PMC 機器制御サーバ
取扱説明書

μTeaboard/ARM7-AT91 版

Version 1.00

目 次

修正履歴 **iii**

1 はじめに **1**
1.1 本製品の概要 1
1.2 本製品の内容 1
1.3 メモリーマップ 3

2 セットアップ **5**
2.1 インストール 5
2.2 起動方法 7
2.3 ファイルのアップロード 10
2.4 セキュリティー管理 11

3 制御内容の定義／カスタマイズ **14**
3.1 I/O 制御の概要 14
3.1.1 RS–232 以外の各 I/O 制御の概要 14
3.1.2 RS–232 制御の概要 16
3.2 I/O の種類とコマンド 18
3.2.1 PIO 18
3.2.2 A/D 変換 18
3.2.3 D/A 変換 19
3.2.4 PWM 19
3.2.5 スイッチ 21
3.2.6 LED 21
3.2.7 RS–232 22
3.2.8 共有変数 23
3.3 機器制御ライブラリ `io.js` 23
3.3.1 `io_init()` : I/O 初期化 24
3.3.2 `io_read()` : I/O 読み込み (ポーリング) 25
3.3.3 `io_wait()` : I/O 状態変化待ち (非同期) 25
3.3.4 `io_write()` : I/O 書き込み 26
3.3.5 `rs_init()` : RS–232 初期化 26
3.3.6 `rs_read()` : RS–232 読み込み 26
3.3.7 `rs_write()` : RS–232 書き込み 27
3.4 HTTP リクエスト一覧 27
3.4.1 `GET` : コンテンツ読み込み 28
3.4.2 `HEAD` : コンテンツのヘッダ読み込み 28
3.4.3 `POST /IO/INIT` : I/O 初期化 28
3.4.4 `POST /IO/READ` : I/O 読み込み (ポーリング) 29

3.4.5 POST /IO/WAIT : I/O 状態変化待ち (非同期) 29
3.4.6 POST /IO/WRITE : I/O 書き込み 30
3.4.7 POST /RS/INIT : RS–232 初期化 31
3.4.8 POST /RS/READ : RS–232 読み込み 31
3.4.9 POST /RS/WRITE : RS–232 書き込み 31
3.4.10 POST /FILE/LS : ディレクトリ一覧の取得 32
3.4.11 POST /FILE/UPLOAD : ファイルのアップロード 32
3.4.12 POST /FILE/UNLINK : ファイル削除 33
3.4.13 POST /FILE/MKDIR : ディレクトリ作成 33
3.4.14 POST /FILE/RMDIR : ディレクトリ削除 33
3.4.15 POST /FILE/SYNC : ファイルシステム保存 33

4 チュートリアル 35
4.1 LED 書き込み 35
4.2 スイッチ状態の読み込み 36
4.3 複数クライアントへの対応 38
4.4 画面の見栄えを追求するには 40

修正履歴

Version 1.00

- 新規作成

1 はじめに

1.1 本製品の概要

本製品「PMC 機器制御サーバ (μTeaboard/ARM7-AT91 版)」は、μTeaboard に接続したさまざまな機器をネットワークから制御するための汎用ソフトウェアです。以下のような特長があります。

- μTeaboard の デジタル I/O (PIO 等) や RS–232 ポートなどに接続した各種機器を制御します。
- エンドユーザはネットワークを通じてウェブブラウザから機器を制御できます。μTeaboard 側は機器制御機能つきのウェブサーバとして動作します。
- 具体的な制御内容やユーザインタフェースは、HTML や JavaScript で簡単に定義／カスタマイズすることができます。JavaScript や Ajax の手法を使うことで、さまざまな機器制御にご活用いただけます。μTeaboard 側を直接プログラミングする必要はありません。
- 複数のマシン上のウェブブラウザから同時にアクセスすることも可能です。従って例えば一つの機器を複数の箇所から同時にモニタリングすることができます。
- 非同期通信に対応しています。状態変化を非同期に通知する手法により、ウェブページ全体を再ロードすることなく機器の状態変化を表示できるとともに、ポーリングに比べて通信量を大幅に削減できます。
- 部外者にアクセスされたくない場合は、ユーザ名とパスワードによるセキュリティー制限を設けることが可能です。

1.2 本製品の内容

本製品「PMC 機器制御サーバ (μTeaboard/ARM7-AT91 版)」には以下の内容が含まれています。

ドキュメント

- 「PMC 機器制御サーバ (μTeaboard/ARM7-AT91 版) 取扱説明書」
  本書です。インストール方法からプロトコルの詳細まで、すべての内容をカバーします。

ソフトウェア

- `dioserv.mot`
  PMC 機器制御サーバ (サーバ内部で用いる DIO ドライバも含みます) をユーザ初期化プログラムとして起動する形式にしたものです。フラッシュ ROM 上に書き込んで使います。

- `html.zip`
  以下の内容を含む zip 形式のアーカイブファイルです。PC 上で展開した上で、必要なファイルを μTeaboard 側にアップロードして下さい。アップロードの方法につきましては「2.3 ファイルのアップロード方法」をご参照ください。

  - `/io.js`
    JavaScript 用の機器制御ライブラリです。機器制御で共通に必要となる関数群を定義しています。詳細は「3.3 機器制御ライブラリ io.js」をご参照ください。
  - `/test.html`
    機器制御のサンプル html ファイルです。以下の簡単な動作確認ができます。

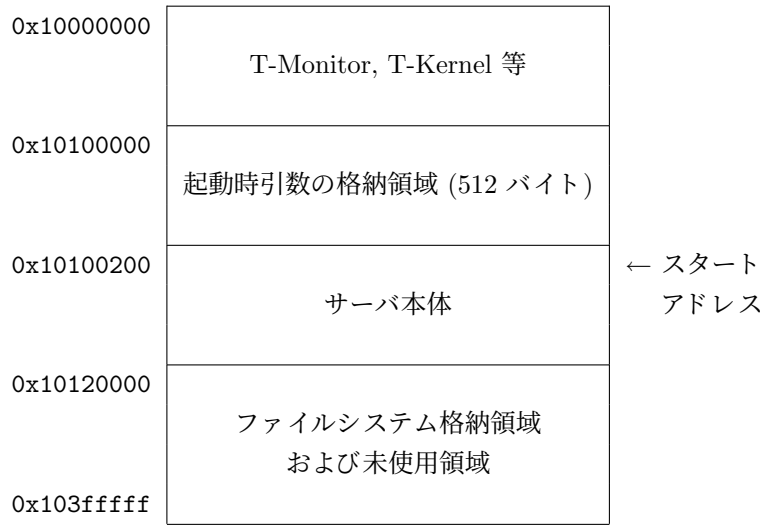
PIO 入力:	PB0(SW1), PB1(SW2) から入力
PIO 出力:	PB20(LED1), PB21(LED2) へ出力
A/D 変換:	AD7 (動作確認には外付け回路が必要です)
D/A 変換:	DA1 (動作確認には外付け回路が必要です)
PWM:	PWM7(LED3), PWM8(LED4) へ出力
スイッチ:	SW1, SW2, ディップスイッチから入力
LED:	LED1, LED2 へ出力
RS–232:	RS–232 ポートを PC などに接続して送受信

    † LED3,4 は、PIO で制御するか、PWM で制御するかのいずれか一方を選択できます。このサンプルは LED3,4 を PWM で制御して高速に点滅させる例です。
  - `/FILE/file.html`
    ファイルアップロード用の html ファイルです。ファイルをアップロードまたは削除する際に使います。詳細は「2.3 ファイルのアップロード方法」をご参照ください。
  - `/.passwd`
    パスワードファイルの例です。デフォルトでは特にアクセス制限は設けていませんので、アクセス制限を設ける場合は、このパスワードファイルを置き換えてください。詳細は「2.4 セキュリティー管理」をご参照ください。

– `/tutorial/sample1.html`, `sample2.html`, `sample3.html`, `sample4.html`
本書の「4 チュートリアル」で取り上げるサンプルです。

1.3 メモリーマップ

ROM



PMC 機器制御サーバでは、μTeaboard のフラッシュ ROM を次のように使います。

起動時引数の格納領域

PMC 機器制御サーバの起動時に指定するオプションを格納しておく領域です。起動時引数の詳細は「2.2 起動方法」をご参照ください。

サーバ本体

PMC 機器制御サーバ (サーバ内部で用いる DIO ドライバも含みます) を、ユーザ初期化プログラムとして起動する形式にしたものです。`dioserv.mot` を書き込んでください。スタートアドレスは 0x10100200 です。

ファイルシステム格納領域

μTeaboard 自身はファイルシステムを持ちませんが、PMC 機器制御サーバ は独自にファイルシステムを持っており、html や JavaScript 等のファイルをアップロードして格納することができます。ファイルアップロードの詳細は「2.3 ファイルのアップロード」をご参照ください。

