# Funnel v1.0：仕様書（草稿）

開発者：小林茂
共同開発者：遠藤孝則＋増田一太郎

2007 年 11 月 22 日

## 1 はじめに

Funnel は、ソフトウェアとハードウェアからなるフィジカルコンピューティングのためのツールキットです。Funnel は 2007 年度第 I 期未踏ソフトウェア創造事業の支援を得て開発されています。

### 1.1 概要

- センサやアクチュエータを、GUI パーツと同様の感覚で扱えるようにするための言語拡張をライブラリ形式で提供します。対象とする言語は ActionScript 3、Processing と Ruby です。
- 機能がシンプルであるため、短いドキュメントを読めばすぐに理解して利用できるようになっています。また、ライブラリが提供する機能で対応できない場合でも、ユーザによって拡張できるような柔軟性を持たせています。
- Funnel 用に開発した I/O モジュール（FIO）以外に、既に広く普及している I/O モジュール（Gainer と Arduino）を利用できます。

### 1.2 ターゲット

Funnel のターゲットは、Flash CS3/Flex 2 や Processing をある程度扱ったことがあり、PC 標準の入出力デバイス以外の世界を扱ってみたいと考えているデザイナー、アーティスト、エンジニアおよび関連した教育分野（フィジカルコンピューティングなど）です。

### 1.3 マイルストーン

- 2007 年 6 月：Sketching in Hardware 2[*1]にてプレゼンテーションとディスカッション
- 2007 年 9 月：最初の公開ビルドをリリース
- 2007 年 11 月：公開ワークショップ（東京）
- 2007 年 12 月：v1.0 リリース

[*1] Mike Kuniavsky 氏が主催するフィジカルコンピューティングの教育と実践に関わるメンバーのための会議。2006 年に第 1 回を開催。http://www.sketching07.com/

# 2　特徴

## 2.1　マルチプラットホームへの対応

Funnel v1.0 は以下の I/O モジュール（一部はマイコンモジュール）への対応を予定しています。新規に開発する Funnel I/O モジュール以外に、既に普及している Gainer[*2]および Arduino シリーズ[*3]、および MaxStream 社の XBee シリーズに対応します。これにより、Funnel は v1.0 のリリース時点で 1 万数千台以上のハードウェア環境で動作できることになります。

- Funnel I/O モジュール（USB 接続のドングル＋無線モジュール）
- Gainer I/O モジュール（USB 接続）
- Arduino（USB 接続）
- MaxStream XBee シリーズ 1/2（USB 接続のドングル＋無線モジュール）

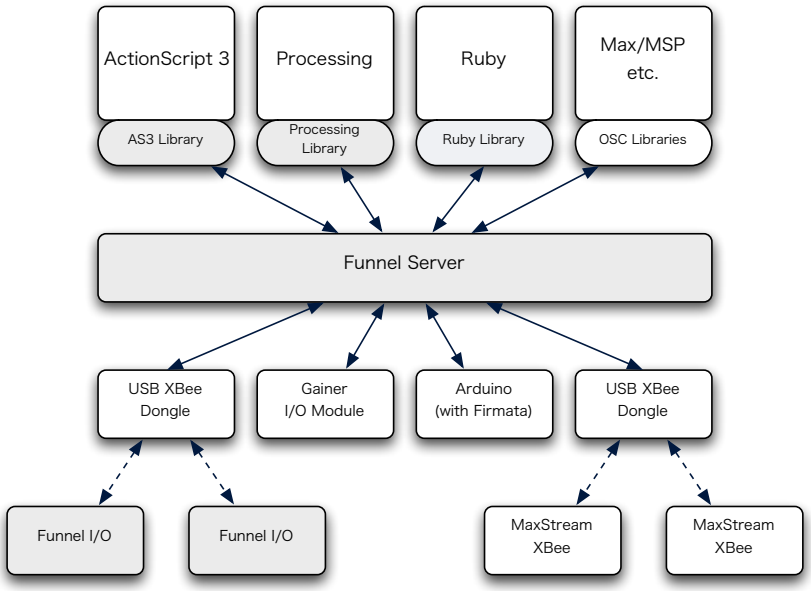


図 1　ツールキット Funnel のシステム全体図（今回開発する部分はグレーの部分）

Funnel Server は仮想シリアルポートドライバ経由でそれぞれの I/O モジュールと通信します。Funnel Server とアプリケーション側との通信は、音楽系のプログラミング環境を中心に広く用いられているプロトコル Open Sound Control[*4]で行います。ActionScript 3 と Processing に対しては言語拡張を行うための Funnel ライブラリ（詳細は後述）で利用できますが、これに加えて Open Sound Control に対応する全てのプログラミング環境[*5]からも利用できます。

