



HASC Tutorial 2011

<http://hasc.jp/>

スライドコンテンツ

- HASC Tool内部構成の解説
- HASC Toolの独自開発（座学）
- HASC Toolの独自開発（演習）

Part 3

名古屋大学 河口研究室
修士2年 小川延宏

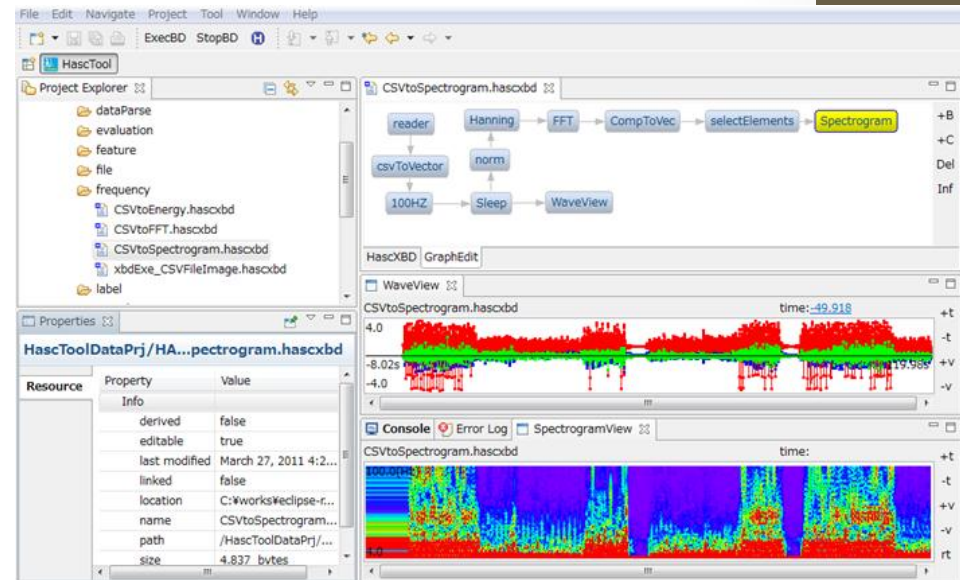
HASC Tool

- HASCが開発する行動認識ツール

- 行動データ収集機能
- 行動データ処理機能
- ラベル付与機能
- 機械学習機能

- HASCToolの種類

- ダウンロード版：お試し版
- Sourceforge版：開発者向け

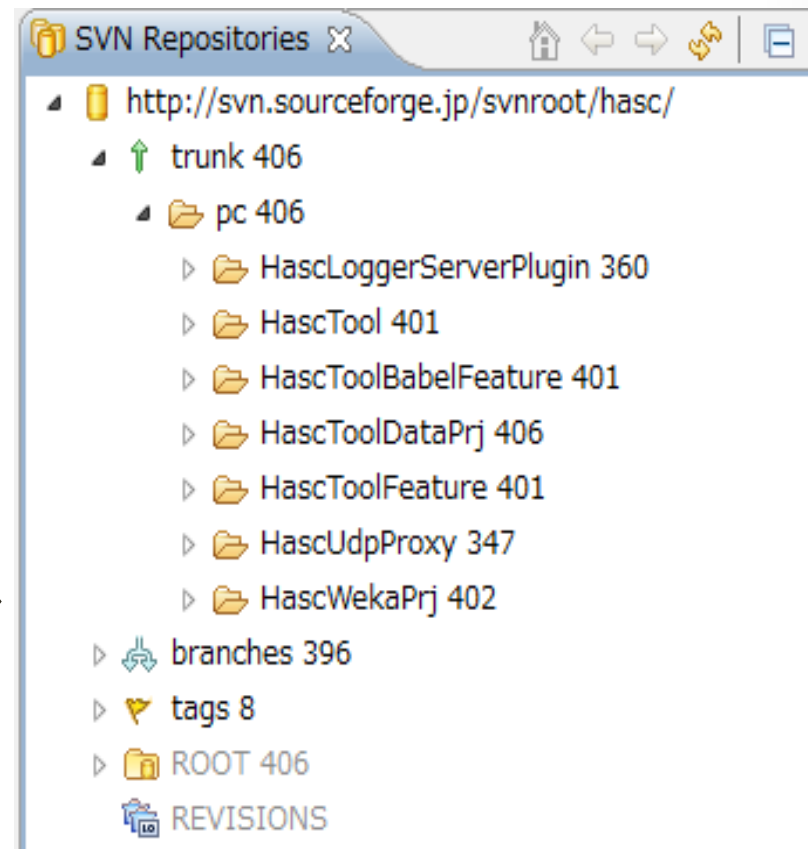


HASC Tool 内部構成

- HASC Tool Projects
 - HascLoggerServerPlugin
 - 行動データ収集機能
 - HascTool
 - 行動データ処理機能
 - ラベル付与機能
 - HascToolDataPrj
 - サンプル行動データ
 - サンプル実行ファイル
 - HascWekaPrj
 - 機械学習機能

