。

  DeviceServer 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")
```

34.7 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" プログラムで MQTT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、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。

DeviceServer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" プログラムの 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")
```

34.8　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" プログラムで MQTT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、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。

DeviceServer 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" プログラムで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")
mqtt_subscribe("EndPointタイトル#1","/xbee/#,/zb/#","1,1")
```

34.9 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" プログラムで MQTT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、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" プログラムで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/#")
```

34.10 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" プログラムで MQTT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、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
```

34.11 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
```

35 DeviceServerクライアントAPI ライブラリ(XASDLCMD.DLL)

Windows PC で動作する Win32アプリケーションや EXCEL VBA 等から DeviceServer の機能を使用する場合に、ログインやスクリプト実行などを行う為の機能を 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 PC (DeviceServer をインストールした以外の PC) で利用する場合には、予めクライアント PCに DLL(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、"インストール"の章中の "ユーザーアプリケーションを利用する場合(APIライブラリを使用)" の項目を参照してください。

XASDLCMD.DLL では、"SX_" 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、DeviceServer のインストール時に提供される EXCEL デモファイル中の標準モジュールで定義しています。このモジュール定義は、自由に複製・配布して利用できます。(C:¥Program Files¥AllBlueSystem¥ExcelDemo¥サンプル.xls または (C:¥Program Files (x86)¥AllBlueSystem¥ExcelDemo¥サンプル.xls) 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。

35.1 "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
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。

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。

35.2 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 DeviceServer のログに出力されている場合がありますので、ログファイル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 DeviceServer のログに出力されている場合がありますので、ログファイルも参照してください。

35.3 SX_LoginUser()

- **機能概要**

  DeviceServerにログインする。ログインに成功すると セッショントークン文字列を取得する。

- **関数定義 VBA**

  VBA からはこの関数を直接コールしないで、代わりに LoginUser() を使用してください。

- **関数定義 Win32**

```
function SX_LoginUser(Host:PChar;Port:Smallint;UserName, Password,
                      NewSessionToken: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
UserName	ログインユーザー名
Password	ログインパスワード名
NewSessionToken	ログイン成功時にセッショントークン文字列が入る。
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**

  NewSessionToken には、予め最大 128 バイトの文字列が格納可能な領域を確保してください。

  VBA からコールする関数は、セッショントークン文字列の更新を同時に行う関数 ”LoginUser()” を使用してください。

35.4 SX_LogoutUser()

- **機能概要**

  DeviceServerからログアウトして、セッションを削除する。

- **関数定義 VBA**

  VBA からはこの関数を直接コールしないで、代わりに LogoutUser() を使用してください。

- **関数定義 Win32**

```
function SX_LogoutUser(Host:PChar;Port:Smallint;UserName,
                       SessionToken: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
UserName	ログインユーザー名
SessionToken	セッショントークン文字列
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**

VBA からコールする関数は、セッショントークン文字列の更新を同時に行う関数 ”LogoutUser()” を使用してください。

35.5 LoginUser() [VBAのみ]

- **機能概要**

DeviceServerにログインする。ログインに成功すると セッショントークン文字列を取得する。

- **関数定義 VBA**

```
Function LoginUser( ByVal Host As String,
                    ByVal UserName As String,
                    ByVal Password As String) As Boolean
```

- **関数定義 Win32**

VBA 専用の関数ですのでコールできません。

- **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
UserName	ログインユーザー名
Password	ログインパスワード名
Return値	成功した場合は true 失敗した場合は falseが返る。

- **備考**

NewSessionToken には、予め最大 128 バイトの文字列が格納可能な領域を確保してください。

ログインに成功するとモジュール内の下記 Public 変数が設定されます。これらの変数はサーバーコマンド(SX_xxxxx)を使用するときにパラメータとして使用します。

LastLoginHost	DeviceServer ホスト名
LastLoginName	DeviceServer にログインした時のログイン名
LastSessionToken	ログイン時に確立されたセッション情報

35.6 LogoutUser() [VBAのみ]

- **機能概要**

DeviceServer からログアウトする。

- **関数定義 VBA**

```
Function LogoutUser() As Boolean
```

- **関数定義 Win32**

VBA 専用の関数ですのでコールできません。

- **パラメータとリターン値**

Return値	成功した場合は true 失敗した場合は falseが返る。

- **備考**

ログアウトに成功するとモジュール内の下記 Public 変数がクリアされます。

LastLoginHost	DeviceServer ホスト名
LastLoginName	DeviceServer にログインした時のログイン名
LastSessionToken	ログイン時に確立されたセッション情報

35.7 SX_session_update()

- **機能概要**

DeviceServer にログイン中の セッション情報を更新する。

- **関数定義 VBA**

```
function SX_session_update( ByVal SessionToken As String,
                            ByVal Host As String,
                            ByVal Port As Integer) As Integer
```

- **関数定義 Win32**

```
function SX_session_update(SessionToken, Host:PChar;Port:Smallint):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
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**

自動ログアウトを防ぐために､定期的にサーバー上のセッション情報を更新するときなどに使用します。

他のライブラリ関数を使用した場合にも同様にセッション情報は更新しますので、通常はこの関数をコールする必要はありません。他のライブラリ関数をコールすることなくアイドル状態が長い場合には、この関数を定期的にコールして DeviceServer 側の自動ログアウトを防ぐ必要があります。

35.8 SX_session_renew()

- **機能概要**

DeviceServer のセッション情報を更新する。

DeviceServer で発行されている、セッショントークンの更新（文字列の変更）も行う。

- **関数定義 VBA**

```
function SX_session_renew( ByVal Host As String,
                           ByVal Port As Integer,
                           ByVal CurrentSessionToken As String,
```

ByVal NewSessionToken As String) As Integer

- **関数定義 Win32**

function SX_session_renew(Host:PChar;Port:Smallint;
CurrentSession, NewSessionToken: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
CurrentSessionToken	DeviceServer に確立された現在のセッショントークンセッション
NewSessionToken	新しく作成したセッショントークンが返る。
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**

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。

他の機能は SX_session_update() と同じ。

35.9 SX_get_shared_data()

- **機能概要**

DeviceServer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 SX_get_shared_data( ByVal SessionToken As String,
                             ByVal Host As String,
                             ByVal Port As Integer,
                             ByVal KeyStr As String,
                             ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

function SX_get_shared_data(SessionToken, Host:PChar;Port:Smallint;
KeyStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
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**

35.10 SX_set_shared_data()

- **機能概要**

DeviceServer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 SX_set_shared_data( ByVal SessionToken As String,
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_set_shared_data(SessionToken, Host:PChar;Port:Smallint;
                        KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
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。

DeviceServer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。

35.11 SX_inc_shared_data()

- **機能概要**

DeviceServer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 SX_inc_shared_data( ByVal SessionToken As String,
                        ByVal Host As String,
                        ByVal Port As Integer,
                        ByVal KeyStr As String,
                        ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_inc_shared_data(SessionToken, Host:PChar;Port:Smallint;
                        KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
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。

35.12 SX_get_permanent_data()

- **機能概要**

DeviceServerのデータベースから、key に対応する値を取得する。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 SX_get_permanent_data( ByVal SessionToken As String,
                                ByVal Host As String,
                                ByVal Port As Integer,
                                ByVal KeyStr As String,
                                ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_get_permanent_data(SessionToken, Host:PChar;Port:Smallint;
                                KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのkey 文字列
ValueStr	Key値に一致するデータベースの値が返る。
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**

35.13 SX_set_permanent_data()

- **機能概要**

DeviceServerのデータベースに、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 SX_set_permanent_data( ByVal SessionToken As String,
                                ByVal Host As String,
                                ByVal Port As Integer,
                                ByVal KeyStr As String,
                                ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

function SX_set_permanent_data(SessionToken, Host:PChar;Port:Smallint;
KeyStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのキー値を指定する。(128 文字以内の文字列)
ValueStr	Key に対応するデータベースの値を指定する。(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**

データベース中のデータを消去したい場合は value に空文字列 "" を指定します。

DeviceServerを再起動してもデータベース中のデータは最後に更新した値で保持されます。

35.14 SX_clear_permanent_data()

- **機能概要**

DeviceServerのデータベースからレコードを削除する。

- **関数定義 VBA**

function SX_clear_permanent_data(ByVal SessionToken As String,
ByVal Host As String,
ByVal Port As Integer,
ByVal KeyPrefixStr As String) As Integer

- **関数定義 Win32**

function SX_clear_permanent_data(SessionToken, Host:PChar;Port:Smallint;
KeyPrefixStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyPrefixStr	キー文字列（前方一致で検索される）に一致したレコードが削除される 空文字列を指定した場合は、全レコードが削除される
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**

35.15 SX_inc_permanent_data()

- **機能概要**

DeviceServerのデータベースから、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 SX_inc_permanent_data(ByVal SessionToken As String,

ByVal Host As String,

ByVal Port As Integer,

ByVal KeyStr As String,

ByVal ValueStr As String) As Integer

- **関数定義 Win32**

function SX_inc_permanent_data(SessionToken, Host:PChar;Port:Smallint;

KeyStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのKey値を指定する。
ValueStr	データベースの 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**

データが見つからないか、既存の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。

35.16 SX_get_oracle_data()

- **機能概要**

DeviceServerに接続した Oracleデータベースから、key に対応する値を取得する。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 SX_get_oracle_data(ByVal SessionToken As String,

ByVal Host As String,

ByVal Port As Integer,

ByVal KeyStr As String,

ByVal ValueStr As String) As Integer

- **関数定義 Win32**

function SX_get_oracle_data(SessionToken, Host:PChar;Port:Smallint;

KeyStr, ValueStr:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのkey 文字列
ValueStr	Key値に一致するデータベースの値が返る。
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**

予め、Oracle接続機能の設定を行う必要があります。”Oracle接続機能”の章を参照してください。

35.17 SX_set_oracle_data()

- **機能概要**

  DeviceServerに接続した Oracleデータベースに、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 SX_set_oracle_data( ByVal SessionToken As String,
                             ByVal Host As String,
                             ByVal Port As Integer,
                             ByVal KeyStr As String,
                             ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_set_oracle_data(SessionToken, Host:PChar;Port:Smallint;
                             KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのキー値を指定する。(128 文字以内の文字列)
ValueStr	Key に対応するデータベースの値を指定する。(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**

  データベース中のデータを消去したい場合は value に空文字列 "" を指定します。

  DeviceServerを再起動してもデータベース中のデータは最後に更新した値で保持されます。

  予め、Oracle接続機能の設定を行う必要があります。”Oracle接続機能”の章を参照してください。

35.18 SX_clear_oracle_data()

- **機能概要**

  DeviceServerに接続した Oracleデータベース(permanent_params テーブル) からレコードを削除する。

- **関数定義 VBA**