[*2] 2007 年 5 月の時点で 500 台以上。http://gainer.cc/
[*3] 2007 年 5 月の時点でシリーズ合計で 10000 台以上。ファームウェアは Firmata を使用。http://arduino.cc/
[*4] http://www.cnmat.berkeley.edu/OpenSoundControl/
[*5] 音楽系では Max/MSP や SuperCollider 3 など、スクリプト言語系では Perl、PHP、Python など多数。

Funnel Server により、物理的な I/O モジュールは抽象化されるため、販売期間終了後の故障などにより特定の I/O モジュールが利用できなくなってしまった場合でも、その時点で対応するものに交換することが可能です。このように抽象化するレイヤーを用意することにより、Funnel は長期間に渡って利用と開発を継続することが可能になります。

## 2.2 疑似コードベースによるロジック重視のプログラミング

### 2.2.1 疑似コードベースのプログラミングとは

初心者に最初にプログラミングの概念を教える際、「疑似コード」と呼ばれるものを使うことが多くあります。ここでの疑似コードとは、次のように自然言語でプログラムのロジックを記述するものです。

リスト 1　疑似コードの例

```
デジタル入力 0 に接続したスイッチが押されたらハンドラを呼ぶ
```

自然言語で記述することから、親しみやすく、可読性に優れているのが特長ですが、そのままでは実行できません。そこで、C や ActionScript といった言語を用いて実装していくことになります。こうした言語は「高級」言語と呼ばれますが、実際にはローレベルの記述しかできず、ロジックを「そのまま」記述することはできません。ロジックをローレベルの記述に変換する作業は、初心者にとってはかなり敷居の高いものになってしまいます。例えば、この場合には次のようなコードを実装することになります。

リスト 2　コード例 1 の実装例（ActionScript 2 + Gainer）

```
var io:Gainer = new Gainer(...);
var wasActivated:Boolean = false;

function loop() {
    var isActivated:Boolean = io.digitalInput[0];
    if (!wasActivated && isActivated) {
        handler();
    }
    wasActivated = isActivated;
}

function handler() {
    ...
}
```

経験者にとっては何でもないこうしたコードですが、初心者にとってはかなり敷居が高いものになります。また、経験者はテンプレート的にこうした実装を行うことができますが、たかだかこれだけのことをするために毎回同じようなコードを書かなければならないというのはかなり無駄です。

そこで、疑似コードに近い形で記述できる新しい言語拡張ライブラリを提案します。例えば、次のように記述することができれば、短い行数で簡潔に記述することができますし、何をしようとしているのかも明確です。

リスト 3　コード例 1 の実装例（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);
gio.digitalInput(0).addEventListener(PortEvent.RISING_EDGE, buttonPressed);

function buttonPressed(e:PortEvent):void {
    ...
}
```

リスト 4　コード例 1 の実装例（Ruby + Funnel）