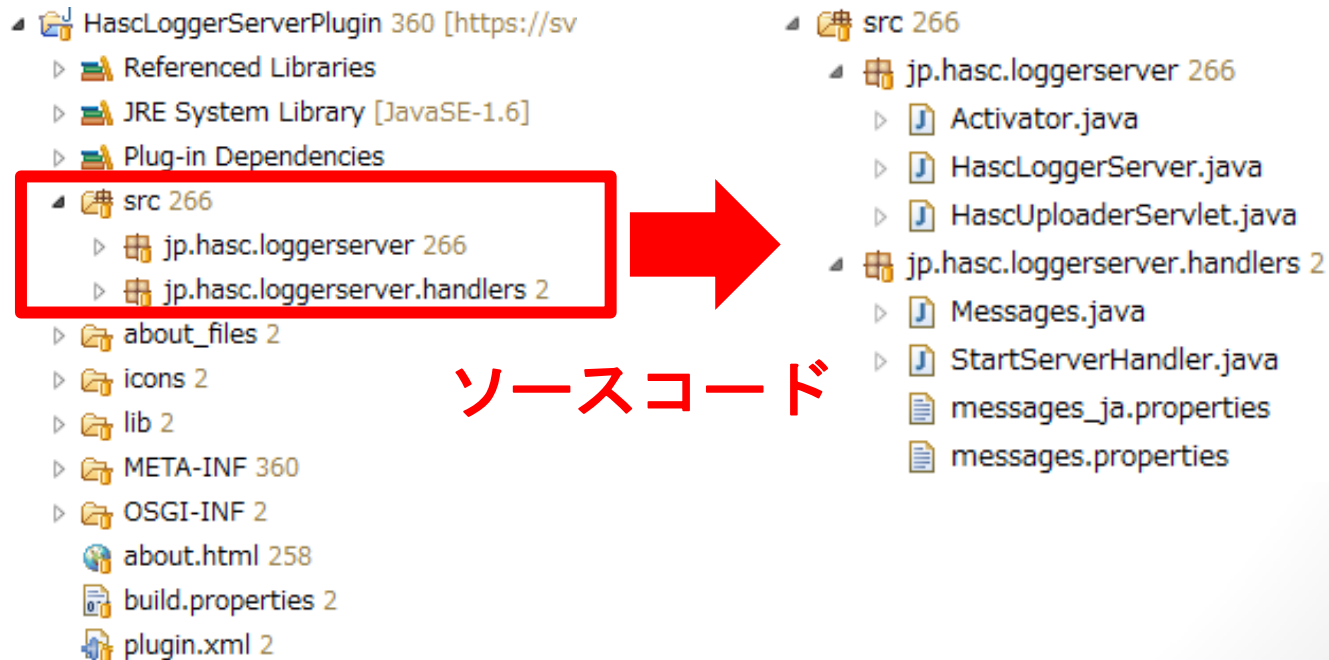
アクセス⇒

<http://svn.sourceforge.jp/svnroot/hasc/>



HascLoggerServerPlugin

- 行動データ収集機能
 - HASC Loggerで収集した行動データを受信
- Projectの構成

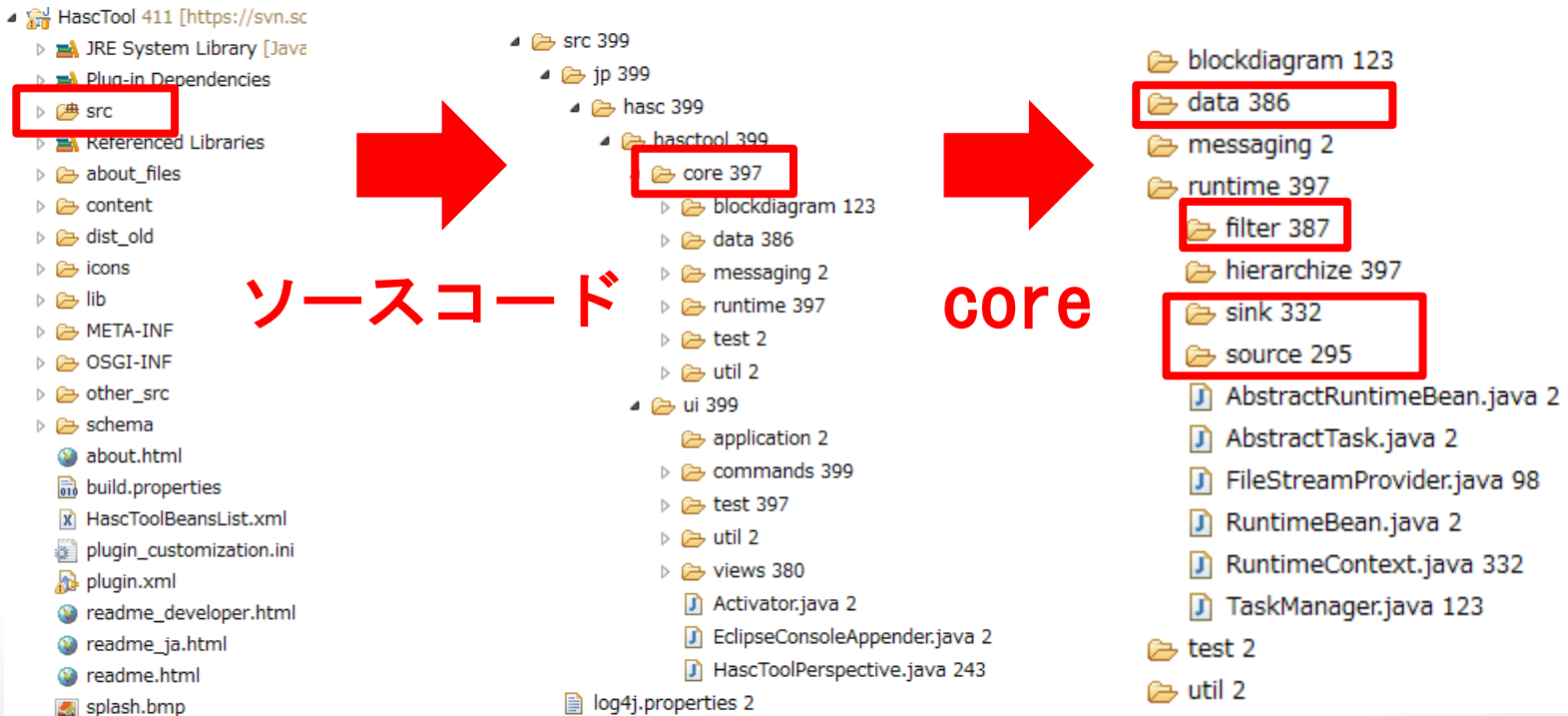


HascTool

- ラベル付与機能
- 行動データ処理機能
- Projectの構成

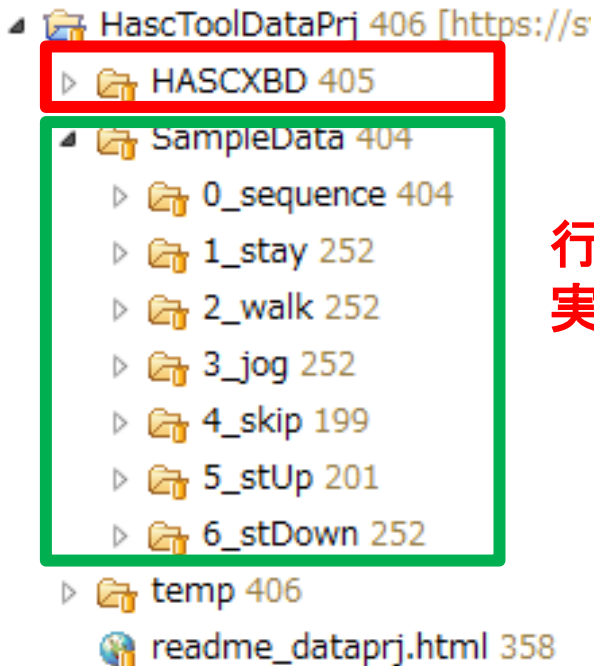
行動データ
処理で重要
な部分

- data
- filter
- sink
- source



HascToolDataPrj

- サンプル行動データ : **HASCXBD**
- サンプル実行ファイル : **SampleData**
- Projectの構成



**行動データ処理
実行ファイル**



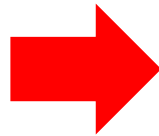
HascWekaPrj



- 機械学習機能
 - WEKAとの連携
- Projectの構成

<http://www.cs.waikato.ac.nz/ml/weka/>

- ▶ HascWekaPrj 402 [https://svn.so
 - ▶ Referenced Libraries
 - ▶ JRE System Library [JavaSE-1]
 - ▶ Plug-in Dependencies
 - ▶ **src 378**
 - ▶ about_files 272
 - ▶ lib 271
 - ▶ META-INF 360
 - ▶ about.html 271
 - ▶ build.properties 402
 - ▶ fragment.xml 271
 - ▶ HascWekaBeansList.xml 376



ソースコード

- src 378
 - jp.hasc.hasctool.core.runtime.filter.weka
 - jp.hasc.hasctool.core.runtime.filter.weka.file
 - jp.hasc.hasctool.core.runtime.filter.weka.label
 - jp.hasc.hasctool.core.runtime.filter.weka.learning
 - jp.hasc.hasctool.core.runtime.filter.weka.message