```
function SX_clear_oracle_data( ByVal SessionToken As String,
                               ByVal Host As String,
                               ByVal Port As Integer,
                               ByVal KeyPrefixStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_clear_oracle_data(SessionToken, Host:PChar;Port:Smallint;
                               KeyPrefixStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyPrefixStr	キー文字列（前方一致で検索される）に一致したレコードが削除される 空文字列を指定した場合は、全レコードが削除される
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**

予め、Oracle接続機能の設定を行う必要があります。”Oracle接続機能”の章を参照してください。

35.19 SX_inc_oracle_data()

- **機能概要**

DeviceServerに接続した Oracleデータベース(permanent_params テーブル) 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 SX_inc_oracle_data( ByVal SessionToken As String,
                             ByVal Host As String,
                             ByVal Port As Integer,
                             ByVal KeyStr As String,
                             ByVal ValueStr As String) As Integer
```

- **関数定義 Win32**

```
function SX_inc_oracle_data(SessionToken, Host:PChar;Port:Smallint;
                             KeyStr, ValueStr:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
Host	サーバーホスト名
Port	サーバーポート番号
KeyStr	データベースのKey値を指定する。
ValueStr	データベースの 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**

予め、Oracle接続機能の設定を行う必要があります。”Oracle接続機能”の章を参照してください。

データが見つからないか、既存の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。

35.20 SX_script_exec()

- **機能概要**

スクリプトを実行する。

- **関数定義 VBA**

```
function SX_script_exec( ByVal SessionToken As String,
                         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 SX_script_exec(SessionToken, Host:PChar;Port:Smallint;
                         ScriptName, KeyList, ValueList:PChar):Smallint;stdcall;export;
```

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
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**