```
gio = Gainer.new(...)
gio.digital_input(0).on PortEvent::RISING_EDGE do
  ...
end
```

### 2.2.2　疑似コードベースのプログラミングの実現方法

今回の提案は、全く新しい言語の提案ではなく、新しいパラダイムの提案です。もちろん、新しい言語をデザインするというアプローチもありますが、今回の主なターゲットとしているユーザ（デザイナー、アーティスト、プログラミング初心者）には、以下の理由により適さないと考えます。

- 初心者が同時に複数の言語を学習しようとすると混乱が生じる
- 新しい言語に対して十分なドキュメントを提供するのは容易ではない
- 新しい言語が十分に普及すれば獲得したスキルは無駄にならないが、そうでない場合にはスキルが無駄になってしまう

Funnel は、ActionScript と Processing という、世界的に既に広く用いられていて、かつ学習用の教材も十分に整っている言語をベースにして、I/O モジュールに関連した処理を行うための必要最小限の拡張を行います。拡張の内容は以下の通りです。

- 入力の変化を検出する（例：閾値付き変化検出）
- アナログ入力に対する代表的なフィルタ処理（例：ローパス、ハイパス）
- 時間とともに変化する出力（例：ワンショット、点滅）
- 異なる単位間にも対応するスケーリング（例：加速度センサの電圧出力→角度）

拡張は必要最小限であるため、ユーザがベースになる言語を習得していれば、いくつかのサンプルを眺めるだけですぐに使い始めることができます。また、これらのライブラリを使用して記述されたコードは、何を目的としているのかが明確になり、ロジックそのものの記述ミスによるバグの発生を防ぎます。また、目的が明確になることで、教育分野で初心者が書いたコードを添削する場合にも有効です。

Funnel は、これらのライブラリをバイナリで提供するのではなく、ソースコードの状態で提供します。これにより、スキルがアップした段階でローレベルではどのように記述されているのか知りたくなった時や、自分なりに機能拡張したくなった時に手軽に調べることが可能になります。

なお、「センサからの入力を一定回数サンプリングして加算平均することで自動的にベースラインを求める」といったレベルでライブラリを用意してしまうと、確かに特定の処理は簡単に行えるようになります。しかし、それぞれ用途が限定されているために応用範囲が狭く、ライブラリが肥大化してしまう危険性が大いにあります。また、内部処理がブラックボックス化されてしまうため、ユーザのスキルアップにつながりません。この観点から、あくまで必要最小限のレベルにとどめます。

## 2.3 専用ハードウェア「FIO」について

Funnel 用に開発する I/O モジュール（FIO：フィオ）は、以下のパートから構成されます。

- USB ⇔ XBee ドングル（New Micros 製 USB XBee Dongle など）
  - USB-to-UART 変換ブリッジ（FTDI 製 FT232RL）
  - ワイヤレス通信モジュール（MaxStream 製 XBee）
- 無線モジュール
  - PSoC マイコン（Cypress Semiconductor 製 CY8C27143）
  - ワイヤレス通信モジュール（MaxStream 製 XBee）

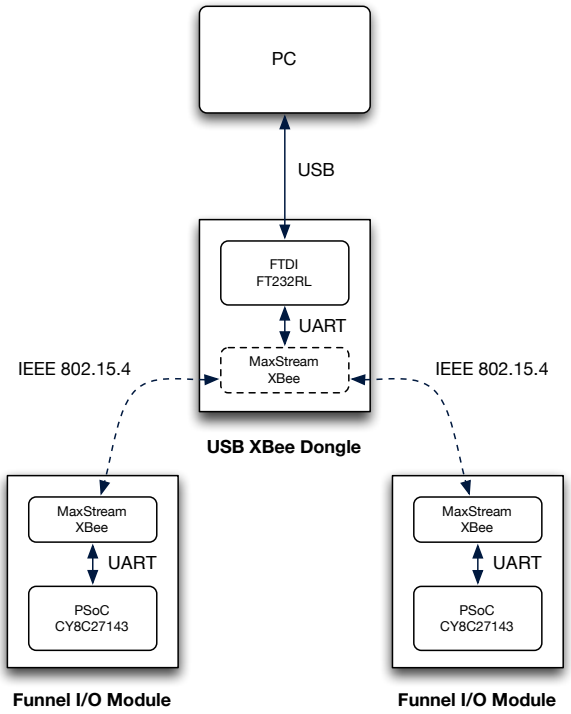


図 2　Funnel I/O モジュールの構成図

USB ⇔ XBee ドングルは USB と IEEE 802.15.4 の間の変換を行います。無線モジュールのハードウェアとファームウェアは基本となるものを開発し、リファレンスデザインとして公開しますが、PSoC Express[*6]や PSoC Designer[*7]により、ユーザ自身がデザインして簡単に追加することも可能です。また、全てオープンソースで公開することにより、基本デザインをベースとして PSoC 以外のマイコン（AVR、MSP430、Propeller[*8]など）を用いたバリエーションを作成することも可能です。

[*6] C やアセンブラでコードを書くことなくマイコンのファームウェアを開発できるツール。
[*7] PSoC マイコンの全ての機能を利用できる開発ツール。利用できる言語は C またはアセンブラ。
[*8] BASIC Stamp シリーズで知られる Parallax 社が開発したマルチコアのマイコン。http://www.parallax.com/propeller/

# 3 API リファレンス

## 3.1 データの表現について

入出力ポートのデータは、0 から 1 までの実数（AS3 では Number、Processing では float）で表現します。これにより、I/O モジュールごとに A/D や D/A（または PWM）の分解能が異なる場合でも同様に扱えます。

## 3.2 IOSystem クラス

IOSystem クラスは複数の I/O モジュールから構成されるシステムを表現するためのクラスです。Gainer I/O モジュールや Arduino I/O ボードを利用する際には、IOSystem オブジェクトをラップするユーティリティークラス（Gainer または Arduino）を利用してアクセスすることにより、内部構造を意識することなくアクセスできます。

### 3.2.1 コンストラクタ

```
IOSystem(hostName:String, portNumber:int, modules:int, samplingInterval:int)
```

### 3.2.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `samplingInterval` | int | 現在設定されているサンプリング間隔 |
| `samplingInterval =` | int | サンプリング間隔を設定 |

### 3.2.3 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `module(id:int)` | IOModule | 指定した ID の I/O モジュールを返す |

## 3.3 IOModule クラス

IOModule クラスは Funnel の基本となるクラスです。IOModule オブジェクトは PC に接続された I/O モジュールを抽象化して、共通の方法でアクセスできるようにします。

### 3.3.1 コンストラクタ