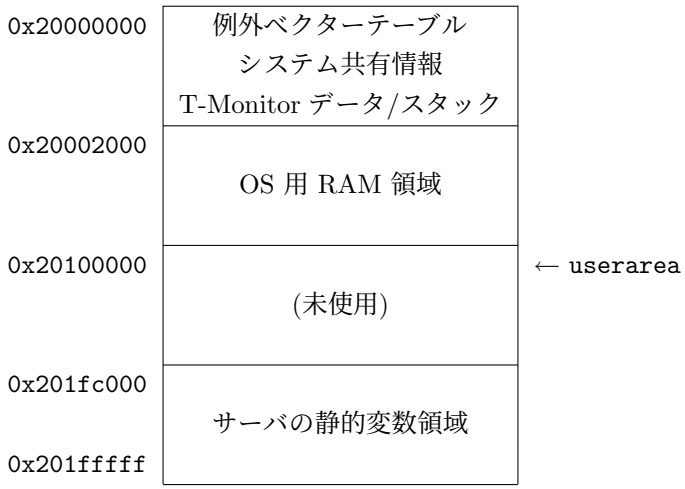
ファイルシステム格納領域の範囲は、`0x10120000` – `0x103fffff` の範囲内で、起動時引数で先頭アドレスとサイズを指定します。フラッシュ

ROM のブロックサイズである 0x10000 で割り切れる値を指定する必要があります。

ただしファイルシステム格納領域の大きさは OS 用の RAM 領域の大きさによっても制約を受けますので、全領域が指定できるわけではありません。用途に応じた必要最低限のサイズを指定するようにしてください。

PMC 機器制御サーバは、起動時にファイルシステム格納領域の内容をチェックして、ファイルシステムが正しく格納されているかどうかを確認します。もし格納されていない場合 (未格納の場合や、ファイルシステム格納領域の範囲を変更した場合などを含む) は、ファイルアップロードに必要な `/FILE/file.html` だけが格納された状態に自動的に初期化されます。

RAM



PMC 機器制御サーバ内部の静的変数は、μTeaboard の RAM のユーザエリアのうち、`0x201fc000` – `0x201fffff` を使います。従って、OS 用の RAM 領域を変更する場合は、これと重ならないようにしてください。

一方、これ以外に PMC 機器制御サーバ内部で動的に必要な RAM は、OS を通じて OS 用の RAM 領域から確保します。

OS 用の RAM 領域が足りない場合は、サーバ内で「メモリが確保できない」または「タスクが生成できない」などのエラーが発生し、エラーメッセージが RS–232 ポートに出力されます。その場合は、ROM 情報 (Rominfo, 先頭は 0x10010000) の `userarea` フィールド (オフセット +0x0c) を書き換えて OS 用の RAM 領域を増やしてください。

2 セットアップ

2.1 インストール

1. μTeaboard と PC を RS–232 で接続して、gterm や TeraTerm などの端末ソフトを用いて T-Monitor 上で操作します。

   (IMS のプロンプトが表示されている場合は、まず T-Monitor を起動してください。詳細は μTeaboard の取扱説明書をご参照ください。)

2. 本製品に含まれるサーバ本体のファイル `dioserv.mot` をフラッシュ ROM 上に転送します。

   端末ソフトとして gterm を使用している場合は、以下のように入力して下さい。(カレントディレクトリ上に `dioserv.mot` がない場合は、`dioserv.mot` へのパスを指定して下さい)

   ```
   TM> .flload dioserv.mot↩
   ```

   一方、端末ソフトとして TeraTerm を使用している場合は、以下のように入力してから、File メニュー → Transfer → XMODEM → Send で、ダイアログから転送すべきファイル `dioserv.mot` を指定してください。

   ```
   TM> FlashLoad ex ↩
   ```

3. 起動時引数をフラッシュ ROM 上の 0x10100000 から ASCII 文字列で格納します。起動時引数の詳細は「2.2 起動方法」をご参照ください。以下はネットワーク環境を設定する一例です。(以下では文字列内の空白を ␣で表記します)

   ```
   TM> mov 0x10100000,#0x10000,0x20100000↩
   TM> mb 0x20100000,"-ipadr=192.168.0.100"↩
   TM> mb ,"␣-dnsadr=192.168.0.10"↩
   TM> mb ,"␣-domain=xxxxxxx"↩
   TM> mb ,"␣-subnet=255.255.255.0"↩
   TM> mb ,0↩
   TM> wrom 0x10100000,0x20100000,1↩
   ```

   重要 T-Monitor の `wrom` コマンドを用いる際は、アドレス指定を間違えないように充分ご注意下さい。特に 0x10000000 – 0x10100000 の範囲は T-Monitor や T-Kernel が格納されており、ここに上書きしてしまうとシステムが起動できなくなりますので、上書きしないようご注意下さい。

4. μTeaboard のシステム起動時にサーバが自動的に起動するように、ユーザ初期化プログラムとしてサーバのスタートアドレスである 0x10100200 を登録します。

```
TM> mov 0x10010000,#0x10000,0x20100000↩
TM> mw 0x20100000+0x10,0x10100200↩
TM> wrom 0x10010000,0x20100000,1↩
```

重要 T-Monitor の wrom コマンドを用いる際は、アドレス指定を間違えないように充分ご注意下さい。

5. サーバを起動するには、システムを再起動します。

```
TM> exit -1↩
PMC T-Kernel/TBAT91 Version 1.0.00
Copyright (C) 2007 by Personal Media Corporation

** MemoryMgr OK
      ⋮
** TcpIpMgr OK
PMC Device Control Server/TBAT91 Version 1.00
Copyright (C) 2007 by Personal Media Corporation
```

T-Kernel の起動メッセージ、および PMC 機器制御サーバの起動メッセージが出力されれば正常です。(バージョン番号や年号は上記と異なる場合があります)

† 再起動後にもし T-Monitor のプロンプト (TM>) が出るようであれば、bd↩ にて OS をブートしてください。
なお μTeaboard のディップスイッチの設定により、自動的にブートすることも可能です。詳細は μTeaboard の取扱説明書をご参照ください。

† 再起動後にもし IMS のプロンプト ([IMS]%) が出るようであれば、ユーザ初期化プログラムの登録がうまくいっていないと思われますので、再度ご確認ください。

† サーバが起動していることを確認するには、他のマシンから μTeaboard に対して ping を投げてみて下さい。ping の応答がない場合は、サーバが起動していない、または起動時引数のネットワーク設定が正しくない、といった可能性が考えられます。

6. この時点ではサーバのファイルシステム上には、ファイルアップロードに必要な/FILE/file.html だけが存在します。そこでウェブブラウザからここにアクセスして、正常にアクセスできることを確認します。

   具体的な URL はネットワーク環境によって異なりますが、例えば μTeaboard の IP アドレスが 192.168.0.100 の場合は、

   http://192.168.0.100/FILE/file.html

   にアクセスしてください。

7. 本製品に含まれる HTML/JavaScript のアーカイブ html.zip を PC 上で展開して、その中から例えば次の二つのファイルをサーバのルート直下にブラウザからアップロードしましょう。

   - /test.html
     機器制御のサンプル html ファイルです。
   - /io.js
     JavaScript 用の機器制御ライブラリです。test.html を使うには、こちらのライブラリも必要です。

   なおファイルアップロードの詳細は「2.3 ファイルのアップロード」をご参照ください。

8. いまアップロードした /test.html にブラウザからアクセスして、正常にアクセスできることを確認します。

   具体的な URL はネットワーク環境によって異なりますが、例えば μTeaboard の IP アドレスが 192.168.0.100 の場合は、

   http://192.168.0.100/test.html

   にアクセスしてください。

2.2 起動方法

自動起動

μTeaboard のシステム起動時にサーバを自動起動するように設定するには、次のように、ユーザ初期化プログラムとして 0x10100200 を ROM 情報 (Rominfo, 先頭は 0x10010000) の userinit フィールド (オフセット +0x10) に登録しておきます。

```
TM> mov 0x10010000,#0x10000,0x20100000↩
TM> mw 0x20100000+0x10,0x10100200↩
TM> wrom 0x10010000,0x20100000,1↩
```


重要 T-Monitor の wrom コマンドを用いる際は、アドレス指定を間違えないように充分ご注意下さい。

† なお、上記のユーザ初期化プログラムの登録を解除してシステム起動時に IMS が起動するように戻すには、次のようにします。

```
TM> mov 0x10010000,#0x10000,0x20100000↩
TM> mw 0x20100000+0x10,0↩
TM> wrom 0x10010000,0x20100000,1↩
```

起動時引数

サーバを起動する際に、以下の起動時引数を指定することができます。

起動時引数は、フラッシュ ROM 上の 0x10100000 から ASCII 文字列で格納して下さい。複数のオプションは空白 (ASCII コード 0x20) で区切り、末尾はヌル文字 (ASCII コード 0x00) で終了して下さい。

- -ipadr=μTeaboard の IP アドレス
  -dnsadr=DNS の IP アドレス
  -domain=ドメイン名
  -subnet=サブネットマスク

  ネットワーク環境を設定します。

  指定しない場合は -ipadr=0.0.0.0, -dnsadr=指定なし, -domain=指定なし, -subnet=255.255.255.0 となります。

- -file-addr=ファイルシステム格納領域の先頭アドレス
  -file-addr=ファイルシステム格納領域のサイズ

  ファイルシステム格納領域として使用するフラッシュ ROM の先頭アドレスとサイズ (単位:バイト) を指定します。どちらもフラッシュ ROM のブロックサイズである 0x10000 で割り切れる値を指定する必要があります。