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。

DeviceServer 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。

35.21 SX_script_exec2()

- **機能概要**

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 SX_script_exec2( ByVal SessionToken As String,
                          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 SX_script_exec(SessionToken, Host:PChar;Port:Smallint;
```

ScriptName, KeyList, ValueList, ResultKeyList, ResultValueList:PChar):Smallint;stdcall;export;

- **パラメータとリターン値**

SessionToken	DeviceServer に確立されたセッション
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**

SX_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 SX_script_exec2()関数よりも速い、SX_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。

DeviceServer 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。

35.22 SX_csv_key_value()

- **機能概要**

SX_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 SX_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 SX_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。

36　メールからの操作

DeviceServer にセットアップされたユーザースクリプトを、携帯電話やPC の電子メール経由で実行することができます。電子メールでログイン認証を行った後に、スクリプト実行を指示することができます。スクリプトの動作をメールから実行時に、スクリプトパラメータで調整することもできます。

電子メールから操作する場合は、ログイン認証のメールとスクリプト実行の２種類のメールを送信することで行います。これらの２種類のメールをDeviceServer ではメールコマンドと呼びます。

36.1　サーバー設定

メールからのコマンド実行を行うためには、“サーバー設定” プログラムで、下記の項目を設定してください。

メールからの操作を行うために、サーバー設定プログラムで設定が必要な項目一覧
メール機能を有効にする（チェックを付ける）
自分のメールアドレス(DeviceServer からのリプライメールの送信元アドレスの設定)
受信メールをチェックする(チェックを付ける)
メール確認間隔（3..5分程度を指定、自動ログアウト待ち時間の半分以下にすること）
POP サーバー設定

36.2　メールからのコマンド操作を行うユーザーアカウント

ログイン可能なユーザーアカウントであれば、どのユーザーからもメールコマンドを送信することができます。

ユーザー情報の付加情報タブの項目での、”メールコマンド応答先” が空白の場合は、メールコマンドを送信した後に DeviceServer から受け取るリプライメールは常に送信者に返されます。(デフォルト動作)
”メールコマンド応答先” に特定のメールアドレスを記載しておくと、DeviceServer からのリプライメールは、指定した特定のメールアドレスに送信されて、メールコマンド送信者の送信元アドレス(From)は使用しなくなります。これによって、メールアカウントやユーザー情報が悪用された場合でも、認証結果は特定のメールアドレスにのみ送信されるので、セキュリティが強化されます。

メールコマンドで使用するユーザーアカウントは、DeviceServer のクライアントプログラムや、その他アプリケーションで使用するアカウントとは別に作成する方が望ましいです。これは電子メールが一般的に、記述したそのままの文字（平文）で送信されるために、ユーザーアカウント情報が漏洩した場合に備えておく必要があるためです。

36.3 $LOGIN$ (ログインメールコマンド)

DeviceServer にログインを行って、セッショントークン文字列を取得します。

メール送信内容	
メール宛先	DeviceServer の POP サーバーで設定したメールアカウント
メール件名	$LOGIN$
メール本文	<UserName> <Password>

- **備考**

メール本文はテキストのみで送信してください。リッチテキストやHTMLメール、携帯電話の機種依存文字は使用できません。

$LOGIN$ は半角 ASCII 大文字で入力してください。

メール本文に空白行があった場合は、その行を無視します。

<UserName> にはログインユーザー名を入れてください

<Password> にはパスワード文字列を入れてください

リプライ内容	
メール宛先	ログインコマンドを送信したメール送信元または、ユーザー情報の“メールコマンド応答先” で設定したメールアドレス
メール件名	$REPLY$ <ServerMessage#1>
メール本文	<ServerMessage#2> [<SessionToken>]

- **備考**

ログインに成功すると<ServerMessage#1> に “-LOGIN SUCCESS-“、<ServerMessage#2> に “ログイン成功”が入ります。

失敗した場合は、<ServerMessage#1> に “-LOGIN FAIL-“、<ServerMessage#2> にエラーメッセージが入ります。

ログインに成功すると <SessionToken> にセッショントークン文字列が入ります。これは、$SCRIPT$ メールコマンド送信時に必要になりますので、メールソフトで受信時にクリップボードにコピーしておくと、次のメールコマンド送信時にペーストして使用できます。

ログインに成功した後(DeviceServer でログイン認証を行った時点から）次の $SCRIPT$ メールコマンド送信を行

って、DeviceServer で処理を開始するまでの時間がかかりすぎると、DeviceServer が自動的にログインセッションを削除しますので注意してください。この時間はサーバー設定プログラムの “自動ログアウト待ち時間” で変更できます。

36.4 $SCRIPT$ (スクリプト実行メールコマンド)

DeviceServer でスクリプト実行を行います。

メール送信内容	
メール宛先	DeviceServer の POP サーバーで設定したメールアカウント
メール件名	$SCRIPT$
メール本文	<SessionToken> <ScriptName> [<ParamKey#1> <ParamValue#1>] [<ParamKey#2> <ParamValue#2>] … [<ParamKey#n> <ParamValue#n>]

- **備考**

メール本文は必ずテキストのみで送信してください。リッチテキストやHTMLメール、携帯電話の機種依存文字は使用できません。

$SCRIPT$ は半角 ASCII 大文字で入力してください。

<SessionToken>は、$LOGIN$ コマンドでDeviceServer からリプライメールで取得した文字列を指定します。

メール本文に空白行があった場合は、その行を無視します。

スクリプトパラメータを渡す場合はキー名と値をスペースで区切って一行ずつ記述してください。

リプライ内容	
メール宛先	ログインコマンドを送信したメール送信元または、ユーザー情報の“メールコマンド応答先” で設定したメールアドレス
メール件名	$REPLY$ <ServerMessage#1>
メール本文 リターン値: スクリプト中で script_result() を使用した場合のみ	<ServerMessage#2> <ResultKey#1> <ResultValue#1> .. <ResultKey#n> <ResultValue#n>

- **備考**

スクリプト実行に成功すると<ServerMessage#1> に “-SCRIPT SUCCESS-“、<ServerMessage#2> に “スクリプト実行成功”が入ります。失敗した場合は、<ServerMessage#1> に “- SCRIPT FAIL-“、<ServerMessage#2> にエラーメッセージが入ります。

スクリプト中で、script_result() 関数を使用してリターン値を設定している場合は、設定したキー名と値のペアがリプライメール本文の内容に追加されます。このとき、キー名と値の間はスペースで区切られます。複数のリターン値を設定することもできます。

スクリプト実行のコマンドの処理が終了すると、DeviceServer は自動的にログアウト操作を行います。ログアウト後は、セッショントークン文字列は無効になり、$SCRIPT$ メールコマンドで同一の<SessionToken>を再び使用してもエラーになります。

36.4.1 実行時に追加されるパラメータ

メールコマンドでスクリプトを実行するときは、メール中に指定されたパラメータに加えて、以下のパラメータが追加指定されます。

- **メールコマンドで実行時に追加指定されるパラメータ**

パラメータキー名	説明	パラメータ値の例
$REPLYTO$	ログインセッションのユーザー情報(メールコマンド応答先)とメールヘッダ(From)を元に決めた、リプライ先メールアドレス。スクリプト中からメールコマンド送信者にメールを送信する場合は、このアドレスに送信してください。	secure_user@mail.xxxx.com
$FROM$	メールヘッダ(From)の送信元メールアドレス	user@mail.xxxx.com

36.5 FAX (FAX送信メールコマンド)

DeviceServer で FAX送信を行います。メール本文3 行目以降に格納されている本文を FAX ドキュメントに格納して送信します。

メール送信内容	
メール宛先	DeviceServer の POP サーバーで設定したメールアカウント
メール件名	FAX
メール本文	<SessionToken> <RecipientFaxNumber> [<fax message body#1>] [<fax message body#2>] [<fax message body#3>] … [<fax message body#n>]

- **備考**

メール本文は必ずテキストのみで送信してください。リッチテキストやHTMLメール、携帯電話の機種依存文字は使用できません。

FAX は半角 ASCII 大文字で入力してください。

<SessionToken>は、$LOGIN$ コマンドでDeviceServer からリプライメールで取得した文字列を指定します。

<RecipientFaxNumber> には、FAX 送信先電話番号を文字列で指定します。例 “0123456789”

FAX 送信を行うために 事前にWindows コンポーネントの Microsoft FAX サーバーをインストールして、FAX を構成して下さい。

DeviceServer が動作している PC のファイル拡張子 (.txt) に関連づけられているプログラムが、Windows インストール時の ”メモ帳” アプリケーションから変更されている場合には正常に FAX 送信できない場合があります。この場合には、Windows インストール時のデフォルト設定に戻してから実行して下さい。

この関数では送信状(カバーシート)を添付することはできません。必要に応じて FAX 本文内に記述してください。

リプライ内容	
メール宛先	ログインコマンドを送信したメール送信元または、ユーザー情報の“メールコマンド応答先” で設定したメールアドレス
メール件名	$REPLY$ <ServerMessage#1>
メール本文	<ServerMessage#2>

- **備考**

FAX 送信に成功すると<ServerMessage#1> に “-FAX SUBMIT SUCCESS-“、<ServerMessage#2> に “FAX送信ジョブ受付成功”が入ります。失敗した場合は、<ServerMessage#1> に “-FAX SUBMIT FAIL-“、<ServerMessage#2> にエラーメッセージが入ります。

DeviceServer では、Microsoft FAX サーバーにFAX送信ジョブをサブミットした時点で FAX 送信ジョブ受付成功となります。実際にFAX が相手先に送信されたかどうかは、サーバーPC のMicrosoft FAX サーバー管理コンソール等を確認する必要があります。

FAX 送信コマンドの処理が終了すると、DeviceServer は自動的にログアウト操作を行います。ログアウト後は、セッショントークン文字列は無効になり、FAX メールコマンドで同一の<SessionToken>を再び使用してもエラーになります。

36.6 NO_LOGOUT (メールコマンドを続けて実行したい場合)

メールコマンドを複数回実行したい場合は、メールコマンドの本文に下記の文字列を入れることによって、DeviceServer が自動的にログアウトしないように指示することができます。

メールコマンドでログアウトしない様に指示するコマンド
NO_LOGOUT

- **備考**

NO_LOGOUT を指定することができるメールコマンド: $SCRIPT$, FAX

NO_LOGOUT はメール本文中のどこに入れても構いません。

NO_LOGOUT を指定したメールコマンドが実行される時には、 DeviceServer はセッショントークンのタイムスタンプを更新して、“自動ログアウト待ち時間”は、コマンド実行時点から再びリセットしてカウントし始めます。

36.7 NO_REPLY (リプライメールの送信を止めたい場合)

メールコマンドのリプライメールが不要な場合は、メールコマンドの本文に下記の文字列を入れることによって、DeviceServer がリプライメールを送信しないように指示することができます。

リプライメール送信を止める様に指示するコマンド
NO_REPLY

- **備考**

NO_REPLY を指定することができるメールコマンド: $SCRIPT$, FAX

メールコマンドに NO_REPLY を付けても、スクリプト名が指定されていないなど、必須パラメータ指定にエラーがあった場合にはエラー発生を通知するためにリプライメールが送信されます。メールコマンド実行時のエラーについては、NO_REPLY を付けるとリプライメールは届きません。

37 WebProxy(Flashプログラムの操作)

DeviceServer は、Webブラウザからアラームデバイスやスクリプトを操作するためのインターフェイス(WebProxy)機能があります。WebProxyの機能は以下の特徴があります。

- HTTPサーバー機能

  インストール時にセットアップされている、 Webページ(“AlarmControl”, ”UIOUSBControl”, ”ScriptControl”)を Webブラウザで表示するために使用します。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もできます。ただし、HTTP サーバー機能としては、イントラネット(LAN)での公開を想定して、必要最低限の機能のみインプリメントしていますので、公開サーバー(インターネット上のHTTPサーバー)で利用する目的には向いていません。
- セッション認証・管理

  Webブラウザで実行されるWebページ(Flashアプリケーション)のログイン認証処理や、セッション管理を行います。
- WebブラウザとDeviceServer間のプロトコル変換

  Webブラウザで実行されるWebページ(Flashアプリケーション) が、DeviceServer にアクセスするときに、適切なポート番号に変換してコマンドのやり取りを行います。また、HTTP 通信パケットの暗号化処理も行います。

37.1 サーバー設定

WebProxy 機能を使うために、サーバー設定プログラムから設定を行います。事前にDeviceServer が動作している PC で既に HTTP サーバープログラムが動作していないことを確認してください。ポート番号80(http) でWebProxy が動作しますので、マイクロソフト社製 HTTPサーバー(IIS)や、Apache 等のHTTPサーバープログラムを既に使用している場合には、同一ポート番号でDeviceServer の WebProxy 機能は動作させることはできません。これらの既存の HTTP サーバーとDeviceServer のWebProxy 機能を両方共使用したい場合は、別のポート番号を使用するかそれぞれを別のPC に分離して設置します。設置の方法については、後述の"DeviceServerとHTTPサーバーを分離して設定する"の項を参照してください。

WebProxy機能を有効にするために、"サーバー設定"プログラムで設定が必要な項目一覧	
WebProxy機能を有効にする	チェックを付ける
Webページトップディレクトリ	通常はデフォルトで設定されている "C:¥Program Files¥AllBlueSystem¥WebRoot" または "C:¥Program Files (x86)¥AllBlueSystem¥WebRoot" のままにしてください
詳細ログを出力	任意
HTTPServerポート	任意のポート番号を指定して下さい デフォルトは 80

DeviceServer 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にエラー発生が記録されます。このとき同時に、グローバル共有変数 $HTTP_ERROR_COUNT に累計エラー数が格納されますので、ユーザースクリプトからHTTPサーバーで発生した累計エラー数を何時でも参照することができます。また、$HTTP_ERROR_COUNT グローバル共有変数を削除すると累計エラー数をリセットできます。

37.2 ユーザーアカウントの設定

インストール時にセットアップされている、 Webページのアプリケーションは、ログイン認証を行ってから操作可能になります。WebProxy経由でログイン可能にするために、ユーザー情報のアプリケーション許可フラグ中の "WebLogin" にチェックを付けて下さい。

37.3 DeviceServer でセットアップ済みのWebページ

DeviceServer のセットアッププログラムによって、以下の４つのWebページを PC にセットアップしてあります。Webページは Flash (ver9) で作成されています。クライアントで使用するWebブラウザに、Flash Player (ver9以降)がインストールされている必要があります。

- AlarmControl(アラーム操作)

  アラーム管理プログラムで登録したアラームデバイスの現在のシグナル値や I/O データの値を参照できます。また、出力ポートやシグナル値の変更をブラウザから操作することもできます。

  使用方法については、”デバイス管理” の章の ”SigSensor, NetUIO 動作確認”の項を参照してください。

AlarmControl(アラーム操作)	
Webページアドレス	http://<server_host>/remote/AlarmControl.html
ページファイル	C:¥Program Files¥AllBlueSystem¥WebRoot¥remote¥AlarmControl.html Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥WebRoot¥remote¥AlarmControl.html

<server_host> には DeviceServer の動作する PCのホスト名またはIP アドレスが入ります。

- UIOUSBControl(UIOUSB操作)

  UIOUSB デバイスの現在のI/O データの参照や変更を行うことができます。

  使用方法については、”デバイス管理” の章の ”UIOUSB動作確認”の項を参照してください。

UIOUSBControl(UIOUSB操作)

Webページアドレス	http://<server_host>/remote/UIOUSBControl.html
ページファイル	C:¥Program Files¥AllBlueSystem¥WebRoot¥remote¥UIOUSBControl.html Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥WebRoot¥remote¥UIOUSBControl.html

<server_host> には DeviceServer の動作する PCのホスト名またはIP アドレスが入ります。

- ScriptControl(スクリプト操作)

  ユーザースクリプトをWebブラウザから実行させることができます。実行時のパラメータも指定できます。スクリプト中から、リターン値を設定した場合はスクリプト実行終了時に、リターン値のリストが画面上に表示されます。使用方法については、"動作確認と簡単な使い方" の章の "スクリプト作成" の項を参照してください。

ScriptControl(スクリプト操作)	
Webページアドレス	http://<server_host>/remote/ScriptControl.html
ページファイル	C:¥Program Files¥AllBlueSystem¥WebRoot¥remote¥ScriptControl.html Windows7(64bit)の場合は C:¥Program Files (x86)¥AllBlueSystem¥WebRoot¥remote¥ScriptControl.html

<server_host> には DeviceServer の動作する PCのホスト名またはIP アドレスが入ります。

37.4 DeviceServerとHTTPサーバーを分離して設定する

既存のHTTPサーバーや、ISPが提供するHTTPサーバーから DeviceServer のWebページをアクセスする方法について説明します。この設定を行うときには、DeviceServer の動作する PC の IP アドレスもしくはホスト名が HTTPサーバーからアクセス可能になっていることが必要です。

DeviceServer がイントラネット(LAN)に設置されていて、HTTPサーバーを外部 ISP が提供しているものを利用する場合には、インターネット(WAN)からイントラネット上の DeviceServer の動作する PC のポート番号80(http) にアクセス可能になっている必要があります。DeviceServer の動作する PC に、グローバル IPアドレスを割り付けるか、ダイナミック DNS の設置などが必要になります。この場合は既存のルータ等の設定変更が必要になる場合があります。

このマニュアルでは、イントラネット上に設置された DeviceServer を外部 ISP の HTTPサーバーからアクセスする方法についての手順を説明します。ルータやダイナミックDNS 等の設定については説明を省略してあります。設置を行う環境に合わせて、適宜設定内容を調整してください。詳しくは使用中のルータのマニュアルや、ネットワーク管理者、ISP のサポート窓口にご相談下さい。

設定例(説明用のアドレスやホスト名なのでこのまま使用しないでください)	
DeviceServer のLANアドレス	192.168.100.20
LAM<->WAN 間ルータのLANアドレス	192.168.100.1

LAN<->WAN 間ルータのWANアドレス	(ISPよりダイナミックに割り当て)
LAN<->WAN 間ルータのWANアドレスに、ダイナミック DNS でアサインされた公開ホスト名	myserver.dy_dns.co.jp
ISPで公開予定のホームページアドレス	wwww.my_company.isp.co.jp

WAN<->LANルータに、WAN 側からポート番号80(http) でアクセスがあったときに、LAN アドレス192.168.100.20 へ自動的に接続を行うような設定がされていることが必要です。

以降の手順の説明では、上記のような設定が既に終了していることを想定しています。具体的には、インターネット上の任意のPC から myserver.dy_dns.co.jp の ポート番号 80(http) にアクセスすると、DeviceServer の ポート番号80 に接続される状態になっている必要があります。

- **手順１ DeviceServer のWebProxy セットアップ**

DeviceServer が動作するPC で WebProxy 機能を有効にします。

LAN 上の PC からDeviceServerの動作しているホスト名もしくは IP を指定して DeviceServer のWebページにアクセス可能な状態にします。DeviceServer の動作 PC でファイアーウォールやウイルスチェックプログラムが動作している場合には、それらの設定変更が必要な場合があります。

- **手順２ ISP の公開ホームページにアプリケーションを転送**

公開予定の ISP ホームページに、ISPより指定された転送方法(通常は FTP)で、DeviceServer セットアップで配置されている、下記のフォルダ内のファイルを全て転送(アップロード)してください。

転送対象のフォルダ “C:¥Program Files¥AllBlueSystem¥WebRoot¥remote”
転送が終了したら、http://wwww.my_company.isp.co.jp/remote/AlarmControl.html にアクセスしてログイン画面が表示されることを確認してください。この時点ではまだ、ログインボタンを押してもエラーになります。

- **手順３ ISP の公開ホームページにservercfg.xml ファイル作成**

Webページのアプリケーションから DeviceServerへのコマンドリクエストの宛先を、指定したホスト名に切り替えるために、servercfg.xml ファイルを作成します。servercfg.xml ファイル内の <hostname> タグにダイナミック DNS でアサインされた公開ホスト名または、グローバルアドレスを入力します。その後、servercfg.xmlファイルをISP ホームページに転送(アップロード)します。転送先はStep2/4 で転送した ”remote” フォルダの中になります。
(http://wwww.my_company.isp.co.jp/remote/)

servercfg.xml の内容例
(インストール時にテンプレートファイル “C:¥Program Files¥AllBlueSystem¥WebRoot¥remote¥_servercfg.xml” が保管されています。リネームの後、内容を編集して利用下さい。)