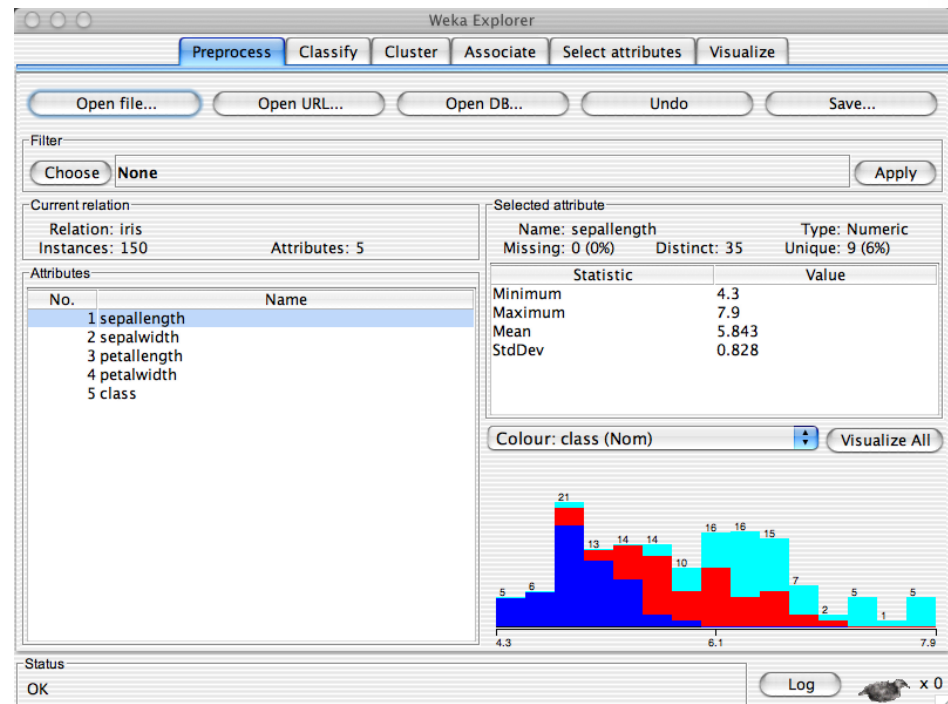
学習器 :

learningディレクトリ内にあります

- jp.hasc.hasctool.core.runtime.filter.weka.learning :
 - J48tree.java
 - KNNClassifier.java
 - LearningClassifier.java
 - MachineLearning.java
 - RBFlearner.java
 - SVMlearner.java

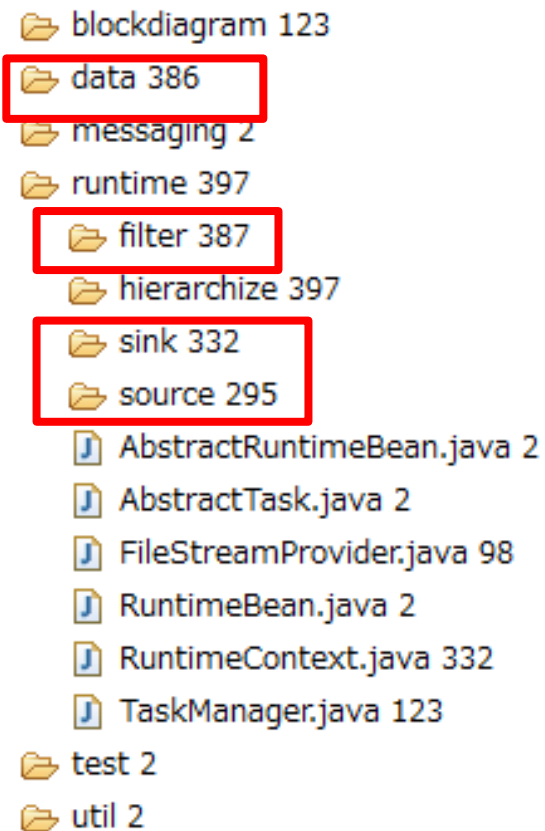
機械学習ツール：WEKA

- Waikato大学で作成された機械学習ツール
 - <http://www.cs.waikato.ac.nz/ml/weka/>
 - 使用言語：Java
- オープンソース
 - GPNライセンス



HascToolの独自開発（座学）

- HascTool
 - ブロック図で様々な処理を記述可能
 - Javaで独自のブロックを開発可能
- メッセージの定義
 - ブロック間のやりとり
 - data
- 新規フィルタを追加
 - filter
 - sink
 - source



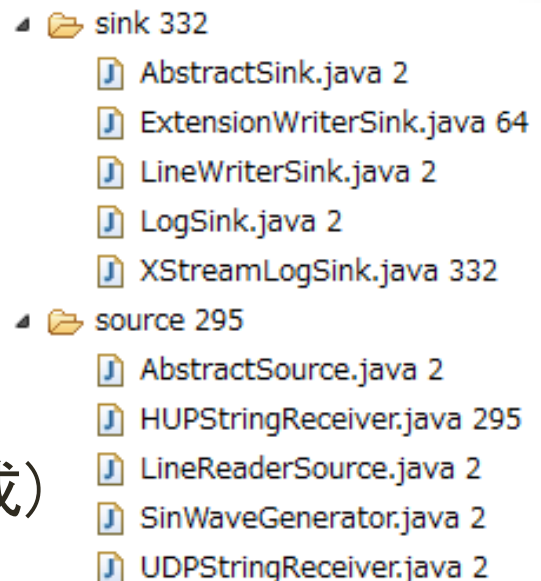
Data (メッセージ)