  指定しない場合は -file-addr=0x10120000, -file-size=0x10000 となります。

  ファイルシステム格納領域として利用できる範囲の詳細については「1.3 メモリーマップ」をご参照ください。

- -port=ポート番号

  サーバの使用する HTTP のポート番号を指定します。
  指定しない場合は 80 となります。

- -keep-alive=タイムアウト時間

サーバの持続的な接続 (keep-alive) のタイムアウトをミリ秒単位で指定します。
指定しない場合は 10000 (10 秒間) となります。

HTTP リクエストに対する応答が完了した後も、サーバ側は接続を切断せずに保持します。クライアント側は同じ接続を用いて次の HTTP リクエストを出すことができます。もし新たな HTTP リクエストが来ないままタイムアウトとして指定した時間が経過すると、その時点でサーバ側は接続を切断します。

なおタイムアウトとして `0` を指定すると、サーバ側は応答完了時にすぐ接続を切断します。また `-1` を指定すると、サーバ側からは接続の切断を行いません。

- `-max-channel=`最大チャンネル番号

  I/O を監視するチャンネルの最大番号を指定します。
  指定しない場合は 4 となります。

- `-io-period=I/O` 監視間隔

  サーバ内部で I/O を監視する間隔をミリ秒単位で指定します。
  指定しない場合は 100(100 ミリ秒間隔) となります。

以下に上記の起動時引数のいくつかを格納する例を示します:

```
TM> mov 0x10100000,#0x10000,0x20100000↩
TM> mb 0x20100000,"-ipadr=192.168.0.100"↩
TM> mb ,"␣-dnsadr=192.168.0.10"↩
TM> mb ,"␣-domain=xxxxxxx"↩
TM> mb ,"␣-subnet=255.255.255.0"↩
TM> mb ,"␣-file-addr=0x10120000"↩
TM> mb ,"␣-file-size=0x20000"↩
TM> mb ,0↩
TM> wrom 0x10100000,0x20100000,1↩
```

重要 T-Monitor の `wrom` コマンドを用いる際は、アドレス指定を間違えないように充分ご注意下さい。特に 0x10000000 – 0x10100000 の範囲は T-Monitor や T-Kernel が格納されており、ここに上書きしてしまうとシステムが起動できなくなりますので、上書きしないようご注意下さい。

RS–232 へのメッセージの抑制

システム起動時には PMC 機器制御サーバの起動メッセージが RS–232 ポートに出力されます。また、サーバ内で何らかのエラーが発生した場合、エラーメッセージも RS–232 ポートに出力されます。

RS–232 ポートを制御する場合など、これらのメッセージを抑制する必要がある場合には、RS–232 ポートをデバッグポートとして使用しないように設定してください。

- DEVCONF の中で `DEBUGMODE 0` としてください。
- その DEVCONF を用いて ROM 情報 (RomInfo) を作成しなおして、μTeaboard のフラッシュ ROM に書き込んでください。

詳細につきましては、μTeaboard のマニュアルをご参照ください。

2.3 ファイルのアップロード

URL とサーバ上のファイルとの対応関係

ウェブサーバとしてブラウザ側に発信するためのコンテンツファイルは、ファイルシステム格納領域の / 以下に格納されています。
ウェブブラウザ側から URL として例えば `http://サーバアドレス/xxx/yyy` を指定してアクセスすれば、それに対応するファイル /xxx/yyy の内容が返されます。

ファイルアップロード のインタフェース

サーバ側のファイルをアップロードしたり削除したりするには、ウェブブラウザからファイルをアップロードします。

ウェブブラウザから `/FILE/file.html` にアクセスすれば、ウェブブラウザ上の簡単なボタン操作によって、ファイルをアップロードしたり削除したりすることができます。

- ファイルのアップロード :
  アップロードしたい PC 上のファイルを選択して、[アップロード] ボタンをクリックします。
- ファイルの削除:
  削除したいファイルの [削除] ボタンをクリックします。

具体的な URL はネットワーク環境によって異なりますが、例えば μTeaboard の IP アドレスが 192.168.0.100 の場合は、

`http://192.168.0.100/FILE/file.html`

にアクセスしてください。

ただし、セキュリティー運用上、限られた管理者だけがここをアクセス可能なように、/FILE 以下にアクセス制限を設けることが望ましいです。詳細は「2.4 セキュリティー管理」をご参照ください。

フラッシュROM へのファイルシステム保存のタイミング

ファイルをアップロードしたり削除した場合、それらはすぐにフラッシュROM に保存されるのではなく、まず RAM 上に保存され、システムをシャットダウンまたは再起動するまでは有効です。

フラッシュ ROM への保存は、/FILE/file.html の 「ファイルシステム保存」ボタンを押すことによって明示的に行います。

フラッシュ ROM への書き込み頻度を必要最低限にするためには、ファイルをアップロードしてからすぐにファイルシステムを保存するのではなく、アップロードしたファイルの動作確認を充分に行った後、必要に応じて /FILE/file.html にアクセスして、ファイルシステムを保存するようにしてください。

2.4 セキュリティー管理

ユーザ名とパスワードによる認証を設けることにより、セキュリティー管理を行うことが可能です。認証プロトコルとしては HTTP 仕様で定められた「基本認証 (Basic 認証)」を用います。

パスワード ファイルの所在

パスワード ファイルをサーバのルートディレクトリ直下の /.passwd にアップロード することで、認証関連の設定を行うことができます。

パスワード ファイルのアップロード の方法は、通常のファイルと同じです。詳細は「2.3 ファイルのアップロード 方法」をご参照ください。

新しいパスワード ファイルが有効になるタイミング

新しいパスワード ファイルをウェブブラウザからアップロード した場合、アップロードが完了した直後から新しいパスワード ファイルが有効になります。

パスワード ファイルの形式

パスワード ファイルはテキスト ファイルで、その各行は

<ユーザ名>:<パスワード >:<認証領域>:<パス名>

という形式です。

ただし行頭が # で始まる場合はコメント 行として無視されます。

ユーザ名は省略可能です。その場合はユーザ名は何でもよく、パスワードのみで判断されます。

認証領域 (realm) は、認証の単位を識別する名称で、自由につけることができます。認証が必要な際には、ブラウザ側はここで指定された認証領域を表示して、それに対するユーザ名とパスワード の入力を求めます。

認証の範囲は、指定されたパス名に対して、その子孫全体に及びます。ただし複数の指定が同時に該当する場合は、パスがより長い (深い) 方が優先されます。

以下にパスワードファイルの例を示します:

例 1 一般ユーザのアクセスは制限せず、ファイルアップロードは管理者だけができるように制限する場合:

```
root:26cxowoa:Admin:/FILE
```

このパスワードファイルを設置した場合、一般ユーザとしてウェブブラウザから/ 以下 (/FILE を除くすべての子孫を含む) にアクセスする際には認証制限はありません。一方、管理者としてウェブブラウザからファイルアップロードを行おうとする際には、Admin に対するユーザ名とパスワードを聞かれますので、root というユーザ名でパスワード 26cxowoa を入れる必要があります。

例 2 一般ユーザのアクセスと、管理者によるファイルアップロードを、それぞれ制限する場合:

```
taro:k8kqzp1k:General:/
hanako:cw3qquw6:General:/
root:26cxowoa:Admin:/FILE
```

このパスワードファイルを設置した場合、一般ユーザとしてウェブブラウザから/ 以下 (/FILE を除くすべての子孫を含む) にアクセスしようとする際には、General に対するユーザ名とパスワードを聞かれますので、taro というユーザ名でパスワード k8kqzp1k を入れるか、または hanako というユーザ名でパスワード cw3qquw6 を入れる必要があります。また、管理者としてウェブブラウザからファイルアップロードを行おうとする際には、Admin に対するユーザ名とパスワードを聞かれますので、root というユーザ名でパスワード 26cxowoa を入れる必要があります。

パスワードファイルに関するセキュリティー

いったんアップロードしたパスワードファイルは、セキュリティー上、ウェブブラウザで仮に /.passwd を閲覧しようとしても閲覧できないようになっています。

従っていったんウェブブラウザ経由でアップロードしたパスワードファイルを再度ウェブブラウザ経由でダウンロードすることはできません。そこでパスワードファイルのコピーを手元に残しておきたい場合は、限られた管理者のみが見られる安全な場所にバックアップするようにしてください。

注意事項

本製品では上記の通り、ユーザ名とパスワードを用いた基本認証によってアクセス制限を設けることができます。しかし通信路上の通信そのものが暗号化されるわけではありませんので、ご注意ください。

3 制御内容の定義／カスタマイズ

3.1 I/O 制御の概要

この節では、各 I/O 制御に共通するいくつかの基本概念について説明します。RS–232 以外の各 I/O の場合と RS–232 の場合では概念が別になりますので、分けて説明します。

ここで説明するコマンドは、機器制御ライブラリ `io.js` を使って発行するか、または直接 HTTP リクエストで発行します。発行の具体的な方法につきましては、それぞれ「3.3 機器制御ライブラリ `io.js`」または「3.4 HTTP リクエスト一覧」をご参照ください。また、具体的なコマンドの詳細につきましては「3.2 I/O の種類とコマンド」をご参照下さい。