```
<?xml version="1.0" encoding="utf-8" ?>
<Document>
```

```
  <Description>Web service configuration</Description>
  <WebProxy>
    <hostname>myserver.dy_dns.co.jp</hostname>
  </WebProxy>
</Document>
```

- **手順４ crossdomain.xml ファイルをDeviceServerのWebRoot に配置**

WebページのFlash アプリケーションが、ドメインを超えてアクセス可能にするために、対象ドメインを crossdomain.xml ファイルに記述しておく必要があります。crossdomain.xml ファイルの <allow-access-from> タグのdomain 属性にISP のホームページに使用しているドメインを記述します。その後、crossdomain.xmlファイルを DeviceServer の動作している PC の “C:¥Program Files¥AllBlueSystem¥WebRoot” フォルダに入れます。

crossdomain.xml の内容例

（インストール時にテンプレートファイル “C:¥Program Files¥AllBlueSystem¥WebRoot¥_crossdomain.xml” が保管されています。リネームの後、内容を編集して利用下さい。）

```
<?xml version="1.0"?>
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="all"/>
  <allow-access-from domain="wwww.my_company.isp.co.jp" />
</cross-domain-policy>
```

crossdomain.xml ファイルについての詳しい説明は adobe 社のホームページ http://livedocs.adobe.com/flash/9.0_jp/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000349.html を参照してください。

38 WebAPIサーバー機能 (HTTP Web API)

DeviceServer 内蔵の HTTP サーバー機能(WebProxy) 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。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、DeviceServer のユーザー認証やスクリプト実行、共有メモリへのアクセスなどの機能を URL 形式でアクセスできます。また実行結果のリターンパラメータを XML 形式で受け取ることが出来ます。リターンパラメータを JSON(JSONP) 形式で受信することもできます。これによって JavaScript 等から簡単に DeviceServer の機能を利用したシステムを構築することができます。

DeviceServer の WebAPI 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 DeviceServer 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。

注意　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。

38.1　/command/script (HTTP-GET, XML形式)

DeviceServer 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 XML 形式で受け取ることも出来ます。

リクエスト URL		
ホスト名	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/script	
URL パラメータ	session (必須)	DeviceServer 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
	name (必須)	DeviceServer に設定した スクリプト名
	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。DeviceServer 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。
	任意のスクリプトパラメータ	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/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。

http://localhost:8080/command/script?session=ST02983095510584&name=PARAM_ECHO&key1=val1&%E3%82%AD%E3%83%BC%EF%BC%92=%E5%80%A4%EF%BC%92

レスポンスXMLフォーマット
<?xml version="1.0" encoding="utf-8"?> <Document xmlns="http://www.allbluesystem.com/xasdl">   <Result></Result>   <ErrorText></ErrorText>   <TaskID></TaskID>   <ResultRecords>   <Item>   <Key></Key>   <Value></Value>   </Item>   </ResultRecords> </Document>

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
<TaskID>	DeviceServer で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
<ResultRecords>	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タグ内容は空になる
<Item>	script_result() 関数で設定したリターン値が入る
<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

例１ スクリプト実行成功で、リターンパラメータが無い場合

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
  <Result>Success</Result>
  <ErrorText></ErrorText>
  <TaskID>LUA029830CC41FC52</TaskID>
  <ResultRecords>
  </ResultRecords>
</Document>
```

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る場合

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
```

```
  <Result>Success</Result>
  <ErrorText></ErrorText>
  <TaskID>LUA029830C5CF6F15</TaskID>
  <ResultRecords>
    <Item>
      <Key>key1</Key>
      <Value>val1</Value>
    </Item>
    <Item>
      <Key>キー２</Key>
      <Value>値２</Value>
    </Item>
  </ResultRecords>
</Document>
```

例３ スクリプト実行に失敗した場合

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns="http://www.allbluesystem.com/xasdl">
  <Result>Fail</Result>
  <ErrorText>CertifyUpdateSession failed</ErrorText>
  <TaskID></TaskID>
</Document>
```

38.2 /command/json/script (HTTP-GET, JSON形式)

DeviceServer 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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer 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
	name (必須)	DeviceServer 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。DeviceServer 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。
	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

⚠ 注意　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>	DeviceServer で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 **
```

38.3 /command/json/shared_data (HTTP-GET, JSON形式)

DeviceServerの共有データから、key に対応する値を取得または設定する。

取得時に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。

他の全てのログインセッション中に実行するスクリプトや DeviceServer 内で実行するイベントハンドラ間で、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共有データは、サーバーが再起動するときに消去されます。再起動した時にデータを保持しておきたい場合には /command/json/permanent_data コマンドを使用してください。

リクエスト URL	
ホスト名	DeviceServer の ホスト名
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer 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
	key (必須)	DeviceServer の共有データキー名
	value (オプション)	DeviceServer の共有データキーに対応する値 このパラメータを省略した場合には、共有データの取得のみが行なわれて、指定した場合には共有データが更新されます value に “” 空文字列を指定すると、key に対応する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。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、既存の共有データが数値に変換できない場合は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)	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。このとき共有データの値に、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。

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>	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 共有データが見つからない場合は、空文字列 "" が設定される 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必ず文字列になります。 noquote=1 を指定した場合には、共有デ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。

38.4 /command/json/session_data (HTTP-GET, JSON形式)

現在ログインしているセッション内でのみ有効なセッション共有データから、key に対応する値を取得または設定する。取得時にセッション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。

他のログインセッション中に実行するスクリプトや DeviceServer 内で実行するイベントハンドラ間でデータを共有したい場合、再ログイン時にデータを保持しておきたい場合には、/command/json/shared_data コマンドを使用してください。このコマンドを使用して設定したセッション共有データは、ログアウトした場合やサーバーが再起動し

たときに消去されます。再起動した時にデータを保持しておきたい場合には /command/json/permanent_data コマンドを使用してください。

リクエスト URL		
ホスト名	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/session_data	
URL パラメータ	session (必須)	DeviceServer 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
	key (必須)	DeviceServer のセッション共有データキー名 session パラメータで指定するセッショントークンが異なる場合には、key が同一でも別々に保持されます。
	value (オプション)	DeviceServer のセッション共有データキーに対応する値 このパラメータを省略した場合には、共有データの取得のみが行なわれて、指定した場合には共有データが更新されます value に “” 空文字列を指定すると、key に対応する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。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)	セッション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″ を指定するとセッション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。このときセッション共有データの値に、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。

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":"<SessionValue>"}

(JSON形式) noquote=1 を指定した場合

リターンパラメータ中の <SessionValue> 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":<SessionValue>}

(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":"<SessionValue>"});

(JSONP形式) noquote=1 を指定した場合

リターンパラメータ中の <SessionValue> 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":<SessionValue>});

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
<SessionValue>	セッション共有データの key に対応する文字列 value パラメータを指定してセッション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 セッション共有データが見つからない場合は、空文字列 "" が設定される 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必ず文字列になります。 noquote=1 を指定した場合には、セッション共有デ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。

38.5 /command/json/permanent_data (HTTP-GET, JSON形式)

DeviceServerのデータベースから、key に対応する値(パーマネントデータ)を取得または設定する。

取得時にデータベース中に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。

他の全てのログインセッション中に実行するスクリプトや DeviceServer 内で実行するイベントハンドラ間で、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パーマネントデータは、サーバーが再起動しても最後に設定したデータが保存されています。

リクエスト URL		
ホスト名	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/command/json/permanent_data	
URL パラメータ	session (必須)	DeviceServer 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
	key (必須)	DeviceServer のデータベースキー名
	value (オプション)	DeviceServer のデータベース中のキーに対応する値(パーマネントデータ) このパラメータを省略した場合には、データの取得のみが行なわれて、指定した場合には データが更新されます value に “” 空文字列を指定すると、key に対応する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。データが見つからないか、既存のデータが数値に変換できない場合は、 ″1″ がデータに設定されます 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、既存のデータが数値に変換できない場合は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)	データベース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" を指定するとデータベース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。このときデータベースの値に、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。

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":"<PermanentValue>"}

(JSON形式) noquote=1 を指定した場合

リターンパラメータ中の <PermanentValue> 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":<PermanentValue>}

(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":"<PermanentValue>"});

(JSONP形式)noquote=1 を指定した場合

リターンパラメータ中の <PermanentValue> 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":<PermanentValue>});

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

<PermanentValue>	データベースの key に対応する文字列 value パラメータを指定して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 データが見つからない場合は、空文字列 "" が設定される 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必ず文字列になります。 noquote=1 を指定した場合には、データベース中に格納された 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。

38.6 /command/json/session_login (HTTP-GET, JSON形式)

ユーザー名とパスワードを指定して、DeviceServer にログインします。ログインに成功すると、セッショントークン文字列を取得します。

このコマンドでログインするためには、ログイン対象のユーザーのアプリケーション許可フラグに “WebLogin” が設定されている必要があります。

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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer に登録されたユーザーのログイン名 予めログイン対象のユーザーのアプリケーション許可フラグに “WebLogin” が設定されている必要があります。
	pass (必須)	ログインパスワード文字列

	permission (オプション)	ユーザー情報のアプリケーション許可フラグ(１つだけ指定可能) このパラメータを指定すると、ログイン対象のユーザーのアプリケーション許可フラグ中に、指定したフラグ名にチェックが付いていない場合にはログインできないようにします。 このパラメータの指定に関わらず、"WebLogin" アプリケーション許可フラグは必ずチェックが付いている必要があります。
	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。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。またアプリケーション許可フラグに “MyApp” が設定されていない場合にはログインを成功させないようにする。

http://localhost:8080/command/json/session_login?user=yamada&pass=tarou&permission=MyApp&callback=my_handler

注意　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)

レスポンスJSONフォーマット
{"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>", "PrevLoginTimeStamp":"<PrevLoginTimeStamp >", "Account":{"<Key>":"<Value>", ...}, "Permission":{"<Key>":"<Value>", ...}}

(JSONP形式)

レスポンスJSONPフォーマット
callback({"Result":"<Result>", "ErrorText":"<ErrorText>", "SessionToken":"<SessionToken>", "PrevLoginTimeStamp":"<PrevLoginTimeStamp >", "Account":{"<Key>":"<Value>", ...}, "Permission":{"<Key>":"<Value>", ...}});

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、DeviceServer で作成されたセッショントークン文字列
PrevLoginTimeStamp	このユーザーで前回ログインした日時が入ります。 “YYYY/MM/DD HH:MM:SS” 形式の文字列で設定されます

レスポンス JSON/JSONP中の “Account” に設定される<Key> と <Value> の内容	
<Key>	**<Value>**
UserID	ログインしたユーザーの UserID 文字列
Name	ログインしたユーザーの UserName 文字列

レスポンス JSON/JSONP中の “Permission” に設定される<Key> と <Value> の内容	
<Key>	**<Value>**
AllowLogin	ユーザーのアプリケーション許可フラグで設定した “AllowLogin” の設定値 “Yes” : フラグが設定されている場合 “No” : フラグが設定されていない場合
xxxxxxx	アプリケーション許可フラグで設定したその他の項目が同様に入ります

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","PrevLoginTimeStamp":"2011/12/22
11:34:11","Account":{“UserID”:”1234”,”Name”:”Yamada Tarou”},"Permission":{“AllowLogin”:”Yes”,
“AdminTask”:”No”, ...}}
```