- ブロック間のデータ送受信
 - 任意のオブジェクトをメッセージとする
- メッセージの紹介
 - ArrayVectorSignalMessage
 - 時刻情報 + 実数値の配列
 - ScalarSignalMessage
 - 時刻情報 + 実数値
 - ComplexVectorSignalMessage
 - 時刻情報 + 複素数値の配列
 - NullSignalMessage : 開発用

```
data 386
├─ AbstractSignalMessage.java 2
├─ ArrayVectorSignalMessage.java 2
├─ Complex.java 2
├─ ComplexVectorSignalMessage.java 2
├─ DefaultVectorSignalComparator.java 2
├─ EnumCommand.java 2
├─ LabelInfo.java 2
├─ LabelSignalMessage.java 2
├─ NullSignalMessage.java 2
├─ ObjectSignalMessage.java 2
├─ ScalarSignalMessage.java 2
├─ SignalMessage.java 2
├─ Spline.java 386
├─ VectorSignalMessage.java 2
└─ VectorSignalMessages.java 2
```

Sink / Source

- HASCToolからデータを出力 : sink
 - LineWriterSink
 - データをファイルに書き出し
- HASCToolにデータを入力 : source
 - LineReaderSource
 - ファイルから読み込み
 - UDPStringReceiver
 - UDP形式でデータを受信
 - SinWaveGenerator
 - 行動データを生成 (Sin波を生成)



Filter

- 行動データ処理を実行

- file

- ファイル関連（例：パーサ）

- Frequency

- フーリエ変換関連

- interpolator

- サンプリング補完関連

- label : ラベルデータ関連

- meta : メタデータ関連

- time : 時間パーサなど

- window : 窓関数

- filter 387
 - evaluation 189
 - file 285
 - frequency 109
 - interpolator 387
 - label 167
 - meta 195
 - tick 2
 - time 85
 - window 106
 - AbsoluteFilter.java 131
 - AbstractFilter.java 2
 - AbstractScalarFilter.java 2
 - AngleVariationFilter.java 240
 - ConcatenateVectorAndLabelFilter.java 29
 - DelayFilter.java 2
 - IIRFilter.java 82
 - MaxFilter.java 168
 - MeanFilter.java 66
 - Messages.java 248

Filter

- ここから、ソースコード的な説明
- RuntimeBeanインタフェースを実装した任意のクラス
 - デフォルトコンストラクタ
 - プロパティをセットするためのsetterメソッド
 - 入力/出力ポートを返すgetterメソッド

サンプル

- スカラ値に、valueToAddプロパティの値を加算する
1 入力 1 出力フィルタ

```
public class SampleAddFilter
    implements Runnable, MessageProcessor
{
    // valueToAdd プロパティ
    private double valueToAdd_ = 0;
    public void setValueToAdd(double valueToAdd) {
        this.valueToAdd_ = valueToAdd;
    }

    // outputPort
    private MessageConnector outputPort_ =
        new MessageConnector();
    public MessageConnector getOutputPort() {
        return outputPort_;
    }

    // inputPort
    public MessageProcessor getInputPort() {
        return this;
    }

    // Runnable interface
    @Override
    public void run() {}
}
```

```
// MessageProcessor interface
@Override
public void processMessage(Object message)
    throws InterruptedException
{
    if (message instanceof ScalarSignalMessage) {
        // ScalarSignalMessageの場合
        ScalarSignalMessage inValue =
            (ScalarSignalMessage)message;
        // 新しい値を計算
        double newValue =
            inValue.getScalarValue() + valueToAdd_;
        // 結果を出力
        outputPort_.processMessage(
            new ScalarSignalMessage(
                inValue.getTime(), newValue));
    } else {
        // その他のメッセージ (BEGIN, ENDなど) は
        //そのまま出力
        outputPort_.processMessage(message);
    }
}
```

入力・出力ポート

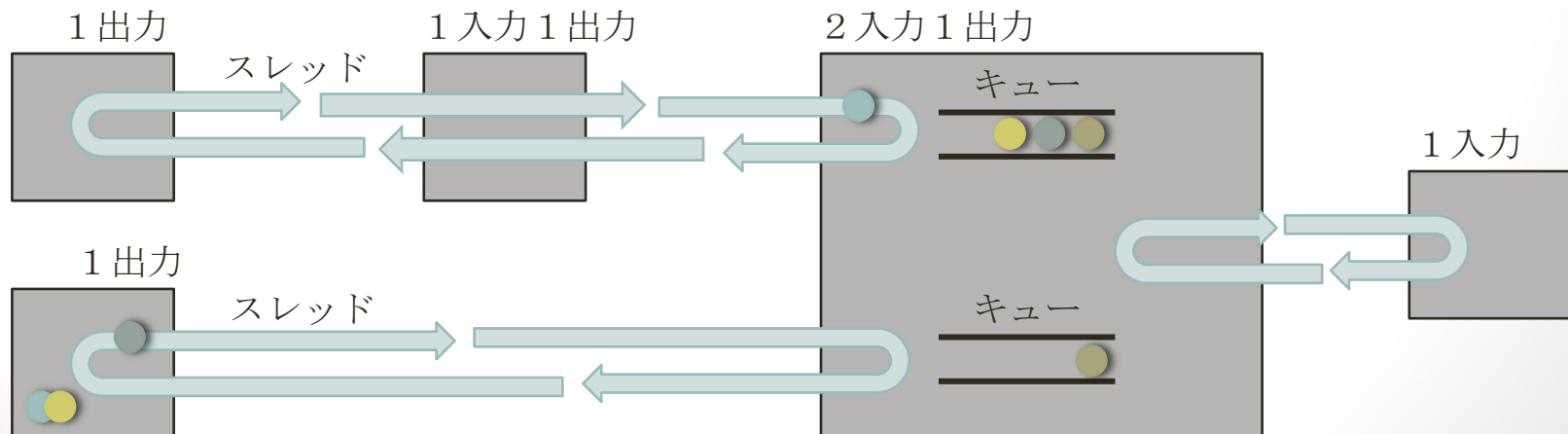
- 入力ポート
 - MessageProcessor インタフェースを実装したオブジェクト
 - 入力ポートのprocessMessageメソッドで、メッセージ受信時の処理を実装
- 出力ポート
 - MessageConnector オブジェクト
 - 出力ポートのprocessMessageメソッドを呼び出して、メッセージを送信
 - 単に、接続先の入力ポートのprocessMessageメソッドを呼び出しているだけ
 - connectメソッドでポート間を接続