3.1.1 RS–232 以外の各 I/O 制御の概要

初期化コマンド

I/O の初期化は「初期化コマンド」を発行することで行います。

初期化コマンドは「変数=初期設定値」という形をしています。

読み込みコマンドや書き込みコマンドを発行する前に、必ず初期化コマンドを発行して、その I/O を初期化しておくことが必要です。

複数のコマンドを並べる場合は「;」(セミコロン) で区切ります。並べた最後のコマンドの後ろには「;」はあってもなくても構いません。

読み込みコマンド

I/O の読み込みは「読み込みコマンド」を発行することで行います。

読み込みコマンドは「変数 : ビット幅」という形をしています。

複数のコマンドを並べる場合は「;」(セミコロン) で区切ります。並べた最後のコマンドの後ろには「;」はあってもなくても構いません。

読み込み結果は、バイト列として格納され、16 進文字列で返されます。

格納される順序は、最上位ビット (MSB) から順に、最下位ビット (LSB) に向かいます。

複数バイトにまたがる場合は上位バイトから先に格納されます (つまりビッグエンディアンで格納されます)。

また、その変数の幅より大きな幅を指定した場合は、余った上位ビットは 0 が詰められます。

一方、その変数の幅より小さな幅を指定した場合は、足りない上位ビットは格納されません。

なお、「0: ビット幅」とすると、指定したビット数のゼロが詰められます。

(例)


```
0:4; PA1:1; PA2:1; PA3:1; PA4:1; PA5:4; PA6:4; $1:16; $2:32
```

1バイト目: 上位4ビットは0,
下位4ビットは上から順に PA1, PA2, PA3, PA4
2バイト目: 上位4ビットは PA5, 下位4ビットは PA6
3〜4バイト目 $1 の下位16ビットがビッグエンディアンで格納される
5〜8バイト目 $2 がビッグエンディアンで格納される

よって結果は計8バイト、つまり16進数16桁で返されます。例えば PA1 = 1, PA2 = 1, PA3 = 0, PA4 = 0, PA5 = 1, PA6 = 0, $1 = 0x12345678, $2 = 0xfedcba98 の場合は、結果は「0C105678FEDCBA98」となります。

書き込みコマンド

I/O の書き込みは「書き込みコマンド」を発行することで行います。

書き込みコマンドは「変数=書き込み値」という形をしています。

複数のコマンドを並べる場合は「;」(セミコロン) で区切ります。並べた最後のコマンドの後ろには「;」はあってもなくても構いません。

状態変化の非同期通知

本製品では、I/O 状態を監視する方法として、読み込みコマンドを毎回発行してポーリングで現在状態を取得する方法だけでなく、サーバ側にいったん状態監視要求を出すと、サーバ側で状態が変化するまで待って、状態が変化した時点で非同期にクライアント側に通知するしくみを用意しております。

状態の変化を監視するには、あらかじめ「どの I/O の状態を監視するのか」という状態監視用の読み込みコマンドをサーバ側に指定しておく必要があります。さらにクライアント側は、状態監視要求を出す際に毎回「比較対象となる状態」を指定して、サーバ側に状態監視を依頼します。サーバ側は、指定された I/O の現在状態を監視して、その結果が比較対象となる状態と異なることを検出した時点で、クライアント側に変化を通知します。

† クライアント側から比較対象となる状態をサーバ側に渡すようにしているのは、次の例のように、クライアントが状態監視要求を出してからその要求がサーバ側に届くまでの間に状態が変化した場合でも、その状態変化を取りこぼさずにクライアントに通知することを保証するためです。

(例) SW2 の状態を監視したい場合、状態監視用の読み込みコマンドとして例えば「SW2:1」をあらかじめサーバ側に登録しておきます。

さて、SW2 の現在状態が「0」の時、クライアント側からサーバ側に、比較対象として現在状態である「0」(の1バイトの16進表記の「00」) を指定して、サーバ側に状態監視要求を出す場合を考えます。

ここで、この状態監視要求がサーバ側に届いた時点では、既に SW2 の状態が「0」から「1」に変化していたと仮定します。

しかしこの場合でもサーバ側はクライアント側から渡された比較対象の「0」と SW2 の現在の状態である「1」を比較して、異なることが分かるため、ただちに変化後の状態「1」をクライアント側に返すことができます。

チャンネル

上記の通り、状態の変化を監視するには、あらかじめ「どの I/O の状態を監視するのか」という状態監視用の読み込みコマンドをサーバ側に指定しておく必要があります。

この目的のために設けられているのが「チャンネル」という概念です。すなわち、指定したチャンネルごとに、あらかじめ状態監視用の読み込みコマンドを登録しておくことができます。

状態監視要求の際に、クライアント側はチャンネル番号を指定すれば、サーバ側はあらかじめそのチャンネルに登録された読み込みコマンドを用いて状態を監視します。

チャンネル番号は 1 以上 4 以下の番号で指定します (起動時引数 -max-channel で最大チャンネル番号は変更できます)。

独立した二つ以上の処理を同じサーバで行わせる場合、あらかじめ異なるチャンネルに対してそれぞれの状態監視用の読み込みコマンドを別々に登録しておけば、お互いに干渉しあうことはありません。

なお、登録できる状態監視用の読み込みコマンドは、一つのチャンネルにつき一つだけです。もし同一のチャンネルに複数登録した場合は、後から登録した方が有効になり、前に登録していた方は無効になります。

3.1.2 RS–232 制御の概要

一方、RS–232 の制御は他の I/O の制御とは異なり、チャンネルという概念はありません。

RS–232 の場合は、ポート番号を指定して初期化コマンドを発行した後、そのポートに対して読み込みコマンドや書き込みのコマンドを発行する形になります。

読み込みコマンドや書き込みコマンドは、完了した時点で非同期に結果がクライアント側に通知されます。

RS–232 の読み込みおよび書き込みは、それぞれ事前の初期化コマンド発行の際にタイムアウトの時間を指定することができます。

読み込みコマンドを発行して、指定した文字数が読み込める前にタイムアウトした場合は、そこまでに読み込めた文字列を返します。

同様に書き込みコマンドを発行して、指定した文字数が書き込める前にタイムアウトした場合は、そこまでに書き込めた文字数を返します。

3.2 I/O の種類とコマンド

この節では、μTeaboard/ARM7-AT91 で制御できる I/O の種類ごとに、具体的なコマンドの使い方を説明します。

なお各信号の詳細やコネクタのピン配置などについては、μTeaboard のハードウェアマニュアルをご参照下さい。

3.2.1 PIO

PIO(Parallel Input/Output) はビット単位のデジタル入出力です。まず初期化時に入力として使うか出力として使うかを指定した後に、読み込みや書き込みを行います。

μTeaboard/ARM7-AT91 では、以下の PIO が使えます。

変数名	備考
PA0 〜 PA31	PIO A0 〜 A31
PB0 〜 PB31	PIO B0 〜 B31

† PB0, PB1 はプッシュスイッチ (入力のみ) (0:押下 / 1:解放)
3.2.5 で述べる SW1, SW2 とは論理が逆になります

† PB22, PB25, PB26, PB27 は DIP スイッチ (入力のみ) (1:ON / 0:OFF)

† PB20, PB21, PB23, PB24 は LED (出力のみ) (0:点灯 / 1:消灯)
LED は 3.2.6 で述べる LED1 〜 LED4 で制御することもできます。ただし点灯と消灯の論理は逆です。

コマンドの形式は次の通りです。

初期化コマンド	入力として使う場合：PXn = IN 出力として使う場合：PXn = 初期値 (例) PA1 = IN; PA2 = IN; PA3 = 0; PA4 = 1
読み込みコマンド	PXn:ビット幅 (例) PA1:1; PA2:1
書き込みコマンド	PXn = 代入値 (例) PA3 = 1; PA4 = 0

3.2.2 A/D 変換

アナログ信号入力を離散値に変換します。

μTeaboard/ARM7-AT91 では、以下の A/D 変換が使えます。

変数名	備考
AD0 – AD3	A/D コンバータ ADC0 (10bit) の ch0 – ch3
AD4 – AD7	A/D コンバータ ADC1 (10bit) の ch0 – ch3

コマンドの形式は次の通りです。

初期化コマンド	ADn = A,B,C, 計測周期 A,B,C: 補正パラメータ (例) AD1 = 0,100,1023,1000
読み込みコマンド	ADn:ビット幅 (例) AD1:8
書き込みコマンド	(書き込み不可)

- A,B,C は補正パラメータです。
  AD 値 x は $(x + A) \times B \div C$ として読み込まれます。
- 「計測周期」は、計測する周期をミリ秒単位で指定します。一般に AD 値は測定毎にゆらぎが生じるため、計測周期を大きく取ることで、ゆらぎの影響を小さくできます。

3.2.3 D/A 変換

離散値をアナログ信号出力に変換します。

μTeaboard/ARM7-AT91 では、以下の D/A 変換が使えます。

変数名	備考
DA0,DA1	D/A コンバータ (10bit)

コマンドの形式は次の通りです。

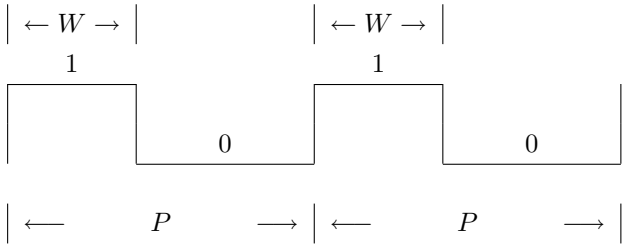
初期化コマンド	DAn = A,B,C, 初期値 初期値 x は $(x + A) \times B \div C$ として出力します (例) DA0 = 0,1023,100,50
読み込みコマンド	DAn:ビット幅 直前に指定された初期値または代入値を返します (例) DA0:8
書き込みコマンド	DAn = 代入値 代入値 x は $(x + A) \times B \div C$ として出力します (例) DA0 = 30