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","PrevLoginTimeStamp":"2011/12/22 11:34:11","Account":{“UserID”:”1234”,”Name”:”Yamada
Tarou”},"Permission":{“AllowLogin”:”Yes”, “AdminTask”:”No”, ...}});
```

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"}
```

38.7 /command/json/session_logout (HTTP-GET, JSON形式)

DeviceServer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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer で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

38.8 /command/json/session_update (HTTP-GET, JSON形式)

DeviceServerのセッション情報を更新する。

DeviceServer 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(デフォルト設定値)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て該当するセッションが削除されます。この API は、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します。

“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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer 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
	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。

38.9 /command/json/getmyip (HTTP-GET, JSON形式)

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、DeviceServer の WebProxy 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 DeviceServer 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、DeviceServer 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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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)	DeviceServer 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
	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。 DeviceServer 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 DeviceServer 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。

38.10 /command/log (HTTP-GET)

DeviceServer 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	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した 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({});”

38.11 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. メールコマンド “$LOGIN$” でログイン認証を行なって、リプライメールで取得したセッショントークン
2. DeviceServer のクライアントライブラリ(XASDLCMD.DLL) 中の SX_LoginUser() でログイン認証して取得したセッショントークン
3. 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
4. Web API コマンド “/command/json/session_login” でユーザー認証後、取得したセッショントークン

上記 1, 2, 4 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。3 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 3 の方法が一番手軽に利用できます。

4. 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 DeviceServer 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。

⚠ 注意　ログイン認証時に使用したPC 以外からは Web API は利用できません

Web API経由で DeviceServer 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、ログイン認証を行って取得したセッショントークンを、ログイン認証時に使用した PC(HTTP クライント端末)以外の別の PC から Web API コマンドを実行するときに使用すると認証エラーが発生するようになっています。
メールコマンドでログインした場合や、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ません。

例: DeviceServer 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

DeviceServer 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 DeviceServer 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
```

38.12　サーバー設定

WebAPI (WebProxy)機能を使うために、サーバー設定プログラムから設定を行います。事前にDeviceServer が動作している PC で既に HTTP サーバープログラムが動作していないことを確認してください。ポート番号80(http) で

WebProxy が動作しますので、マイクロソフト社製 HTTPサーバー(IIS)や、Apache 等のHTTPサーバープログラムを既に使用している場合には、同一 ポート番号でDeviceServer の WebProxy 機能は動作させることはできません。この場合には別のポート番号を指定してください。

WebAPI 機能を有効にするために、 "サーバー設定"プログラムで設定が必要な WebProxy タブ項目一覧	
WebProxy機能を有効にする	チェックを付ける
Webページトップディレクトリ	通常はデフォルトで設定されている "C:¥Program Files¥AllBlueSystem¥WebRoot" または "C:¥Program Files (x86)¥AllBlueSystem¥WebRoot"のままにしてください
詳細ログを出力	任意
HTTPServerポート	任意のポート番号を指定して下さい デフォルトは 80
WebAPI を有効にする	チェックを付ける

39 Ajax Proxy機能

DeviceServer の HTTP サーバー機能(WebProxy) で公開した Webページをアクセスしているブラウザで、クロスドメイン通信を可能にするために proxy機能を利用することができます。

JavaScript 等を利用して、XMLHttpRequest 形式で外部のドメインにアクセスする場合には、DeviceServer 経由でリクエストを転送することでクロスドメイン通信を可能にします。

39.1 /proxy

パラメータで指定したURL に DeviceServer 中から HTTPリクエストを発行してレスポンスを受信・転送します。
url パラメータには、サービスを公開しているサーバーと任意のURLパラメータを渡すことができます。

リクエスト URL		
ホスト名	DeviceServer の ホスト名	
ポート番号	WebProxy サーバー設定で指定した HTTPServer ポート番号	
HTTP メソッド	GET	
パス名	/proxy	
URL パラメータ	url (必須)	サービスを公開しているサーバーの URL を指定する。 任意のURLパラメータを含めることができる。 urlパラメータ全体はURLエンコードを行なっておくこと。

例１ 外部ドメインにあるサーバー名 svr01.comで ポート番号 8080, url 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。Webブラウザは DeviceServer のHTTP サーバー機能で公開されているペー

ジ中の JavaScript 等からアクセスするものとします。

（直接アクセスする場合に指定する URL。ただしこれはブラウザ側のセキュリティの為にエラーになります）

http://svr01.com:8080?key1=val1&%E3%82%AD%E3%83%BC%EF%BC%92=%E5%80%A4%EF%BC%92

（上記の URL を proxy 経由でアクセスする場合に指定する URL。URL 全体がエンコードされている点に注意）

/proxy?url=http%3A%2F%2Fsvr01.com%3A8080%3Fkey1%3Dval1%26%25E3%2582%25AD%25E3%2583%25BC%25EF%25BC%2592%3D%25E5%2580%25A4%25EF%25BC%2592

レスポンスデータは DeviceServerで受信された内容がそのままクライアント側に返されます。

40 Oracle接続機能

DeviceServer は Oracle RDBMS に接続して、Lua ライブラリ関数からデータの保存や参照をすることができます。Oracle接続機能を有効にするためには DeviceServer にエンハンスライセンスが必要です。また、DeviceServer の動作する PC に Oracleクライアントを導入しておく必要があります。

Oracleクライアント導入時は、Oracleデータベースのクライアントライセンスが必要となります。オールブルーシステムの DeviceServer エンハンスライセンスには Oracle データベースのクライアントライセンスは含まれません。Oracleクライアントライセンスについては使用中の Oracle データベースのサービスプロバイダやデータベース管理者にお問い合わせ下さい。

40.1 動作環境

Oracle 接続機能を使用するために、必要となるOracle 関連の動作環境は以下の通りです。

動作環境	
項目	必要なバージョン・リソース
Oracleデータベースサーバー (DeviceServer の動作PC または､別 PC で運用されている必要があります)	Oracle 10gR2
Oracleクライアントソフトウエア (DeviceServer の動作PC に導入が必要です)	Oracle 10gR2 データベースサーバーに接続可能なクライアントソフトウエアバージョンを使用してください。 テーブル作成等を行う場合は SQL Plus プログラムも同時にセットアップしてください。

40.2 Oracleクライアント導入

DeviceServer が動作する PC に Oracleクライアントソフトウエアをセットアップしてください。必ず、動作中のOracleサーバーに接続可能なバージョンの、Oracleクライアントを使用してください。

Oracleクライアントのセットアップ方法については、Oracleのマニュアルを参照してください。セットアップに使用するOracleクライアントの種類は幾つかありますが、下記のいずれかをお勧めします。

- Instant Clientインストールタイプ
- 管理者インストールタイプ

セットアップが終了したら、必ず下記の環境変数が正しく設定されているかを確認してください。**環境変数はシステム環境変数に設定されていることが必要です。**

- **PATH 環境変数**
  Oracle クライアントをインストールしたディレクトリ内の “¥bin” フォルダが含まれていること。
- **NLS_LANG 環境変数**
  Instant Client で設定した場合は、必ず NLS_LANG 環境変数を JAPANESE_JAPAN.JA16SJIS に設定すること。
  この設定を行わないと、DeviceServer でOracle データベースにデータが文字化けする場合があります。

注意　Oracleクライアントインストール時の注意

DeviceServer が動作中の PC 環境に Oracleクライアントソフトウエアを後からセットアップした場合は、セットアップ終了後に必ず PCの再起動を行ってください。

40.3　Oracleユーザー作成

DeviceServer からOracleへ接続するための、Oracleユーザーを Oracleサーバーに作成してください。ユーザーが使用する表領域は DeviceServer でoracle データを格納するサイズ分が必要です。DeviceServer で使用するユーザースクリプトやユーザーアプリケーションで使用予定のデータサイズを考慮して決定してください。既存のOracle ユーザーや表領域を共用して使用することもできます。

ユーザー名やパスワードは任意の文字列を指定してください。 また、Oracleユーザーには “CONNECT” と “RESOURCE” ロールを付与してください。Oracleユーザーの作成方法や表領域の割り当て、ロールの付与については Oracle のマニュアルを参照してください。

40.4　DeviceServerデータ保管用テーブル作成

先に作成した Oracle ユーザーが DeviceServer 経由でデータを保存するためのテーブルを定義します。
テーブル定義の SQL 文と、インデックス作成用の SQL 文から成っています。
下記の SQL定義用のファイルを使用して、Oracle ユーザが使用する表領域にテーブルを作成してください。

テーブル作成用の SQL設定ファイルは、DeviceServer インストール先の “Etc” フォルダにファイル名 “oracle_table_def.sql” で格納されています (“C:¥Program Files¥AllBlueSystem¥etc¥oracle_table_def.sql” または “C:¥Program Files (x86)¥AllBlueSystem¥etc¥oracle_table_def.sql”)

DeviceServer からアクセスするテーブル定義のＳＱＬ文