ヘルパークラス

- AbstractFilter
 - RuntimeBeanを実装した抽象クラス
 - 1入力1出力のブロックを作るのに使う
- AbstractSink
 - RuntimeBeanを実装した抽象クラス
 - 1入力、出力無し of ブロックを作るのに使う

別スレッドでの処理

- 入力ポートのprocessMessageメソッドは、入力ポートごとに、別々のスレッドから非同期に呼び出される可能性がある
 - 1入力ブロックの場合：メッセージ受信時に、同じスレッドで処理を行って良い。（別スレッドで行っても良い）
 - 多入力ブロックの場合：スレッド間の同期のため、メッセージをキューにためておき、別スレッドで処理を行う
 - 入力無し（出力のみ）のブロックの場合：別スレッドで処理



別スレッド処理に使うクラス

- AbstractTaskクラス

- RuntimeBeanを実装した抽象クラス
- 別スレッドで処理を行う場合に使う
- サブクラスでrunメソッドを実装して処理を記述（Threadクラスと同様）

- MessageQueueクラス

- MessageProcessorを実装したクラス（入力ポート用）
- processMessageメソッドで渡されたメッセージをキューにためておき、別スレッドから取り出せる

ヘルパークラス

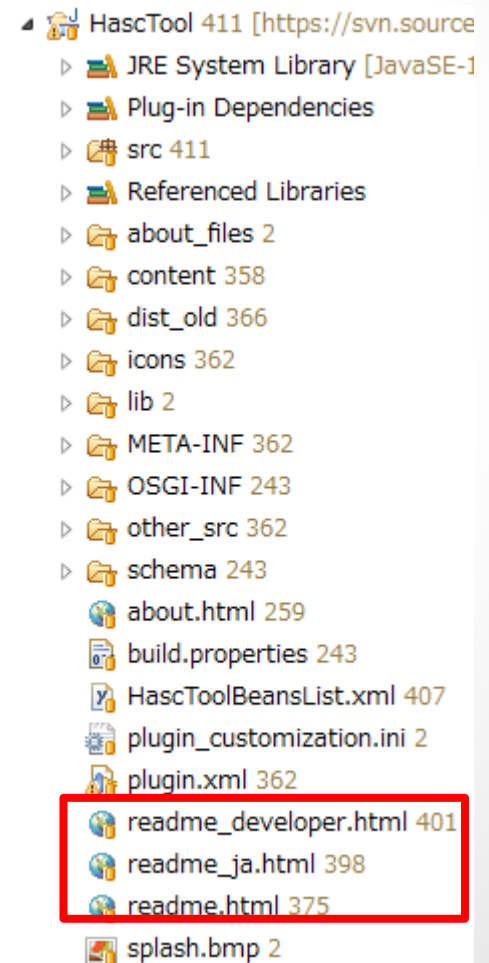
- AbstractMultipleInputsFilterクラス
 - AbstractTaskを継承した抽象クラス
 - 多入力 1 出力のブロックを作る時に使う
 - 全入力ポートから入力された信号メッセージの、時刻が同期され、processSignalMessagesメソッド（サブクラスで実装）が呼び出される
 - サンプル：VectorAdderクラス
- AbstractSourceクラス
 - AbstractTaskを継承した抽象クラス
 - 入力無し、1 出力のブロックを作るのに使う

その他参考

- 独自ブロックの開発方法の概要を説明

- 詳細は、以下を参照

- readme.html
 - ブロックリファレンス
 - メッセージリファレンス
- readme_developer.html
 - 開発環境のセットアップ
- 各ブロックのソースファイル



HascToolの独自開発（演習）

- Filterの作成
 - 参考：SampleFilter1
 - SampleFilter1をコピー
 - 名前を変えて貼り付け
 - 今回はTestFilter1

- SampleFilter1.java 248
- SampleFilter2.java 23
- ScalarAdder.java 2
- ScalarMultiplier.java 2
- SignalTimeSorter.java 2
- SleepFilter.java 2
- TestFilter1.java**
- VarianceFilter.java 66