3.2.4 PWM

PWM (Pulse Wave Modulator) は、任意のパルスを生成する機能であり、一般に次のような構造になっています:

$$\boxed{C} \rightarrow \boxed{\frac{1}{D}} \rightarrow \boxed{\frac{1}{S}} \rightarrow \boxed{P\text{進カウンタ}} \rightarrow \boxed{W\text{と比較}} \rightarrow \text{パルス出力}$$

クロックソース C を分周器 D およびプリスケーラ S で分周して、カウンタで数えます。カウンタの値は周期 P で繰り返します。カウンタの値をパルス幅 W と比較して、出力が 0 か 1 かを決定します。出力されるパルスの周波数は $C \div (D \times S \times P)$ となり、波形に占める 1 の幅の割合 (デューティー比) は $W \div P$ になります。



μTeaboard/ARM7-AT91 では、タイマーを用いて以下の PWM を実現します。(なおプリスケーラは存在しないので 1 固定です。)

変数名	備考
PWM1	TIOA3 (PA1 と共有)
PWM2	TIOB3 (PA2 と共有)
PWM3	TIOA4 (PA4 と共有)
PWM4	TIOB4 (PA5 と共有)
PWM5	TIOA5 (PA7 と共有)
PWM6	TIOB5 (PA8 と共有)
PWM7	TIOA1 (PB23 と共有, LED3 (1:消灯, 0:点灯))
PWM8	TIOB1 (PB24 と共有, LED4 (1:消灯, 0:点灯))

† PWM1 と PWM2, PWM3 と PWM4, PWM5 と PWM6, PWM7 と PWM8 は、それぞれタイマーを共用しているため、パルス幅 W は独立して指定できますが、分周比 D と周期 P はそれぞれ同じ値を指定しなければなりません。

† LED3,4 を PWM で制御する場合は、PIO では制御できません。

† TIOA0 と TIOB0 は, μTeaboard では T-Kernel のシステムタイマーとして使っているため、PWM としては使用できません。

† TIOA2 と TIOB2 は, μTeaboard では DIP スイッチ入力に接続されているため、PWM としては使用できません。

コマンドの形式は次の通りです。

初期化コマンド	PWMn = C,D,S,P,W C: クロックソース周波数 (32000000 (32MHz) 固定) D: 分周比 (2, 8, 32, 128, 1024 のいずれか) S: プリスケール (1 固定) P: 周期 (2 以上 65535 以下) W: パルス幅 (0 以上 P 以下) (例) PWM1 = 32000000,128,1,2500,0 (パルス周波数 $32000000 \div (128 \times 1 \times 2500) = 100$ Hz)
読み込みコマンド	PWMn:ビット幅 直前に指定されたパルス幅を返し ます (例) PWM1:16
書き込みコマンド	PWMn = W W: パルス幅 (0 以上 P 以下) (例) PWM1 = 625 (デューティー比は $625 \div 2500 = 0.25 = 25\%$)

3.2.5 スイッチ

ボード上のスイッチです。ビット単位のデジタル入力です。

μTeaboard/ARM7-AT91 では、以下のプッシュスイッチや DIP スイッチから入力することができます。

変数名	備考
SW1 〜 SW2	プッシュスイッチ (SW1, SW2) (1:押下 / 0:解放)
SW11 〜 SW14	DIP スイッチ (DSW1–1,2,3,4) (1:ON / 0:OFF)

コマンドの形式は次の通りです。

初期化コマンド	SWn = IN (例) SW1 = IN; SW2 = IN
読み込みコマンド	SWn:ビット幅 (例) SW1:1; SW2:1
書き込みコマンド	(書き込み不可)

3.2.6 LED

ボード上の LED です。ビット単位のデジタル出力です。

μTeaboard/ARM7-AT91 では、以下の LED を PIO で制御します。

変数名	備考
LED1 〜 LED4	汎用 LED(LED1〜LED4) (1:点灯 / 0:消灯)

コマンドの形式は次の通りです。0 は消灯、1 は点灯を表します。

初期化コマンド	LEDn = 初期値
	(例) LED1 = 0; LED2 = 1
読み込みコマンド	LEDn:ビット幅
	(例) LED1:1; LED2:1
書き込みコマンド	LEDn = 代入値
	(例) LED1 = 1; LED2 = 0

3.2.7 RS–232

RS–232 ポートです。

μTeaboard/ARM7-AT91 では、以下の RS–232 ポートが使えます。

ポート番号	備考
0	シリアルポート 0

RS–232 については他の I/O とはコマンドの形式が異なります。コマンドの形式は次の通りです。

初期化コマンド

ポート番号を指定して、以下の初期化コマンドを発行します。

PAR = パリティ (0:なし, 1:偶数, 2:奇数)
LEN = 文字長 (7 または 8)
STOP = ストップビット長 (0: 1 ビット, 1: 1.5 ビット, 2: 2 ビット)
BAUD = ボーレート (単位は bps)
CS = CS 制御 (0:制御なし, 1:制御あり)
RS = RS 制御 (0:制御なし, 1:制御あり)
SX = 送信 XON/XOFF 制御 (0:制御なし, 1:制御あり,
3: 制御あり, 任意文字で XON)
RX = 受信 XON/XOFF 制御 (0:制御なし, 1:制御あり)
STMO = 送信タイムアウト (単位はミリ秒, −1 はタイムアウトなし)
RTMO = 受信タイムアウト (単位はミリ秒, −1 はタイムアウトなし)

(例) PAR=0;LEN=8;STOP=0;BAUD=9600;STMO=500;RTMO=500

読み込みコマンド

ポート番号と読み込み文字数を指定して、コマンドを発行します。

書き込みコマンド

ポート番号と書き込み文字列を指定して、コマンドを発行します。

ここで読み込み文字列および書き込み文字列は、文字コードを 16 進数で「%xx」で指定する形式で表します。ただし制御文字と % 自身を除く ASCII 文字 (%20 〜 %7E) はそのまま表しても構いません。

(例) `Hello%0D%0A`　　「H」「e」「l」「l」「o」「復帰」「改行」の 7 文字を表します。

3.2.8 共有変数

ユーザが自由に使える共有変数 `$1`, ..., `$8` が用意されており、サーバ側で値を保持します。

この共有変数を使えば、例えば複数のクライアント (ウェブブラウザ) 間で値を共有したり受け渡したりすることが可能です。

共有変数の値の範囲は 32 ビット整数です。

コマンドの形式は次の通りです。

初期化コマンド	`$n = 初期値` (例) `$1 = 0; $2 = 100`
読み込みコマンド	`$n:`ビット幅 (例) `$1:32; $2:16`
書き込みコマンド	`$n = 代入値` (例) `$1 = 0xffffffff; $2 = 1234`

3.3 機器制御ライブラリ `io.js`

機器制御ライブラリ `io.js` を使うと、「3.2 I/O の種類とコマンド」で説明されたコマンドを JavaScript から簡単に発行することができます。

RS–232 以外の I/O と RS–232 は別系統のライブラリ関数になっています。

同一マシン上のブラウザからの同時接続数に関する制限事項

ここで説明する各ライブラリ関数は、ぞれぞれブラウザの HTTP 接続を 1 つずつ使います。

通常、同一マシン上のブラウザの HTTP の同時接続数は 2 までに制限されています。従って、同一マシン上のブラウザから同時に 3 回以上のライブラリ関数を発行することはできません。

一方、異なるマシン上でそれぞれブラウザを起動して、その中から各ライブラリ関数を使用するぶんには、この制限はありません。

(例 1) ブラウザのウィンドウを一つ開いて、その中で `io_wait()` で状態変化待ちをしている間に、`io_write()` で I/O 書き込みを行うことは可能です。

(例 2) しかし、同じマシン上でブラウザのウィンドウを二つ開いて、その中でそれぞれ `io_wait()` で状態変化待ちをしている間は、さらに `io_write()` で I/O 書き込みを行うことはできません。

(例 3) 一方、異なる二つのマシン上でブラウザのウィンドウをそれぞれ一つずつ開いて、その中でそれぞれ io_wait() で状態変化待ちをしている間に、さらに io_write() で I/O 書き込みを行うことは可能です。

3.3.1 io_init() : I/O 初期化

書式: io_init(チャンネル , 初期化コマンド ,
状態監視用の読み込みコマンド , コールバック関数)

戻値: 成功した場合は null 以外の値を返します。

失敗した場合は null を返します。

説明: 指定したチャンネルに対して、初期化コマンドを発行するとともに、状態監視用の読み込みコマンドを登録します。

チャンネル番号は 1 以上 4 以下の番号で指定します (起動時引数 -max-channel で最大チャンネル番号は変更できます)。独立した二つ以上の処理を同じサーバで行わせる場合は、異なるチャンネルを選ぶことにより、お互いに干渉しないようにすることができます。

初期化コマンドは省略することができます。その場合は null を指定して下さい。省略した場合、初期化は行われません。

状態監視用の読み込みコマンドは省略することができます。省略する場合は null を指定して下さい。省略した場合、それ以前に同じチャンネルに登録されていた状態監視用の読み込みコマンドが引き続き有効になります。(なお一度も登録していない場合は、状態監視用の読み込みコマンドは未定義状態となりますので、状態監視はできません)