```
--
-- ABS-9000 DeviceServer のオラクル接続機能で使用するテーブル定義ファイル
--
--    下記の SQL でテーブル定義を行ってください。トリガー等をこのテーブルに
--    対して定義することもできます。ただしオールブルーシステムではトリガー等の
--    スクリプトはサポート対象外となります。
--
-- トリガー定義に関する補足
--
--    DeviceServer がテーブルにデータを保存するときは、必ず既存のキーに該当する
--    レコードを delete SQL文で削除してから、新規レコードを insert SQL文で登録しています。
--    update SQL 文は使用していませんので注意してください。
--
--                                        copyright(c) 2008-2011 All Blue System

CREATE TABLE PERMANENT_PARAMS
(
  KEY_DATA          VARCHAR2(256) NOT NULL,
  VALUE_DATA        VARCHAR2(256) NOT NULL,
 PRIMARY KEY (KEY_DATA)
);

CREATE TABLE STAT_DATA
(
  KEY_DATA          VARCHAR2(256) NOT NULL,
  TIME_DATA         TIMESTAMP NOT NULL,
  VALUE_DATA        BINARY_DOUBLE NOT NULL
);

CREATE INDEX STAT_KEYTIME_IDX ON STAT_DATA(KEY_DATA, TIME_DATA);
CREATE INDEX STAT_KEY_IDX ON STAT_DATA(KEY_DATA);

CREATE TABLE STRING_LIST
(
  CHANNEL           VARCHAR2(256) NOT NULL,
  LIST_COUNT        INTEGER NOT NULL,
  STRING_DATA       VARCHAR2(256) NOT NULL,
 PRIMARY KEY (CHANNEL, LIST_COUNT)
```

```
);
```

40.5 DeviceServer のサーバー設定

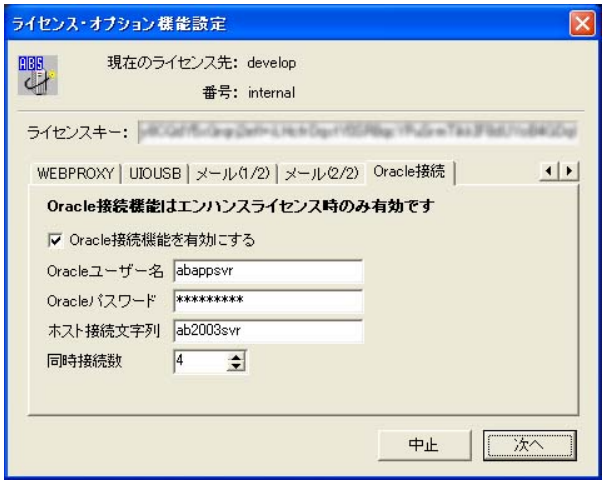
サーバー設定プログラムで Oracle接続の設定を行います。

Oracle 接続機能を有効にして、ユーザー名とパスワード、ホスト接続文字列を入力します。
Oracle クライアントにInstant Client を使用する場合は、接続文字列は “<hostname>:1521/<SID>” の形式で入力してください。<hostname> には oracle サーバーのホスト名が入ります。<SID>には Oracleサーバーの SID 文字列が入ります。（詳しくは、使用中のOracle サーバーの管理者に問い合わせて下さい。）

Oracle Net Configuration を使用してネーミングメソッドを使用する場合は、そのホスト接続名(TNS名) をそのまま接続文字列に入れてください。

Oracle SID, ネーミングメソッドの詳しい内容は Oracle のマニュアルを参照してください。

同時接続数は、DeviceServer がOracleサーバーに接続するクライアントセッション数になります。通常はデフォルトのまま変更しないで下さい。DeviceServer は、ここで設定したセッション数分のクライアント接続リソースを共有して、Oracleデータベースにアクセスします。



40.6 動作確認

下記のスクリプトを実行してデータが保存できる事を確認します。
スクリプト実行デモプログラム(ScriptExecDemo)や Webブラウザから ScriptControl(スクリプト操作)ページにアクセスして “ORACLE_DATA” スクリプトを実行してください。

ORACLE_DATA スクリプト

```
file_id = "ORACLE_DATA"
log_msg("start..",file_id)
```

```
-- データ保存
for cnt = 1,100,1 do
  if not set_oracle_data("Key" .. tostring(cnt),"テストデータ値 " .. tostring(cnt)) then error() end
end
-- データ参照
for cnt = 1,100,1 do
    stat,value = get_oracle_data("Key" .. tostring(cnt))
    if not stat then error() end
    log_msg(string.format("Key[%d] = %s",cnt,value),file_id)
end
-- データ削除（コメントアウト）
--for cnt = 1,100,1 do
--    if not set_oracle_data("Key" .. tostring(cnt),"") then error() end
--end
```

スクリプトを実行した後、Oracle クライアントソフトの SQL Plus でデータを確認します。SQL Plus で接続に使用するユーザーとホスト接続文字列は、DeviceServer のサーバー設定に使用したものと同一のものを指定してください。

SQL Plus の実行結果

```
SQL>  select count(*) from permanent_params;

COUNT(*)
----------
       100

SQL> select * from permanent_params;

KEY_DATA
--------------------------------------------------------------------------------
VALUE_DATA
--------------------------------------------------------------------------------
Key1
テストデータ値 1

Key2
テストデータ値 2

.....（データ続きがダンプされる）.....
```

40.7 運用時の注意事項(DeviceServer起動・停止)

DeviceServer は、サーバー設定でOracle同時接続数に設定した数だけ、Oracleサーバーに常に接続を行っています。これによって、スクリプトや API 実行時のパフォーマンスを向上させて、リクエスト集中時にもサーバーが高負荷状態になるのを防止しています。

Oracle サーバーを再起動したり停止する場合には、必ず DeviceServer を事前に停止させる必要があります。DeviceServer を動作させたまま、Oracleサーバーを再起動したり停止すると、DeviceServer でエラーが発生します。この様なエラー状態になった場合は DeviceServer を再起動させてください。

DeviceServer の再起動の方法は、DeviceServer の動作している PC 自身(Windows OS)を再起動させるか、コマンドプロンプトでサービス(ABAppService)を停止させた後に、再びサービス(ABAppService)を起動させます。

DeviceServer 停止コマンド実行例

```
C:¥tmp>
C:¥tmp>net stop ABAppService

ABAppService サービスは正常に停止されました。
```

DeviceServer 起動コマンド実行例

```
C:¥tmp>net start ABAppService
ABAppService サービスを開始します.
ABAppService サービスは正常に開始されました。
```

41 Windowsタスクスケジューラを使用してスクリプトを実行

DeviceServer に設定したスクリプトを予め決められた日付時刻に実行したい場合や、定期的にスクリプトを実行したい場合には Windows のタスクスケジューラ機能を利用できます。

DeviceServer には簡単に定期的なスクリプト実行を行うために、PERIODIC_TIMER イベントによるスクリプト実行機能がありますので、分単位で定期的にスクリプトを実行する場合などはこちらを利用する方法が簡単です。
PERIODIC_TIMER スクリプトの使用方法は "12.6 PERIODIC_TIMER" イベントの項を参照してください。

PERIODIC_TIMER イベントでは決められた日時にスクリプトを実行したり、曜日や時間を予め決めてスケジュールすることはできません。（ただし、ユーザーがPERIODIC_TIMER イベントハンドラに日時や曜日を判断するスクリプト

を記述すればできないことはありません)。Windows に備えられたタスクスケジューラ機能を利用すると、正確な日時や時間帯にDeviceServer のスクリプト実行をスケジュールすることができます。定期的に決められた時刻に繰り返しスクリプトを実行することも簡単にできるようになります。

Windowsタスクスケジューラを操作するたには、コマンドプロンプトからスケジュール用コマンド“schtasks” を使用します。"schtasks" でスケジュールタスク登録時に指定する、DeiveServer の実行プログラムは、"C:¥Program Files (x86)¥AllBlueSystem" または "C:¥Program Files¥AllBlueSystem"フォルダに保管されている ScriptExecCmd.exe プログラムを使用します。ScriptExecCmd.exe プログラムはコマンドプロンプトから DeviceServer に設定された任意のスクリプトを実行できます。詳しくは "11.5 スクリプト実行プログラム・コマンドライン版" を参照してください。

41.1 スケジュール実行時のDeviceServer ユーザー登録

タスクスケジューラを使用してスクリプト実行する場合には、スケジュールタスク毎に DeviceServer中に別々のユーザーアカウントを作成することをお勧めします。これは、同一ユーザーが完全に同一のタイミングで DeviceServer にログインする場合（複数のスケジュール実行が完全に同一時刻で実行される場合に発生します）には、ログイン処理中に、ユーザー情報がロック中の為にログインエラーが発生する場合があるためです。DeviceServer に登録可能なユーザーアカウント数に制限はありません。ライセンスに関係なく好きなだけ作成することができます。

ここでは ScriptExecCmd.exe プログラムで使用するユーザーアカウントをDeviceServer に新規に作成しておきます。ユーザーのアプリケーション許可フラグには、デフォルトの “AllowLogin” を指定します。（他のオプションを追加指定しても構いません）

ここでは、ユーザー名: “task1_user” パスワード: “task1_pass” として作成します。別のユーザー名やパスワードにしても構いません。その場合は、後で説明する ScriptExecCmd.ini ファイルに指定する内容も変更してください。

DeviceServer のユーザーアカウント作成の詳しい説明は “DeviceServer ユーザーマニュアル” 中の ”動作確認と簡単な使い方” の章と、”クライアントソフトウエア” の章を参照してください。

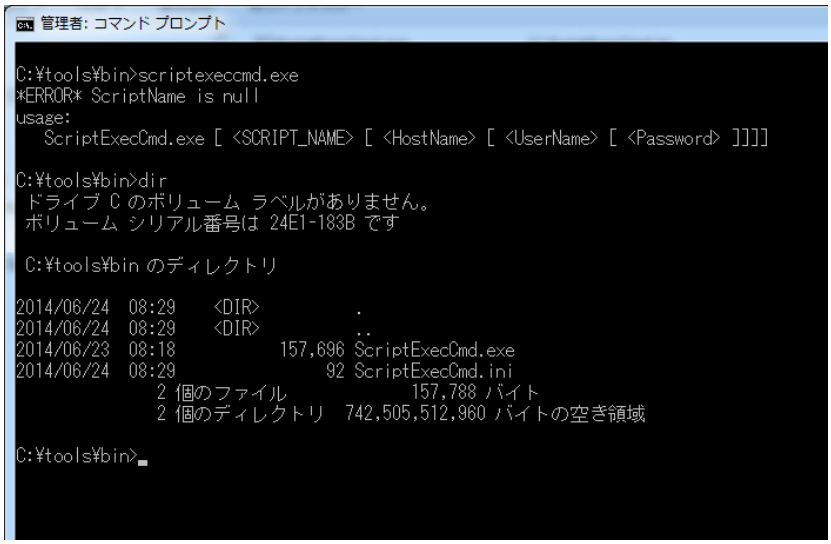
41.2 ScriptExecCmd.exe プログラムの設定

ScriptExecCmd.exe コマンドラインプログラムは、DeviceServer にログインするためのユーザーアカウントとパスワード、実行するスクリプト名などをコマンドラインパラメータに指定して実行します。コマンドラインパラメータを省略したときには、同一フォルダにある ScriptExecCmd.ini ファイルに記述された内容を参照して実行パラメータを補完できます。

タスクスケジューラに登録するプログラムは実行時のコマンドパラメータを記述することができませんので、ScriptExecCmd.ini ファイルにスクリプト名などの情報を予め格納しておきます。タスクスケジューラに追加で別のスクリプトを実行するためのタスクを登録する場合には、それぞれのタスク用に ScriptExecCmd.exe と ScriptExecCmd.ini ファイルを格納するフォルダを別に作成してください。

DeviceServerをインストールしたフォルダ("C:¥Program Files (x86)¥AllBlueSystem" または "C:¥Program Files¥AllBlueSystem")に格納されている ScriptExecCmd.exe プログラムを、新規に作成したフォルダ "C:¥Tools¥Bin" にコピーしてください。ここで作成する "C:¥Tools¥Bin" フォルダは、フォルダ名に空白文字や日本語を含まないようにするために作成しています。このフォルダ以外の場所に ScriptExecCmd.exe ファイルを配置する場合は、"schtasks" コマンドを実行する時のパス名指定部分を適宜変更してください。

ScriptExecCmd.iniファイルは最初にScriptExecCmd.exe を実行したときに、ScriptExecCmd.iniファイルが見つからない場合は自動的に作成されます。Windows のコマンドプロンプトを起動して、パラメータ無しで ScriptExecCmd.exe を実行してください。（このときエラーが表示されますが無視してください）



("C:¥Tools¥bin" フォルダにコピーした ScriptExecCmd.exe を実行して、ScriptExecCmd.ini ファイルを作成した様子)

ScriptExecCmd.iniファイルをエディタで開いて下記の内容に修正します。

"ScriptName=" の右部分には DeviceServerのスクリプト名を指定します。
"HostName=" には DeviceServer のホスト名を指定します。“localhost”を指定すると ScriptExecCmd.exe プログラムを実行する同じPC で動作している DeviceServer のスクリプトを実行します。

ここではタスクスケジューラのへのタスク登録は、DeviceServer の動作している PC で行っていますが、"HostName=localhost" 以外を記述してスケジュール実行を行うPC と DeviceServer の動作するPC を分けることもできます。この場合には ScriptExecCmd.exe とランタイムDLL(XASDLCMD.DLL) ファイルをスケジュール実行する別 PC にコピーして使用します。詳しくは "11.5 スクリプト実行プログラム" の項を参照してください。

"UserName=", "Password=" は、前の項でスケジュール実行用に新規に作成した DeviceServerユーザー情報を指定します。

"KeyList=" と "ValList=" にはスクリプトに渡すリクエストパラメータをキー名リストと値リストに分けて指定します。複数のキーと値を指定する場合にはカンマ "," で区切ります。