```
IOModule(hostName:String, portNumber:int, config:Configuration, samplingInterval:int)
```

### 3.3.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `autoUpdate` | boolean | 出力ポートの値を自動で更新するか否か |
| `portCount` | int | ポート数 |

### 3.3.3 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `port(number:int)` | Port | number で指定したポートを返す |
| `update()` | void | 全ての出力ポートの状態を手動で更新する |

#### 3.3.4 イベントハンドラ（Processing のみで実装）

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `risingEdge(PortEvent event)` | void | `PortEvent.RISING_EDGE` のハンドラ |
| `fallingEdge(PortEvent event)` | void | `PortEvent.FALLING_EDGE` のハンドラ |
| `change(PortEvent event)` | void | `PortEvent.CHANGE` のハンドラ |

### 3.4 Gainer クラス

Gainer クラスは Gainer I/O モジュールを扱うためのユーティリティークラスです。

#### 3.4.1 コンストラクタ

```
Gainer(hostName:String, portNumber:int, mode:int)
```

#### 3.4.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `button` | Port | I/O モジュール上のボタンを示すポート |
| `led` | Port | I/O モジュール上の LED を示すポート |

#### 3.4.3 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `analogInput(number:int)` | Port | number で指定したポートを返す |
| `digitalInput(number:int)` | Port | number で指定したポートを返す |
| `analogOutput(number:int)` | Port | number で指定したポートを返す |
| `digitalOutput(number:int)` | Port | number で指定したポートを返す |

#### 3.4.4 定数

| 名前 | 説明 |
|---|---|
| `AIN` | アナログ入力 |
| `DIN` | デジタル入力 |
| `AOUT` | アナログ（または PWM）出力 |
| `DOUT` | デジタル出力 |

### 3.5 Arduino クラス

Arduino クラスはファームウェアとして Firmata を搭載した Arduino を I/O モジュールとして扱うためのユーティリティークラスです。

#### 3.5.1 コンストラクタ

```
Arduino(hostName:String, portNumber:int)
```

### 3.5.2 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `setDigitalPinMode(pin:int, mode:int)` | void | デジタルピンのモードを設定 |
| `analogPin(number:int)` | Port | number で指定したポートを返す |
| `digitalPin(number:int)` | Port | number で指定したポートを返す |

### 3.5.3 定数

| 名前 | 説明 |
|---|---|
| `IN` | デジタル入力 |
| `OUT` | デジタル出力 |
| `PWM` | 疑似アナログ（PWM）出力 |

## 3.6 Configuration クラス

Configuration クラスは I/O モジュールのコンフィギュレーションを設定するためのクラスです。通常は Gainer クラスや Arduino クラスなどのユーティリティークラスを経由してアクセスされ、ユーザが意識することはありません。

### 3.6.1 コンストラクタ

`Configuration(model:int, mode:int)`

`Configuration(model:int)`

### 3.6.2 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `setDigitalPinMode(pin:int, mode:int)` | void | デジタルピンのモードを設定（Adruino） |

### 3.6.3 定数

| 名前 | 説明 |
|---|---|
| `GAINER` | Gainer I/O モジュール |
| `ARDUINO` | Arduino I/O ボード |
| `XBEE` | XBee モジュール |
| `FUNNEL` | Funnel I/O モジュール |
| `IN` | デジタル入力 |
| `OUT` | デジタル出力 |
| `PWM` | 疑似アナログ（PWM）出力 |

## 3.7 Port クラス

Port クラスは、I/O モジュールの入出力ポートを表現するためのクラスです。ユーザからは IOModule オブジェクトの port() メソッド経由でアクセスします。

### 3.7.1 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| number | int | ポートの番号 |
| type | int | ポートのタイプ（AIN、DIN、AOUT、DOUT のいずれか） |
| value | float | ポート n の現在の値 |
| value = | float | ポート n の値を設定（ポートの type が AOUT または DOUT の場合） |
| lastValue | float | ポート n の変化する前の値 |
| average | float | 平均値 |
| minimum | float | 最小値 |
| maximum | float | 最大値 |
| filters | Array | 現在設定されているフィルタ |
| filters = | Array | フィルタを設定 |

### 3.7.2 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| clear() | void | ヒストリをリセット |
| addEventListener(e:Event, f:function) | void | イベントリスナを設定 |

## 3.8 Event クラス

Event クラスは全てのイベントの基本となる抽象クラスです。

### 3.8.1 Event のインスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| text | String | エラーメッセージなど |

## 3.9 PortEvent クラス

PortEvent クラスは、それぞれの入出力ポートで発生するイベントを表現するためのクラスです。

### 3.9.1 コンストラクタ