コールバック関数が指定されている場合は、状態監視用の読み込みコマンドを用いて現在状態を取得し、その結果を 16 進文字列でコールバック関数に渡します。コールバック関数を指定しない場合は null を指定してください。

既に同じチャンネルに対して以前に初期化コマンドが発行されていた場合は、以前に発行した初期化コマンドと今回の初期化コマンドを比較して、もしも同じ場合には、今回の初期化コマンドは無視され、初期化は行われません。これはブラウザの再読み込みなどのときに、同じ初期化が再度行われることを防ぐためです。一方、初期化コマンドが異なる場合は、今回指定した初期化コマンドによって再度初期化が行われます。

既に同じチャンネルに対して以前に状態監視用の読み込みコマンドが登録されていた場合は、後から登録した方が有効になり、前に登録していた方は無効になります。

3.3.2 io_read() : I/O 読み込み (ポーリング)

書式:

同期型: io_read(チャンネル, 読み込みコマンド)

非同期型: io_read(チャンネル, 読み込みコマンド, コールバック関数)

戻値:

同期型: 成功した場合は結果を 16 進文字列で返します。失敗した場合は null を返します。

非同期型: 成功した場合は、null 以外の値を返します。
失敗した場合は null を返します。

説明: 読み込みコマンドを発行し、結果を受け取ります。結果は 16 進文字列で返されます。

同期型の場合は、結果が返ってくるまで待って、その結果を文字列で戻値に返します。非同期型の場合は、結果が返ってくるのを待たずにすぐ終了し、結果が返ってきた時点でその結果を文字列でコールバック関数に渡します。

読み込みコマンドとして null を指定した場合は、そのチャンネルに登録された状態監視用の読み込みコマンドが用いられます。

3.3.3 io_wait() : I/O 状態変化待ち (非同期)

書式: io_wait(チャンネル, 比較対象となる状態,
コールバック関数, タイムアウト時間)

戻値: 成功した場合は null 以外の値を返します。

失敗した場合は null を返します。

説明: そのチャンネルの初期化時に記憶された状態監視用の読み込みコマンドを用いて状態を監視するように、サーバ側に状態監視要求を出します。

タイムアウト時間はミリ秒単位で指定します。null または -1 を指定すると永久待ちとなります。

状態が比較対象となる状態から変化したことを検出した場合、その時点で非同期にコールバック関数に通知されます。コールバック関数には変化した状態が 16 進文字列で渡されます。

一方、タイムアウトした場合も、コールバック関数に通知されます。この場合にはコールバック関数には null が渡されます。

いずれの場合も、コールバック関数の中から再び状態監視要求を出したい場合は、コールバック関数の中から次の `io_wait()` を発行することも可能です。

なお、コールバック関数が呼ばれる前の時点で別の `io_wait()` を発行することはできません。

I/O 状態変化待ちの応答速度について

サーバ内部での I/O 監視周期はデフォルトでは 100 ミリ秒です (起動時引数 `-io-period` で変更できます)。実際の応答速度は、これに LAN の通信時間を加えたものとなります。

3.3.4 `io_write()` : I/O 書き込み

書式: `io_write( チャンネル, 書き込みコマンド )`

戻値: 成功した場合は, `null` 以外の値を返します。

失敗した場合は `null` を返します。

説明: 書き込みコマンドを発行します。

3.3.5 `rs_init()` : RS–232 初期化

書式: `rs_init( ポート番号, 初期化コマンド )`

戻値: 成功した場合は `null` 以外の値を返します。

失敗した場合は `null` を返します。

説明: RS–232 の指定されたポートに対して、初期化コマンドを発行します。

3.3.6 `rs_read()` : RS–232 読み込み

書式:

同期型: `rs_read( ポート番号, 文字数 )`

非同期型: `rs_read( ポート番号, 文字数, コールバック関数 )`

戻値:

同期型: 成功した場合は、読み込んだ文字列を返します。
失敗した場合は `null` を返します。

非同期型: 成功した場合は、`null` 以外の値を返します。
失敗した場合は `null` を返します。

説明: RS–232 の指定されたポートから、指定された文字数の文字列を読み込みます。

同期型の場合は、結果が返ってくるまで待って、その結果を文字列で戻値に返します。非同期型の場合は、結果が返ってくるのを待たずにすぐ終了し、結果が返ってきた時点でその結果を文字列でコールバック関数に渡します。

指定された文字数を読み込み終わる前にタイムアウトした場合は、その時点までに読み込めた文字列が結果となります。特に 1 文字も読み込めずにタイムアウトした場合は、結果は空文字列となります。

3.3.7 `rs_write()` : RS–232 書き込み

書式:

同期型: `rs_write( ポート番号, 書き込み文字列 )`

非同期型: `rs_write( ポート番号, 書き込み文字列, コールバック関数 )`

戻値:

同期型: 成功した場合は、書き込んだ文字数を返します。
失敗した場合は `null` を返します。

非同期型: 成功した場合は、`null` 以外の値を返します。
失敗した場合は `null` を返します。

説明: RS–232 の指定されたポートに、指定された文字列を書き込みます。

同期型の場合は、結果が返ってくるまで待って、その結果を文字数で戻値に返します。非同期型の場合は、結果が返ってくるのを待たずにすぐ終了し、結果が返ってきた時点でその結果を文字数でコールバック関数に渡します。

指定された文字列がすべて書き込み終わる前にタイムアウトした場合は、その時点までに書き込めた文字数を返します。特に 1 文字も書き込めずにタイムアウトした場合は、結果は 0 となります。

3.4 HTTP リクエスト一覧

サーバの受理する HTTP リクエストの一覧を示します。

機器制御ライブラリ `io.js` やファイルアップロード用の `FILE/file.html` は、これらの HTTP リクエストを発行するためのインタフェースです。これらを使えば、通常は直接 HTTP リクエストを発行する必要はありませんが、これらを置き換える場合など、必要に応じてご利用ください。

なお、以下の説明では、リクエストヘッダのうち、Content-Length: (リクエスト本体の長さ) や Authorization: (認証情報の送信) などについては、常に必要となるため表記を省略しています。

3.4.1 GET：コンテンツ読み込み

書式: GET パス名 HTTP/1.1 ↩

戻値: 読み込んだファイルの内容を返します。

失敗した場合はエラーを返します。

説明: パス名で指定されるファイルの内容を返します。

パス名の文字の範囲としては、ASCII 文字のみ使用可能です。

3.4.2 HEAD：コンテンツのヘッダ読み込み

書式: HEAD パス名 HTTP/1.1 ↩

戻値: ヘッダを返します。

失敗した場合はエラーを返します。

説明: パス名で指定されるファイルの内容を返すためのヘッダのみを返します。

3.4.3 POST /IO/INIT：I/O 初期化

書式: POST /IO/INIT HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
チャンネル番号 ↩
初期化コマンド ↩
状態監視用の読み込みコマンド ↩

戻値: 成功した場合は、状態監視用の読み込みコマンドを用いて現在状態を求め、その結果を 16 進文字列で返します。

失敗した場合はエラーを返します。

説明: 指定したチャンネルに対して、初期化コマンドを発行するとともに、状態監視用の読み込みコマンドを登録します。

チャンネル番号は 1 以上 4 以下の番号で指定します (起動時引数 -max-channel で最大チャンネル番号は変更できます)。独立した二つ以上の処理を同

じサーバで行わせる場合は、異なるチャンネルを選ぶことにより、お互いに干渉しないようにすることができます。

初期化コマンドは省略することができます。その場合は空文字列を指定して下さい。省略した場合、初期化は行われません。

状態監視用の読み込みコマンドは省略することができます。省略する場合は空文字列を指定して下さい。省略した場合、それ以前に同じチャンネルに登録されていた状態監視用の読み込みコマンドが引き続き有効になります。(なお一度も登録していない場合は、状態監視用の読み込みコマンドは未定義状態となりますので、状態監視はできません)

既に同じチャンネルに対して以前に初期化コマンドが発行されていた場合は、以前に発行した初期化コマンドと今回の初期化コマンドを比較して、もしも同じ場合には、今回の初期化コマンドは無視され、初期化は行われません。これはブラウザの再読み込みなどのときに、同じ初期化が再度行われることを防ぐためです。一方、初期化コマンドが異なる場合は、今回指定した初期化コマンドによって再度初期化が行われます。

既に同じチャンネルに対して以前に状態監視用の読み込みコマンドが登録されていた場合は、後から登録した方が有効になり、前に登録していた方は無効になります。

3.4.4 POST /IO/READ : I/O 読み込み (ポーリング)

書式: POST /IO/READ HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
チャンネル番号 ↩
読み込みコマンド ↩