```
[ScriptExecCmd]
ScriptName=XBEE_ACQUISITION
HostName=localhost
UserName=task1_user
Password=task1_pass
KeyList=Key1,キー２
ValList=Value1,値２
```

41.3 タスクスケジューラへのJOB登録

ここでは、5 分毎に XBEE_ACQUISITION スクリプトを実行する例で説明します。

DeviceServer の動作する PC で、コマンドプロンプトを起動して、下記のコマンドを実行してください。必ず、システム管理者権限を持った Windows アカウントでログインしてから実行してください。

下記のコマンドは、Windows のシステム管理者特権のユーザー名が “Administrator” でパスワードが “xxxxx” の場合の例になります。2 行で表示されていますが、実際には１行で入力してください。

```
schtasks /create /tn ACQUISITION_TASK /sc minute /mo 5 /st 00:00:00 /ru Administrator /rp xxxxx
  /tr C:¥tools¥bin¥scriptexeccmd.exe
```

タスクスケジューラに対する、他のコマンド実行例は下記のようになります。

```
** Windows スケジューラ確認コマンド例 **
schtasks /query

** Windows スケジューラ削除コマンド例 **
schtasks /delete /tn ACQUISITION_TASK
```

Windows タスクスケジューラと “schtasksコマンド” についての詳しい説明はマイクロソフト社のドキュメントを参照してください。

42 preload 機能

42.1 独自のライブラリ関数や変数(定数)の作成

DeviceServerをインストールした "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(定数の様に使用する)

DeviceServer 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" フォルダ中の 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。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。

42.2 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

DeviceServer 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 DeviceServer 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。

```
-------------------------------------------------
-- xbee デバイス Device1の DIO#0 を High に設定 --
-------------------------------------------------
xbee_at_command("Device1","D0","05")
```

この場合には下記の様に記述します。