```
PortEvent(type:int, text:String, port:Port)
```

### 3.9.2 PortEvent のインスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| target | Port | イベントが発生した Port への参照 |

### 3.9.3 定数

| イベント名 | 設定先 | 説明 |
|---|---|---|
| PortEvent.RISING_EDGE | 各ポート | SetPoint 使用時に入力が変化（0 → 0 以外） |
| PortEvent.FALLING_EDGE | 各ポート | SetPoint 使用時に入力が変化（0 以外→ 0） |
| PortEvent.CANGE | 各ポート | SetPoint 使用時に入力が変化 |

## 3.10 ErrorEvent クラス

ErrorEvent クラスは、動作中に発生する様々なエラーを表現するためのクラスです。

### 3.10.1 コンストラクタ

```
ErrorEvent(type:int, text:String)
```

### 3.10.2 エラー

| 名前 | 説明 |
|---|---|
| ErrorEvent.SERVER_NOT_FOUND_ERROR | Funnel Server が見つからなかった |
| ErrorEvent.REBOOT_ERROR | I/O モジュールの再起動に失敗した |
| ErrorEvent.CONFIGURATION_ERROR | コンフィギュレーションに失敗した |

## 3.11 Filter について

Funnel では、任意のポート（入力または出力）にフィルタをセットすることができます。フィルタは次の関数を実装しているクラスのインスタンスです。

リスト 5　フィルタが実装するインタフェース

```
interface Filter {
    public function processSample(in:Number, buffer:Array):Number;
}
```

具体的には、以下のクラスが Funnel v1.0 でのフィルタとなるクラスです。

- Convolution
- Scaler
- SetPoint
- Osc

次のようにフィルタをセットすると、filter1、filter2、filter3 の順で実行されます。これにより、入力をスムージングした後にスケーリングし、その変化を検出する、というような処理が可能になります。

```
funnel.port(0).filters = [filter1, filter2, filter3];
```

それぞれのフィルタは Funnel の入力ポート（Osc のみ出力ポート）に対して利用することを想定したものですが、processSample() メソッドを手動で呼んで更新することにより、画面表示などの他の部分でも利用することができます。

## 3.12 Convolution クラス

Convolution クラスは入力に対していわゆるデジタル信号処理的なフィルタ処理を行うためのクラスです。細かいノイズを取り除くためのローパスフィルタや、ドリフトを取り除くためのハイパスフィルタがあります。あらかじめ良く使われる種類の処理に対する係数は用意されていますが、ユーザが独自に定義した係数も利用できます。なお、動作中に係数を変更することもできますが、係数の数を変更した場合にはそれまでの履歴がクリアされます。

### 3.12.1 コンストラクタ

```
Convolution(coef:Array)
```

### 3.12.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `coef` | Array | 現在設定されているフィルタの係数 |
| `coef = []` | Array | フィルタの係数を設定 |

### 3.12.3 定数（型：const Array）

| 名前 | 説明 |
|---|---|
| Convolution.LPF | ローパスフィルタ |
| Convolution.HPF | ハイパスフィルタ |
| Convolution.MOVING_AVERAGE | 移動平均フィルタ |

## 3.13 Scaler クラス

Scaler はある範囲の入力をある範囲にスケーリングするためのクラスです。その際、直線でのスケーリング以外に、よく使われるカーブも用意されています。また、ユーザが独自に定義した関数も利用できます。

リスト 6　ユーザ定義関数の実装例

```
function myFilterFunc(in:float, buffer:Array):float {
    return Math.abs(in);
}
```

指定した入力の範囲外（inMin から inMax）の値が入力された場合、デフォルトではリミッタがかかりません。このため、結果として出力は指定した範囲（outMin から outMax）を超えてしまいます。入力の範囲を制限したい場合には、最後の引数を true に設定します。

### 3.13.1 コンストラクタ