戻値: 成功した場合は結果を 16 進文字列で返します。

失敗した場合はエラーを返します。

説明: 読み込みコマンドを発行し、結果を受け取ります。結果は 16 進文字列で返されます。

読み込みコマンドとして空文字列を指定した場合は、そのチャンネルに登録された状態監視用の読み込みコマンドが用いられます。

3.4.5 POST /IO/WAIT : I/O 状態変化待ち (非同期)

書式: POST /IO/WAIT HTTP/1.1 ↩
Content-Type: text/plain ↩

↩
チャンネル番号 ↩
タイムアウト時間（ミリ秒単位）↩
比較対象となる状態（16 進文字列）↩

戻値: 成功した場合は変化後の状態が非同期に 16 進文字列で戻されます。

タイムアウトした場合はヌル文字 1 文字からなる文字列が非同期に戻されます。

失敗した場合はエラーを返します。

説明: そのチャンネルの初期化時に記憶された状態監視用の読み込みコマンドを用いて状態を監視するように、サーバ側に状態監視要求を出します。

タイムアウト時間はミリ秒単位で指定します。-1 を指定すると永久待ちとなります。

状態が比較対象となる状態から変化したことを検出した場合、その時点で非同期に結果が返されます。結果は変化した状態を 16 進文字列で返します。

タイムアウトした場合はヌル文字 1 文字からなる文字列を返します。

なお、タイムアウトになる前にクライアント側が接続を切断した場合は、その時点でサーバ側は監視処理を中止します。

I/O 状態変化待ちの応答速度について

サーバ内部での I/O 監視周期はデフォルトでは 100 ミリ秒です (起動時引数 -io-period で変更できます)。実際の応答速度は、これに LAN の通信時間を加えたものとなります。

3.4.6 POST /IO/WRITE：I/O 書き込み

書式: POST /IO/WRITE HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
チャンネル番号 ↩
書き込みコマンド ↩

戻値: 成功した場合は「200 OK」を返します。

失敗した場合はエラーを返します。

説明: 書き込みコマンドを発行します。

3.4.7 POST /RS/INIT：RS–232 初期化

書式: POST /RS/INIT HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
ポート番号 ↩
初期化コマンド ↩

戻値: 成功した場合は「200 OK」を返します。

失敗した場合はエラーを返します。

説明: RS–232 の指定されたポートに対して、初期化コマンドを発行します。

3.4.8 POST /RS/READ：RS–232 読み込み

書式: POST /RS/READ HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
ポート番号 ↩
読み込み文字数 ↩

戻値: 成功した場合は、読み込んだ文字列を非同期に返します。

失敗した場合はエラーを返します。

説明: RS–232 の指定されたポートから、指定された文字数の文字列を読み込みます。

読み込みが成功した時点で、読み込めた文字列を非同期に返します。

指定された文字数を読み込み終わる前にタイムアウトした場合は、その時点までに読み込めた文字列を返します。特に 1 文字も読み込めずにタイムアウトした場合はヌル文字 1 文字からなる文字列を返します。

3.4.9 POST /RS/WRITE：RS–232 書き込み

書式: POST /RS/WRITE HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
ポート番号 ↩
書き込み文字列 ↩

戻値: 成功した場合は、書き込んだ文字数を非同期に返します。

失敗した場合はエラーを返します。

説明: RS–232 の指定されたポートに、指定された文字列を書き込みます。

書き込みが成功した時点で、書き込めた文字数を非同期に返します。

指定された文字列がすべて書き込み終わる前にタイムアウトした場合は、その時点までに書き込めた文字数を返します。特に 1 文字も書き込めずにタイムアウトした場合は、結果は 0 となります。

3.4.10 POST /FILE/LS：ディレクトリ一覧の取得

書式:
```
POST /FILE/LS HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
ディレクトリのパス名
```

戻値: パス名で指定されたディレクトリのリストを返します。

説明: パス名で指定されたディレクトリのリストを、テキストで返します。テキストの各行の形式は次の通りです:

ディレクトリの場合: ディレクトリ名 / ↩
ファイルの場合: ファイル名 ↩

なお「.」および「..」は含みません。

3.4.11 POST /FILE/UPLOAD：ファイルのアップロード

書式:
```
POST /FILE/UPLOAD HTTP/1.1 ↩
Content-Type: multipart/form-data; boundary=境界文字列 ↩
↩
--境界文字列 ↩
ファイルのパス名 ↩
--境界文字列 ↩
アップロード成功時に返す HTML ↩
--境界文字列 ↩
アップロード失敗時に返す HTML ↩
--境界文字列 ↩
ファイルの内容 ↩
--境界文字列--
```

戻値: 成功すれば、アップロード成功時に返す HTML を返します。

失敗すれば、アップロード失敗時に返す HTML を返します。

説明: ファイルをアップロードします。

3.4.12 POST /FILE/UNLINK：ファイル削除

書式: POST /FILE/UNLINK HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
削除したいファイルのパス名

戻値: 成功した場合は「200 OK」を返します。
失敗した場合はエラーを返します。

説明: パス名で指定されたファイルを削除します。

3.4.13 POST /FILE/MKDIR：ディレクトリ作成

書式: POST /FILE/MKDIR HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
作成したいディレクトリのパス名

戻値: 成功した場合は「200 OK」を返します。
失敗した場合はエラーを返します。

説明: パス名で指定されたディレクトリを作成します。

3.4.14 POST /FILE/RMDIR：ディレクトリ削除

書式: POST /FILE/RMDIR HTTP/1.1 ↩
Content-Type: text/plain ↩
↩
削除したいディレクトリのパス名

戻値: 成功した場合は「200 OK」を返します。
失敗した場合はエラーを返します。

説明: パス名で指定されたディレクトリを削除します。
ディレクトリが空でない場合は、再帰的に子孫のファイルやディレクトリの削除を行います。

3.4.15 POST /FILE/SYNC：ファイルシステム保存

書式: POST /FILE/SYNC HTTP/1.1 ↩
Content-Type: text/plain ↩
↩

戻値: 成功した場合は「200 OK」を返します。
失敗した場合はエラーを返します。

説明: ファイルシステムを保存します。

4 チュートリアル

機器制御ライブラリを使って実際に機器制御を行うためのご参考に、ここでは簡単なチュートリアルをご紹介します。

4.1 LED 書き込み

書き込みだけで読み込みの不要な簡単な例として、ラジオボタンで LED をつけたり消したりしてみましょう。

sample1.html

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html>
3  <head>
4  <!-- サンプル 1: LED 書き込み
5       Copyright (C) 2007 by Personal Media Corporation
6  -->
7  <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
8  <meta http-equiv="Content-Script-Type" content="text/javascript">
9  <title>LED 書き込み</title>
10 <script type="text/javascript" src="/io.js"></script>
11 </head>
12
13 <body onLoad="io_init(1, 'LED1=1', null, null);">
14
15 <h1>LED 書き込み</h1>
16
17 <form>
18 LED1
19 <input type="radio" name="LED1" id="LED1_ON"
20      onClick="io_write(1, 'LED1=1');" checked>
21      <label for="LED1_ON">点灯</label>
22 <input type="radio" name="LED1" id="LED1_OFF"
23      onClick="io_write(1, 'LED1=0');">
24      <label for="LED1_OFF">消灯</label>
25 </form>
26
27 </body>
28 </html>
```

- `<script type="text/javascript" src="/io.js"></script>`

  機器制御ライブラリ `io.js` を取り込みます。
  この例ではルート直下に `io.js` を置くものとしています。

- `<body onLoad="io_init(1, 'LED1=1', null, null);">`

  ページをロードした時に、`io_init()` を使って I/O を初期化します。

  チャンネル番号はここではたまたま 1 としていますが、他の用途と重ならなければ、別のチャンネルを選んでも構いません。

  初期化コマンドは「`LED1=1`」です。これによって LED1 の初期状態は 1 (点灯状態) になります。

読み込みコマンドとコールバック関数は今回は不要ですので、両方とも null としています。

- `<input type="radio" name="LED1" id="LED1_ON" onClick="io_write(1, 'LED1=1');" checked>`

  LED1 を点灯するためのラジオボタンです。

  クリックされたら io_write() を使って書き込みコマンド「LED1=1」を発行します。その結果 LED1 が点灯するはずです。

  初期状態は点灯状態としたので、checked を指定してデフォルトでチェックを入れています。

- `<input type="radio" name="LED1" id="LED1_OFF" onClick="io_write(1, 'LED1=0');">`

  LED1 を消灯するためのラジオボタンです。

  クリックされたら io_write() を使って書き込みコマンド「LED1=0」を発行します。その結果 LED1 が消灯するはずです。

4.2 スイッチ状態の読み込み

次に、状態変化待ちの簡単な例として、スイッチ SW1 と SW2 の現在状態をリアルタイムにチェックボックスに表示してみましょう。

sample2.html

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html>
3  <head>
4  <!-- サンプル 2: スイッチ状態の読み込み
5       Copyright (C) 2007 by Personal Media Corporation
6  -->
7  <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
8  <meta http-equiv="Content-Script-Type" content="text/javascript">
9  <title>スイッチ状態の読み込み</title>
10 <script type="text/javascript" src="/io.js"></script>
11 <script type="text/javascript">
12 <!--
13 function f(x) {
14      document.getElementById("SW1").checked =
15                    parseInt(x.substring(0,1),16);
16      document.getElementById("SW2").checked =
17                    parseInt(x.substring(1,2),16);
18      io_wait(2, x, f, null);
19 }
20 // -->
21 </script>
22 </head>
23
24 <body onLoad="io_init(2, 'SW1=IN;SW2=IN', 'SW1:4;SW2:4', f);">
25
```

```
26  <h1>スイッチ状態の読み込み</h1>
27
28  <form>
29  <input type="checkbox" id="SW1" disabled><label for="SW1">SW1</label>
30  <input type="checkbox" id="SW2" disabled><label for="SW2">SW2</label>
31  </form>
32
33  </body>
34  </html>
```