```
-------------------------------------------------------------------
-- xbee デバイス Device1の DIO#0 を High に設定する関数を定義 --
-------------------------------------------------------------------
function my_xbee_d0()
        xbee_at_command("Device1","D0","05")
end
```

preload 機能によってこのスクリプトが実行されると my_xbee_d0() 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my_xbee_d0() 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、DeviceServer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 DeviceServer を再起動するようにしてください。

43 サポート

オールブルーシステムは、ABS-9000 DeviceServer のライセンスのサポート期間内において、メールにてサポートを行います。ご質問や不具合がありましたら、メールでご連絡ください。

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-9000 DeviceServer 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。

オールブルーシステムの ABS-9000 DeviceServerは、無料のデモライセンスを使用することで、全ての機能を事前にお客様が検証可能なようなシステムを提供しております。 DeviceServer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ことを確認されることをお勧めします。機能評価の為にデモライセンスの延長等をご希望の方は、メールにて連絡をいただければ対応致します。

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。

44 本製品に使用したソフトウェアライセンス表記

44.1 License for Lua 5.0 and later versions

Copyright © 1994–2010 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF

CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

45 更新履歴

REV A.2.29 2017/11/19

topic_to_tbl ライブラリ関数を追加した。

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.2.28 2016/6/11

g_script 変数にスクリプト名を設定する機能を追加した。

exclusive_check() ライブラリ関数を追加した。

REV A.2.27 2016/4/3

ビット演算 Lua ライブラリ関数の有効ビット数について記述を追加した。

REV A.2.26 2016/3/24

serial_print() ライブラリ関数の trailing_data パラメータに NULL 指定を追加した。

REV A.2.25 2016/3/22

シリアルデバイスで、文字列形式のコマンド送信と受信を行う serial_command() ライブラリ関数を追加した。

REV A.2.24 2016/2/18

set_shared_data(), inc_shared_data(), dec_shared_data() ライブラリ関数に no_global_watch フラグを指定することで、GLOBAL_WATCH イベントのチェックを省略できるようにした。

REV A.2.23 2016/2/2

統計データベース、時系列データベース API に、リモート DeviceServer にアクセスするためのライブラリ関数を追加した。

REV A.2.22 2016/1/13

イベント,マスターの説明を一部修正

REV A.2.21 2015/10/26

script_file_list() ライブラリ関数に、指定したフォルダ以下のみを検索するパラメータを追加した

script_net_list() ライブラリ関数を追加した

REV A.2.20 2015/8/31

誤字修正

REV A.2.19 2015/6/1

mqtt_unsubscribe() の説明を一部修正

REV A.2.18 2015/5/20

MQTT 接続機能についての記述を追加した。

イベントハンドラを追加 MQTT_KEEP_ALIVE_TIMER, MQTT_PUBLISH

API ライブラリ関数を追加

mqtt_open(), mqtt_close(), mqtt_connect(), mqtt_disconnect(), mqtt_subscribe(),

mqtt_unsubscribe(), mqtt_publish(), mqtt_ping(), mqtt_all_list(), mqtt_find_endpoint()

REV A.2.17 2015/3/7

zb_find_device() リターン値の並びで、serial と netaddr が入れ替わっていたのを修正した

REV A.2.16 2015/2/25

下記のAPIライブラリ関数を新規追加

writeUTF_hex(), readUTF_hex()

REV A.2.15 2015/1/24

下記のAPIライブラリ関数を新規追加

list_shared_strlist(), list_net_strlist(), dec_permanent_data(), dec_per_net_data()

Web API /command/json/shared_data, /command/json/permanent_data の URL パラメータ "value" に

"_decrement_" を指定できるようにした

時系列文字列データ機能と下記 API ライブラリ関数を新規追加

add_series_str(), clear_series_str(), search_series_str(), key_list_series_str(),

count_series_str()

REV A.2.14 2014/12/24

リモート DeviceServer へのネットワーク接続に失敗した場合に、ロスト状態を管理するためのフラグ

$REMOTE_LOST_HOST グローバル共有文字列リストを追加した。

リモート側のグローバル共有変数とデータベース(Firebird)を操作するためのライブラリ関数を追加した。

preload 機能についての説明を追加した

REV A.2.13 2014/11/6

APIライブラリ関数を新規追加 script_free_count(), dec_shared_data()

REV A.2.12 2014/9/19

シリアルモジュールにTOCOS ワイヤレスエンジンを使用したデバイスを接続するための "TWE" タイプを追

加した。また "TWE" タイプのシリアルデータを受信したときに実行する、SERIAL_TWEイベントハンドラを新規追加した。

APIライブラリ関数を新規追加 twe_print(), ssv_to_tbl(), key_val_to_tbl(), serial_find_device()

REV A.2.11 2014/8/29

APIライブラリ関数の使用例について説明文を追加

REV A.2.10 2014/7/28

add_shared_strlist 関数に文字列リストの最大保持数を制限する limit パラメータを追加

str_to_datetime(), seconds_between() 関数を追加

REV A.2.09 2014/6/28

Windows タスクスケジューラを使用してスクリプトを実行する説明を追加

ScriptTest クライアントプログラムに別スレッド実行ボタンを追加した

REV A.2.08 2014/6/8

xbee_at_command(), zb_at_command() のリターンパラメータに cmd_data を追加した。

アラームシミュレータ(AlarmSignal.exe) プログラム起動時の記述を追加した。

コマンドリファレンスの内容に一部追記した。xbee_tdcp_command(), zb_tdcp_command()

tbl_to_str() 関数を新規追加した

REV A.2.07 2014/5/4

webapi_get_str() 関数名を http_get() に変更した。また、HTTP カスタムヘッダ情報を指定するパラメータを追加した。CSV 形式でのパラメータ渡しをやめて、Table 形式のみ受け付けるように変更した。

http_post(), http_put() 関数を新規追加した

REV A.2.06 2014/4/24

clear_permanent_strlist() 関数で指定する channel 名を前方一致の部分文字列を指定するように変更した

list_permanent_strlist() 関数を新規追加した

REV A.2.05 2014/4/10

key_list_stat_data() 関数を追加した

REV A.2.04 2014/3/7

Web API コマンドの URLパラメータに、JSON 文字列をリターンパラメータ中でネイティブで扱うための "noquote" を追加した。

REV A.2.03 2014/2/16

script_task_list()のリターン値にスクリプト名リストを追加した

critical_section_enter2(), critical_section_leave2() 関数を追加した

API関数名の間違いを修正した(互換性のため修正前の関数名も引き続き使用可)

critical_session_enter() -> critical_section_enter()

critical_session_leave() -> critical_section_leave()

REV A.2.02 2014/1/23

xbee_at_command(), zb_at_command() に コマンドオプション指定パラメータを追加した

scratch_send() ライブラリ関数を追加した

REV A.2.01 2013/12/26

誤字、その他修正

REV A.2.00 2013/12/19

XBee-ZB ZigBee 対応 Series2 デバイス対応のサービスモジュール "ZB" を追加した

REV A.1.68 2013/8/29

HTTPサーバーの累計アクセスエラー数をグローバル共有変数 $HTTP_ERROR_COUNT に保持する機能を追加

get_shared_data(),get_permanent_data(),get_session_data() に、データ取得後に自動削除するための

delete_flag オプションパラメータを追加した

REV A.1.67 2013/6/12

グローバル共有変数の変化監視用のイベント GLOBAL_WATCH"を追加した

REV A.1.64 2013/4/26

serial_all_list() ライブラリ関数を追加した

SX_csv_key_value() ライブラリ関数を追加した。

REV A.1.63 2013/3/11

Flash アプリ LoginForm でログインする機能を削除した。サーバー本体のセキュリティ強化の為、動作環

境が LAN 内に固定される制限が発生したため。

REV A.1.62 2013/3/8

Web API コマンド /command/json/session_data を追加した

REV A.1.61 2012/11/30

search_stat_data(),script_file_list() ライブラリ関数を追加した

DEVICE_MESSAGE イベントを追加した。

SERIAL デバイスを指定するときに、任意のタイトル文字列を使用可能にした

サーバー設定プログラムに “パフォーマンス” 設定タブを追加した

REV A.1.60 2012/7/18

スクリプトタスク管理プログラム(TaskList) を追加した。

script_kill(),script_task_list() ライブラリ関数を追加した。

REV A.1.59 2012/6/20

udp_send_data() ライブラリ関数に、encode パラメータを追加した

下記のライブラリ関数を追加した。

event_wait2_either(),tcp_send_data(),tcp_send_recv_data()

REV A.1.58 2012/6/5

下記のライブラリ関数を追加した。

remove_shared_strlist(),event_set2(),event_wait2()

REV A.1.57 2012/5/2

メールコマンドに メールからFAX を送信するためのコマンド FAX を追加した

REV A.1.56 2012/4/23

mail_send() 関数でメール本文を文字列配列で指定できるようにした

FAX 送信用 API fax_submit() を追加した

REV A.1.55 2012/3/27

Web API コマンド (/command/json/session_login) を追加した。

REV A.1.54 2012/3/6

SERIAL サービスモジュールを追加した

下記のライブラリ関数を追加した。

serial_readln(),serial_available(),serial_print(),serial_write(),firmata_str_encode()

firmata_str_decode(),hex72_to_tbl(),tbl_to_hex72()

下記のイベントを追加した。

SERIAL_STRING, SERIAL_FIRMATA, SERIAL_RAW

REV A.1.53 2011/12/29

WebAPI 機能にクロスドメイン通信についての記述を追加した

REV A.1.52 2011/11/30

下記のライブラリ関数を追加した

master_create(),master_delete(),master_item_add(),master_item_delete()

REV A.1.51 2011/11/17

下記のライブラリ関数を追加した

service_module_status(),service_module_restart(),perform_csvif()

REV A.1.50 2011/11/08

下記のライブラリ関数を追加した

add_oracle_strlist(),clear_oracle_strlist(),get_oracle_strlist(),remove_oracle_strlist()

shift_oracle_strlist(),pop_oracle_strlist(),count_oracle_strlist()

add_stat_oracle(),clear_stat_oracle(),summary_stat_oracle()

REV A.1.49 2011/10/27

WEBPROXY サービスのWebAPI機能に /command/json/getmyipを追加した。

REV A.1.48 2011/10/22

WEBPROXY サービスのWebAPI機能に /command/json/session_logout, /command/json/session_updateを追加した。

LoginForm プログラムの説明を追加した。

REV A.1.47 2011/09/03

WEBPROXY サービスのWebAPI機能に /command/json/permanent_data を追加した

認証省略時のセッション作成方法を SERVER_START イベントハンドラを使用するように変更

REV A.1.46 2011/08/10

誤記・スペルミス修正

REV A.1.45 2011/07/30

下記のライブラリ関数のパラメータに、テーブル型でキーと値を指定できるようにした。

script_exec() script_exec2() script_rexec() script_rexec2() script_fork_exec()

script_fork_rexec() webapi_get_str()

REV A.1.44 2011/06/04

XBee デバイス管理プログラムに、"TDCP" コマンド送信機能を追加した。

REV A.1.43 2011/05/21

Lua ソフトウエアライセンス表記を追加。

REV A.1.42 2011/04/13

メールの自動チェック機能を有効にした場合に、グローバール共有変数 $MAILBOX_COUNT の更新機能を追加した。

スクリプト中で、critical_section_enter() をコールした後に、critical_section_leave() 関数がコールされなかった場合には、強制的に開放・クローズするようにした。

REV A.1.41 2011/03/05

ServerInit(サーバー初期設定) プログラムにサーバー起動・停止ボタン、ライセンスロードボタンを追加した。

renew_session() ライブラリ関数を新規追加

REV A.1.40 2011/02/10

SERVER_START, SERVER_STOP イベントを新規追加

REV A.1.39 2011/01/22

イベントの説明に、UIOUSB と XBEE(TDCP) デバイスから出力されるイベントのリストを追加

xbee_tdcp_far_command() ライブラリ関数を新規追加

REV A.1.38 2010/12/12

json ライブラリ関数を新規追加

ビット演算ライブラリ関数を新規追加

REV A.1.37 2010/12/05

webapi_get_str() ライブラリ関数を新規追加

REV A.1.36 2010/11/30

XBEE_TDCP_DATA イベントデータ実行時のパラメータに TDCP_WHOLE を追加した。

UIOUSB_EVENT_DATA イベントデータ実行時のパラメータに EVENT_DATA_WHOLE を追加した。

REV A.1.35 2010/11/6

uio_command()ライブラリ関数のリクエストパラメータに max_retry を追加した、またリターン値にリプライ文字列を追加した。

uio_set_xxxxとuio_get_xxxx 関数を統合して入力パラメータの有無で、取得と設定を切り替えるように仕様を変更した。

uio_force_sample()関数を新規追加した

UIOUSB_EVENT_DATA イベントを新規追加した。

REV A.1.34 2010/8/11

script_fork_rexec()ライブラリ関数を追加した。

REV A.1.33 2010/7/31

gps_coordinate_dmm()ライブラリ関数を追加した。

REV A.1.32 2010/7/21

ライブラリ関数 pop_permanent_strlist(),shift_permanent_strlist(),count_permanent_strlist() を追加した

REV A.1.31 2010/7/3

ライブラリ関数 pop_shared_strlist(),shift_shared_strlist(),count_shared_strlist() を追加した

REV A.1.30 2010/6/25

WEBPROXY サービスのWebAPI機能に /command/json/shared_data と /command/log を追加した

WEBPROXY サービスに Ajax 用 proxy 機能(/proxy)を追加した。

REV A.1.29 2010/6/9

WEBPROXY サービスのWebAPI機能に JSON 形式の応答フォーマットを追加した。

gps_coordinate_deg()ライブラリ関数を追加した。

REV A.1.28 2010/6/2

WebAPI機能 (XMLHttpRequestサーバー)を WEBPROXY サービスに追加した。

create_session(),delete_session()　ライブラリ関数を新規追加

XASDLCMD.DLL で提供するライブラリ関数から アラーム、UIOUSB 関連のものを削除した。

これらは XASDLCMD.DLLライブラリ中のスクリプト実行関数 SX_script_exec(), SX_script_exec2() を使用してスクリプト内から操作可能です。

REV A.1.27 2010/5/18

以下のライブラリ関数を新規追加

xbee_find_device,gps_utc_to_local(),gps_distance_course()

REV A.1.26 2010/4/30

ライブラリ関数 xbee_my_serial_number() が 16ビットアドレスも取得するようにした。

REV A.1.25 2010/4/18

スクリプトをサブフォルダ内に作成した場合の記述を追加した。

REV A.1.24 2010/3/29

スタンダードライセンスに XBee 機能を含めるようにした

REV A.1.23 2010/3/1

以下のライブラリ関数を新規追加

event_wait(),event_set(),critical_section_enter(),critical_section_leave()

REV A.1.22 2010/1/29

以下のライブラリ関数を新規追加

xbee_tdcp_safe_retry()

REV A.1.21 2009/12/24

XBEE_IO_DATA,XBEE_IO_OTHER,XBEE_PACKET_DATA,XBEE_TDCP_DATA イベントハンドラに渡されるパラメータに SerialNumber を追加した

REV A.1.20 2009/12/09

以下のライブラリ関数を新規追加

inc_day(),inc_second(),days_in_month(),add_shared_strlit(),clear_shared_strlit(),
get_shared_strlit(),add_session_strlit(),clear_session_strlit(),get_session_strlit(),
add_permanent_strlit(),clear_permanent_strlit(),get_permanent_strlit(),
remove_permanent_strlist(),channels_permanent_strlist()
add_all_session_strlit(),clear_all_session_strlit()

REV A.1.19 2009/11/03

clear_stat_data() の実行例サンプルの誤りを修正

day_of_week() ライブラリ関数を新規追加

REV A.1.18 2009/10/13

下記のスクリプトライブラリ関数を新規追加

get_session_data(),set_session_data(),set_all_session_data()

REV A.1.17 2009/10/7

イベント(XBEE_TDCP_DATA) を新規追加

REV A.1.16 2009/08/23

スクリプトを別スレッドで実行するための LUA ライブラリ関数(script_fork_exec)を新規追加