```
Scaler(inMin:float, inMax:float, outMin:float, outMax:float,
 type:function, limiter:boolean)
```

### 3.13.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| type | function | 現在設定されているタイプ |
| type = | function | タイプを設定する |
| inMin | float | 現在設定されている入力範囲の最小値 |
| inMin = | float | 入力範囲の最小値を設定する |
| inMax | float | 現在設定されている入力範囲の最大値 |
| inMax = | float | 入力範囲の最大値を設定する |
| outMin | float | 現在設定されている出力範囲の最小値 |
| outMin = | float | 出力範囲の最小値を設定する |
| outMax | float | 現在設定されている出力範囲の最大値 |
| outMax = | float | 出力範囲の最大値を設定する |
| limiter | boolean | 現在設定されているリミッタの状態 |
| limiter = | void | リミッタの状態を設定する |

### 3.13.3 定数（型：function）

| 名前 | 説明 |
|---|---|
| `Scaler.LINEAR` | 直線（$y = x$） |
| `Scaler.SQUARE` | 平方カーブ（$y = x^2$） |
| `Scaler.SQUARE_ROOT` | 平方根カーブ（$y = \sqrt{x}$） |
| `Scaler.CUBE` | 立方カーブ（$y = x^4$） |
| `Scaler.CUBE_ROOT` | 立方根カーブ（$y = \sqrt[4]{x}$） |

## 3.14 SetPoint クラス

SetPoint オブジェクトは、アナログの値に対して閾値とヒステリシスを持つポイントをセットし、processSample() を実行する度に現在の状態を判定して返します。ポイントが 1 つの場合の返り値は 0 または 1 の 2 種類、ポイントが 2 つの場合は 0 または 1 または 2 の 3 種類、ポイントが n 個の場合は 0 から n までの n 種類になります。

### 3.14.1 コンストラクタ

```
SetPoint(threshold:float, hysteresis:float)
SetPoint([[t0:float, h0:float], [t1:float, h1:float], ...])
```

### 3.14.2 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `point[n]` | Array | 現在設定されている n 番目のポイント（閾値とヒステリシス） |

### 3.14.3 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| `addPoint(threshold:float, hysteresis:float)` | void | 新しいポイントを追加する |
| `removePoint(threshold:float)` | void | 指定したポイントを削除する |

## 3.15 Osc クラス

Osc クラスは主に出力に用い、LED をふわふわ点滅させたりする時などに使います。また、回数を 1 回に設定すると、ワンショットの制御にも使えます。あらかじめ良く使われる種類の波形は用意されていますが、ユーザが独自に定義した関数も利用できます。サービス間隔は Osc オブジェクトのクラス変数 serviceInterval の設定に従います。手動で利用する場合には、update() メソッドを利用します。

### 3.15.1 コンストラクタ

```
Osc(wave:function, freq:float, times:int)
Osc(wave:function, freq:float, amp:float, offset:float, phase:float, times:int)
```

### 3.15.2 クラス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| `serviceInterval` | int | サービス間隔 |

### 3.15.3 インスタンス変数

| 変数名 | 型 | 説明 |
|---|---|---|
| wave | function | 現在設定されている波形 |
| wave = | function | 波形を設定 |
| freq | float | 現在設定されている周波数 |
| freq = | void | 周波数を設定 |
| amplitude | float | 現在設定されている振幅 |
| amplitude = | void | 振幅を設定 |
| offset | float | 現在設定されているオフセット |
| offset = | void | オフセットを設定 |
| phase | float | 現在設定されている位相（degree） |
| phase = | void | 位相を設定（degree） |
| times | int | 現在設定されている回数（times） |
| times = | void | 回数を設定（times） |

### 3.15.4 イベント

| イベントハンドラ名 | 変数 | 説明 |
|---|---|---|
| Event.UPDATE | value | オシレータが更新された |

### 3.15.5 メソッド

| メソッド名 | 戻り値 | 説明 |
|---|---|---|
| start() | void | Osc オブジェクトの動作をスタートする |
| stop() | void | Osc オブジェクトの動作をストップする |
| reset() | void | Osc オブジェクトの動作をリセットする |
| update(interval:int) | void | 指定したインターバルだけ時間を進める |
| update() | void | serviceInterval だけ時間を進める |
| addEventListener(e:Event, f:function) | void | イベントリスナを設定する |

### 3.15.6 定数（型：function）

| 名前 | 説明 |
|---|---|
| Osc.SIN | サイン波 |
| Osc.SQUARE | 矩形波 |
| Osc.SAW | ノコギリ波 |
| Osc.TRIANGLE | 三角波 |
| Osc.IMPULSE | インパルス（1 区間のみ 1 になりその後は 0） |

# 4 コード例

Funnel のライブラリを使用することにより、どの程度コードがシンプルかつ可読性の高いものになるか、いくつか具体的な例を示します。

## 4.1 アナログ入力に対する閾値付き変化検出

リスト 7　疑似コード

```
光センサの値があらかじめ設定した閾値を下回ったらハンドラを呼ぶ
例：レーザー光源と光センサの間を通行者が遮ったことを検出
```

リスト 8　7 の実装例（ActionScript 2 + Gainer）