- `<script type="text/javascript" src="/io.js"></script>`

  機器制御ライブラリ `io.js` を取り込みます。
  この例ではルート直下に `io.js` を置くものとしています。

- `<body onLoad="io_init(2, 'SW1=IN;SW2=IN', 'SW1:4;SW2:4', f);">`

  ページをロードした時に、`io_init()` を使って I/O を初期化します。

  チャンネルは、ここでは「4.1 LED 書き込み」の例と今回の例を並行して同時に実行することも考慮に入れて、重ならないように 2 としています。

  初期化コマンドは「`SW1=IN;SW2=IN`」です。SW1 と SW2 を入力用に指定しています。

  状態監視用の読み込みコマンドは「`SW1:4;SW2:4`」です。スイッチの状態は 1 ビットの値ですが、ここでは 4 ビットと大きめに指定しています。これにより 16 進数 1 桁に対応します。

  コールバック関数は `f` を指定しています。初期化が成功すれば、`f` には初期状態が渡されます。

- ```
  function f(x) {
      document.getElementById("SW1").checked =
                          parseInt(x.substring(0,1),16);
      document.getElementById("SW2").checked =
                          parseInt(x.substring(1,2),16);
      io_wait(2, x, f, null);
  }
  ```

  コールバック関数 `f` は、引数 `x` として 2 桁の 16 進文字列を受け取ります。

  1 桁目の値によって、チェックボックス (id = `SW1`) に印をつけます。

  2 桁目の値によって、チェックボックス (id = `SW2`) に印をつけます。

  最後に `io_wait()` を使って、現在状態 `x` と比較して状態が変化したら同じコールバック関数 `f` が呼ばれるように設定します。その結果とし
  ```

て、スイッチの状態が変化するたびに毎回コールバック関数 f が呼ばれ、チェックボックスにスイッチの状態を表示します。

io_wait() の第 4 引数にはタイムアウトを指定できますが、ここでは null、つまり永久待ちを指定しています。すなわち、状態が変化しないかぎりはタイムアウトせず、コールバック関数 f は呼ばれません。

- <input type="checkbox" id="SW1" disabled>

  スイッチ SW1 の状態を表示するためのチェックボックスです。

  チェックボックスを区別するため、id として「SW1」を付けています。

  表示専用なのでユーザが変更できないように disabled を指定しています。

- <input type="checkbox" id="SW2" disabled>

  スイッチ SW2 の状態を表示するためのチェックボックスです。

  チェックボックスを区別するため、id として「SW2」を付けています。

  表示専用なのでユーザが変更できないように disabled を指定しています。

4.3 複数クライアントへの対応

複数のクライアント (別々のマシン上にあるウェブブラウザ) から、同じ HTML を表示させた場合はどうなるでしょうか。

† ここで「複数のクライアント」とは、別々のマシン上にあるウェブブラウザを指します。同一マシン上での複数のブラウザからの接続については制限があります。詳しくは「3.3 機器制御ライブラリ io.js」の冒頭をご参照下さい。

「4.2スイッチ状態の読み込み」の場合

こちらの場合は、各クライアントがそれぞれ状態変化の監視要求を io_wait() で出していますから、スイッチの状態が変化すると、その状態変化はいっせいに各クライアントに通知されます。従って複数クライアントに最初から対応しています。

「4.1 LED 書き込み」の場合

あるクライアント からラジオボタンを押して io_write() を発行すると、サーバ側はそれに応じて LED に書き込みを行って、その結果として LED の状態が変化します。しかしその状態変化を別のクライアントに通知すること

はしていませんでしたので、他のクライアント上のラジオボタンは変化しないままです。

しかし、LED の状態変化を通知するように修正すれば、簡単に複数クライアントにも対応させることができます。そのポイントは次の通りです。

- io_init() の第3引数で、LED 状態監視用の読み込みコマンド「LED1:4」を指定します。
- io_init() の第 4 引数で、コールバック関数 f を指定します。
- コールバック関数 f では、渡された現在の LED 状態をラジオボタンに反映するようにします。
- 次の状態変化に備えるために、コールバック関数 f の中から io_wait() を用いて、状態変化を自分自身 f に通知するように設定します。
- ラジオボタンの初期状態はコールバック関数 f で設定しますので、デフォルトの checked は不要です。

実際の HTML は次のようになります。

sample3.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<!-- サンプル 3: LED 書き込み (複数クライアント対応版)
     Copyright (C) 2007 by Personal Media Corporation
-->
<meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<title>LED 書き込み (複数クライアント対応版)</title>
<script type="text/javascript" src="/io.js"></script>
<script type="text/javascript">
<!--
function f(x) {
     document.getElementById("LED1_ON").checked =
                    parseInt(x.substring(0,1),16);
     document.getElementById("LED1_OFF").checked =
                    !parseInt(x.substring(0,1),16);
     io_wait(1, x, f, null);
}
// -->
</script>
</head>

<body onLoad="io_init(1, 'LED1=1', 'LED1:4', f);">

<h1>LED 書き込み (複数クライアント対応版)</h1>

<form>
LED1
<input type="radio" name="LED1" id="LED1_ON"
     onClick="io_write(1, 'LED1=1');">
     <label for="LED1_ON">点灯</label>
```

```
33  <input type="radio" name="LED1" id="LED1_OFF"
34       onClick="io_write(1, 'LED1=0');">
35       <label for="LED1_OFF">消灯</label>
36  </form>
37
38  </body>
39  </html>
```

4.4 画面の見栄えを追求するには

このチュートリアルでは、理解のしやすさの観点から、単純に HTML のラジオボタンやチェックボックスをそのまま使ってきました。そのため見栄えという観点からすると若干貧相なことは否めませんでした。しかしもちろん、ウェブページ制作のためのいろいろな技術を組み合わせることで、もっと見栄えのよい画面を作成することも可能です。

ここでは簡単な一例として、Internet Explorer でサポートされている VML (Vector Markup Language) を使って LED を描画してみましょう。

- VML の `<v:oval>` で円を描きます。
- 光沢を表現するために、`<v:fill type="gradientRadial">` で円の内部をグラデーションで塗り、中心付近を白くします。
- `focusposition="0.3,0.3"` でグラデーションを円の中心から左上にずらし、左上方向から光が当たっているように見せます。

- 点灯時の円と消灯時の円を `<v:group>` でまとめて同じ座標に配置します。
- 点灯時の円がクリックされた場合は、`io_write()` で LED を消灯させます。一方、消灯時の円がクリックされた場合は、`io_write()` で LED を点灯させます。
- 状態変化がコールバック関数に通知されると、円の `style.visibility` を操作することで、点灯時の円と消灯時の円の表示を切り替えます。

実際の HTML は次のようになります。

sample4.html

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html xmlns:v="urn:schemas-microsoft-com:vml">
3  <head>
4  <!-- サンプル 4: LED 書き込み (VML 版: Internet Explorer のみ実行可)
5       Copyright (C) 2007 by Personal Media Corporation
6  -->
7  <style>v\:* { behavior: url(#default#VML); }</style>
```

```
 8  <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
 9  <meta http-equiv="Content-Script-Type" content="text/javascript">
10  <title>LED 書き込み (VML 版)</title>
11  <script type="text/javascript" src="/io.js"></script>
12  <script type="text/javascript">
13  <!--
14  function f(x) {
15       document.getElementById("LED1_ON").style.visibility =
16                 parseInt(x.substring(0,1),16) ? "visible": "hidden";
17       document.getElementById("LED1_OFF").style.visibility =
18                 !parseInt(x.substring(0,1),16) ? "visible": "hidden";
19
20       io_wait(1, x, f, null);
21  }
22  // -->
23  </script>
24  </head>
25
26  <body onLoad="io_init(1, 'LED1=1', 'LED1:4', f);">
27
28  <h1>LED 書き込み (VML 版)</h1>
29
30  <form>
31  <v:group style="width:40;height:40;cursor:hand;" coordsize="40,40">
32       <v:oval style="width:40;height:40;" id="LED1_ON" strokecolor="white"
33            onClick="io_write(1, 'LED1=0');">
34            <v:fill type="gradientRadial" color="red" color2="white"
35                 focusposition="0.3,0.3" />
36       </v:oval>
37       <v:oval style="width:40;height:40;" id="LED1_OFF" strokecolor="white"
38            onClick="io_write(1, 'LED1=1');">
39            <v:fill type="gradientRadial" color="gray" color2="white"
40                 focusposition="0.3,0.3" />
41       </v:oval>
42  </v:group>
43  </form>
44
45  </body>
46  </html>
```

† VML は現在のところ Internet Explorer 以外のブラウザではサポートされていません。したがってこの `sample4.html` は Internet Explorer 専用です。

しかし Firefox など他のブラウザのいくつかでは `<canvas>` タグがサポートされています。`<canvas>` タグを用いることにより、VML と同様に見栄えのよい画面を作成可能です。

PMC 機器制御サーバ (μTeaboard/ARM7-AT91 版)
取扱説明書
Version 1.00

パーソナルメディア株式会社
〒 141–0022 東京都品川区東五反田 1–2–33 白雉子ビル
Tel: 03–5475–2185　Fax: 03–5475–2186
Web: http://www.t-engine4u.com/
E-Mail: te-sales@personal-media.co.jp