- src
 - jp.hasc.hascTool.core.blockdiagram
 - jp.hasc.hascTool.core.blockdiagram.model
 - jp.hasc.hascTool.core.blockdiagram.test
 - jp.hasc.hascTool.core.data
 - jp.hasc.hascTool.core.messaging
 - jp.hasc.hascTool.core.runtime
 - jp.hasc.hascTool.core.runtime.filter
 - AbsoluteFilter.java
 - AbstractFilter.java
 - AbstractScalarFilter.java
 - AngleVariationFilter.java
 - ConcatenateVectorAndLabelFilter.java
 - DelayFilter.java
 - IIRFilter.java
 - MaxFilter.java
 - MeanFilter.java
 - Messages.java
 - MinimumFilter.java
 - SampleFilter1.java**
 - SampleFilter2.java
 - ScalarAdder.java
 - ScalarMultiplier.java

TestFilterのソースコード

```
TestFilter1.java ×
package jp.hasc.hasctool.core.runtime.filter;

import jp.hasc.hasctool.core.data.VectorSignalMessage;

/**
 * AbstractFilterを使ったフィルタのサンプル。
 * ベクトルの各要素に0.5を掛けて1を加算する。
 * @author iwasaki
 */
public class TestFilter1 extends AbstractFilter {
    private double constantToMultiply_ = 0.5;
    private double constantToAdd_ = 1;

    public static String getRuntimeBeanDescription() { return Messages.SampleFilter1_description; }

    @Override
    public void processMessage(Object message) throws InterruptedException {
        if (message instanceof VectorSignalMessage) {
            // VectorSignalMessageの場合
            VectorSignalMessage vsig=(VectorSignalMessage)message;
            // 新しいベクトルの値を計算
            double[] newValue=new double[vsig.getVectorSize()];
            for(int i=0;i<newValue.length;++i) newValue[i]=vsig.getVectorElement(i)*constantToMultiply_+constantToAdd_;
            // 結果を出力
            outputMessage(VectorSignalMessages.create(vsig.getTime(), newValue));
        }else{
            // その他のメッセージ(BEGIN, ENDなど)はそのまま出力
            outputMessage(message);
        }
    }
}
```

Filterの編集

- processMessage

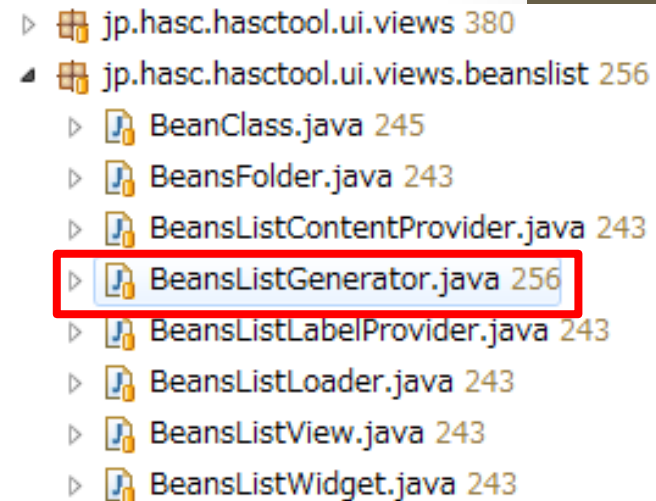
- if (message instanceof **VectorSignalMessage**)
 - VectorSignalMessageを入力として受け付ける

```
// VectorSignalMessageの場合
VectorSignalMessage vsig=(VectorSignalMessage)message;
// 新しいベクトルの値を計算
double[] newValue=new double[vsig.getVectorSize()];
for(int i=0;i<newValue.length;++i) newValue[i]=vsig.getVectorElement(i)*constantToMultiply_+constantToAdd_;
// 結果を出力
outputMessage(VectorSignalMessages.create(vsig.getTime(), newValue));
```

- **double[] newValue =**~~
 - 実際に処理を行っている部分
 - ベクトルの各要素に**0.5**をかけて**1**を加算
- **outputMessage** (~~)
 - 出力を行っている部分
 - () 内に出カメッセージを記述