```
var io:Gainer = new Gainer(...);
var lastStatus:Number = -1; // -1: unknown, 0: low, 1:high
var threshold = 80;
var hysteresis = 20;

loop() {
    if (io.analogInput[0] < (threshold - hysteresis)) {
        status = 0;
    } else if (io.analogInput[0] > (threshold + hysteresis)) {
        status = 1;
    } else {
        status = lastStatus;
    }

    if ((lastStatus == 0) && (status == 1)) {
        handler();
    }

    lastStatus = status;
}

function handler():void {
    ...
}
```

リスト 9　7 の実装例（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);
var threshold:Number = 0.3;
var hysteresis:Number = 0.1;

gio.analogInput(0).filters = [new SetPoint(threshold, hysteresis)];
gio.analogInput(0).addEventListener(FALLING_EDGE, handler);

function handler():void {
    ...
}
```

## 4.2 デジタル出力の状態を時間とともに変化させる

リスト 10　疑似コード

```
デジタル出力 0 の状態を 2Hz で交互に切り替える
例：LED を点滅させる
```

リスト 11　10 の実装例（ActionScript 2 + Gainer）

```
var io:Gainer = new Gainer(...);
var value:Boolean = false;

var blinkTimer:Timer = new Timer(250, 0); // 間隔、回数（0 は無限回）
blinkTimer.addEventListener(TimerEvent.TIMER, blink);
blinkTimer.start();

function blink():void {
    if (value == false) {
        value = true;
    } else {
        value = false;
    }

    io.digitalOutput(0, value);
}
```

リスト 12　10 の実装例 1（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);
var blinkOsc:Osc = new Osc(Osc.SQUARE, 2, 0); // 波形、周波数、回数（0 は無限回）

gio.digitalOutput(0).filters = [blinkOsc];
blinkOsc.start();
```

リスト 13　10 の実装例 2（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);

gio.digitalOutput(0).filters = [new Osc(Osc.SQUARE, 2, 0)]; // 波形、周波数、回数（0 は無限回）
gio.digitalOutput(0).filters[0].start();
```

## 4.3 アナログ出力の状態を時間とともに変化させる

リスト 14　疑似コード

```
アナログ出力の値を三角波で連続的に変化させる
例：LED をふわふわ点滅させる
```

リスト 15　14 の実装例（ActionScript 2 + Gainer）

```
var io:Gainer = new Gainer(...);
var value:Number = 0;
var i:Number = 0;

var blinkTimer:Timer = new Timer(20, 0);  // 20ms, forever
blinkTimer.addEventListener(TimerEvent.TIMER, dimming);
blinkTimer.start();

function dimming():void {
    i += 1;
    if (i < 255) {
        value += 1;
    } else if (i < 509) {
        value -= 1;
    } else {
        i = 1;
    }

    io.analogOutput(0, value);
}
```

リスト 16　14 の実装例 1（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);
var dimmer:Osc = new Osc(Wave.TRIANGLE, 0.5, 0); // 波形、周波数、回数（0 は無限回）

gio.analogOutput(0).filters = [dimmer];
dimmer.start();
```

リスト 17　14 の実装例 2（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);

gio.analogOutput(0).filters = [new Osc(Wave.TRIANGLE, 0.5, 0)]; // 波形、周波数、回数（0 は無限回）
gio.analogOutput(0).filters[0].start();
```

## 4.4 デジタル出力 0 を一定時間だけ high にする

リスト 18　疑似コード

```
デジタル出力 0 を 20ms だけ high にする
例：ソレノイドを駆動して対象物を叩く（電流を流したままだとコイルが焼き切れるため時間を制限する）
```

リスト 19　18 の実装例（ActionScript 2 + Gainer）

```
var io:Gainer = new Gainer(...);

function startTrigger():void {
    io.digitalOutput(0, true);
    var tTimer:Timer = new Timer(20, 1); // 間隔、回数
    tTimer.addEventListener(TimerEvent.TIMER, finishTrigger);
    tTimer.start();
}

function finishTrigger():void {
    io.digitalOutput(0, false);
}
```

リスト 20　18 の実装例（ActionScript 3 + Funnel）

```
var gio:Gainer = new Gainer(...);
var trigger:Osc = new Osc(Osc.IMPULSE, 1000/20, 1); // 波形、周波数、回数

gio.digitalOutput(0).filters = [trigger];
trigger.start();
```

# 5 コマンドプロトコルについて

サーバとクライアント間の通信は OSC に準拠した形で行われます。サーバにはコマンド用、通知用の 2 つのポートが連番で用意されています（例：9000 番と 9001 番）。コマンド用のポートでのやりとりは全て同期型ですが、通知用のポートからの通知はサーバ側でイベントが発生した時点で任意に通知されます。

## 5.1 サーバに対するコントロール（コマンド用ポート）

| コマンド | 引数 | 説明 |
|---|---|---|
| /quit | none | サーバを終了する |
| /reset | none | I/O モジュールをリセットする |
| /polling | int | 入力のポーリングを開始（1）または停止（0）する |
| /configure | int, int, int, ... | モジュール ID、ポート設定 1、ポート設定 2… |
| /samplingInterval | int | I/O モジュールのサンプリング間隔をミリ秒単位で設定する |
| /out | int, int, float, ... | モジュール ID、最初のポート番号、値 1、値 2… |
| /in | int, int, int | モジュール ID、最初のポート番号、ポート数 |

## 5.2 サーバからのリプライ（コマンド用ポート）

| アドレス | 引数 | 説明 |
|---|---|---|
| **/コマンドと同じアドレス** | int, string | 第 1 引数はエラーコード、第 2 引数はエラーメッセージ |

## 5.3 サーバからの通知（通知用ポート）

| コマンド | 引数 | 説明 |
|---|---|---|
| /in | int, int, float, ... | モジュール ID、ポート番号、値 1、値 2… |

## 5.4 定数（型：int）

| 名前 | 値 | 説明 |
|---|---|---|
| PORT_AIN | 0 | アナログ入力（値は 0〜1.0） |
| PORT_DIN | 1 | デジタル入力（値は 0 または 1.0） |
| PORT_AOUT | 2 | アナログ出力（値は 0〜1.0） |
| PORT_DOUT | 3 | デジタル出力（値は 0 または 1.0） |
| NO_ERROR | 0 | エラーなし |
| ERROR | -1 | エラーあり |
| REBOOT_ERROR | -2 | I/O モジュールの再起動に失敗した |
| CONFIGURATION_ERROR | -3 | 指定したコンフィグレーションに間違いがある |

# 6 補足

## 6.1 ソフトウェアの動作環境について

### 6.1.1 PC

ActionScript 3 と Processing は、PC 上で動作することを想定しています。PC 側の処理能力を活用することにより、動画や音声などのリッチなメディアを自由に利用できます。

また、カメラを使った画像認識とのハイブリッド処理も考えられます。画像認識とセンサーによる検出は、それぞれ得意とする分野が異なります。これら 2 つを組み合わせることにより、プロセッサの負荷を軽減したり、精度を向上させたりすることが期待できると考えます。

### 6.1.2 組込み系デバイス

ここでの組込み系デバイスとは、Gumstix[*9]などの Linux が動作する小型デバイスを想定しています。組込み系デバイスの場合には、Ruby によるコントロールを想定しています。Ruby はインタプリタであり、リアルタイム性は保証されていません。確かに、1/1000 秒オーダーの処理は難しいかもしれませんが、「ネットワーク経由で RSS を取得し、その内容を解析して LED などのアクチュエータをコントロールする」という場合には十分なパフォーマンスが期待できます。

ActionScript 3 をはじめとする JavaScript を、Tamarin[*10]などの仮想マシンを組込み系デバイスに実装することで動作させることも考えられますが、Funnel v1.0 の正式リリース後の状況により検討したいと考えています。

[*9] http://www.gumstix.com/
[*10] http://www.mozilla-japan.org/projects/tamarin/

# 7 更新履歴

2007.11.22　/configure コマンドの第 1 引数にモジュール ID を追加。

2007.11.19　System クラスの名称を IOSystem に変更した。システム全体及び Funnel I/O モジュールの説明図を更新した。

2007.11.16　ユーティリティークラスとして Gainer および Arduino を追加し、複数モジュールに対応するための System クラスを追加した。

2007.10.31　AS3 のサンプルで変数の型に float を指定していた部分を Number に修正。XBee に関する定義を追加。ビルド 001 で追加された Configuration を利用した形式にサンプルを修正。

2007.9.29　Configuration クラスを追加。Funnel クラスに analogInput() などのメソッドを追加。

2007.9.22　FIO の構成を XBee または XBee Pro を想定したプランに変更。Funnel クラスと Port クラスの説明を分割。 その他細かな記述上の間違いを修正。

2007.9.13　Port のインスタンス変数に lastValue を追加。Event から value と lastValue を削除し、PortEvent に target を追加。

2007.9.9　Osc.update() に引数ありのバージョンを追加。Osc に reset() メソッドを追加。Scaler.LOG と Scaler.EXP を削除。Event の説明を追加。

2007.9.4　setFilters() メソッドを削除。SetPoint の引数の形式を変更。

2007.9.2　Threshold を SetPoint に改名し、複数のポイントをセットできるように変更。Scaler にリミッタを追加。

2007.9.1　致命的なエラーを個別のイベントに変更。Osc のサービス間隔をクラス変更に変更し、start メソッドを追加。Osc にインスタンス変数 times と、インスタンスメソッド stop() を追加。

2007.8.17　イベントハンドラを廃止してイベントリスナに変更。

2007.7.3　ポートのプロパティ hysteresis、および OSC コマンドの定数 PORT_DIN の説明に誤りがあったのを修正。