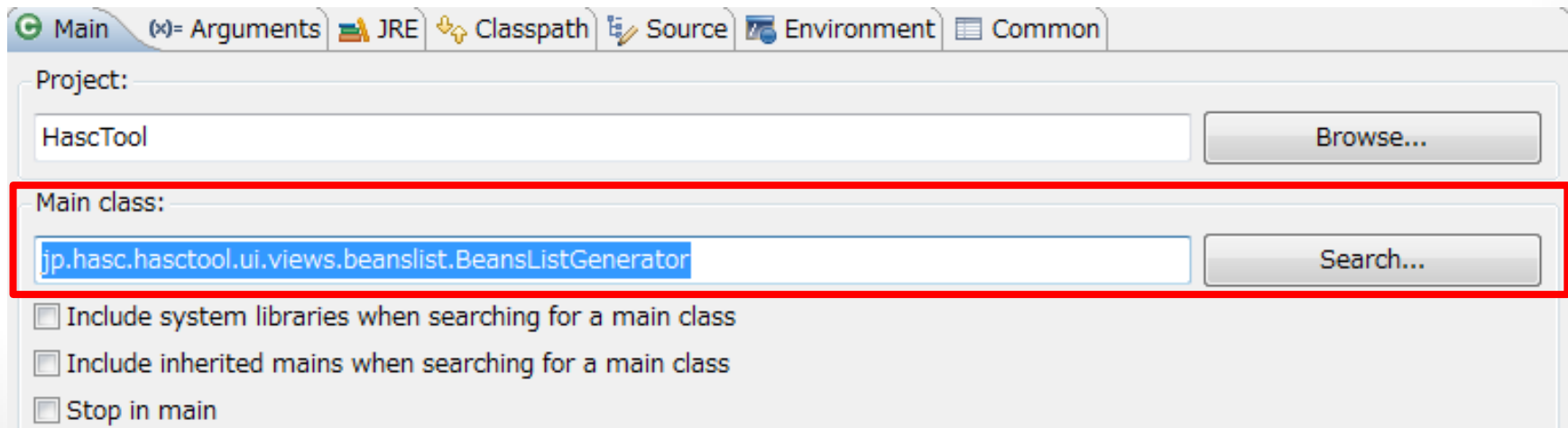
Block Class Listの更新

- **HascToolBeansList.xml** を更新
- 実行ファイル
 - `jp.hasc.hasctool.ui.views.beanslist.BeansListGenerator`
 - 上記クラスを右クリック > Run As > Run Configurations を選択
 - **Java Application** を右クリック > New
 - **Name** を設定
 - BeansListGenerator
 - Run Configuration の設定
 - Main
 - Arguments



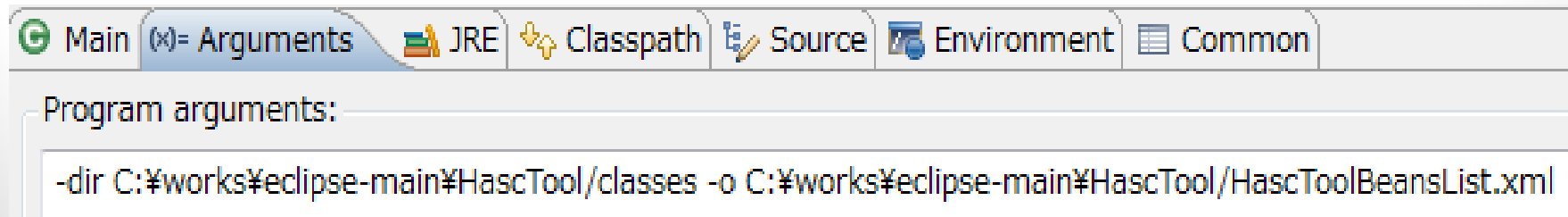
Mainタブ

- Main class
 - `jp.hasc.hasctool.ui.views.beanslist.BeansListGenerator`



Argumentsタブ

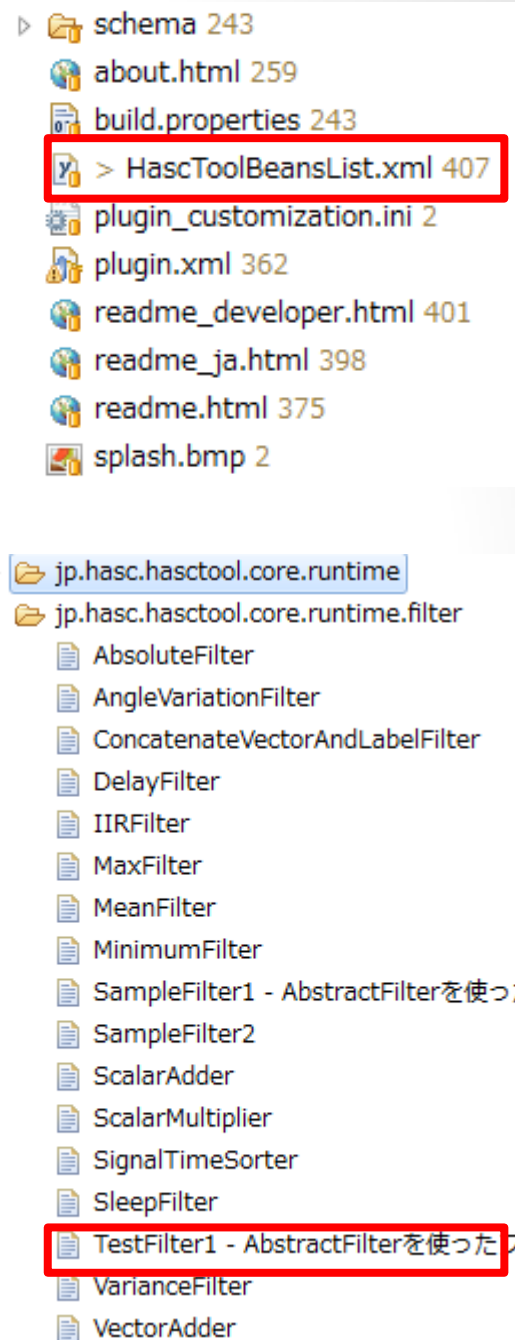
- Program Arguments:
 - -dir クラスファイルのルートディレクトリのパス (HascToolのパス/classes)
 - -o 出力先xmlファイルのパス (HascToolのパス/HascToolBeansList.xml)
 - “-dir” と “-o” の間に半角スペース
- Applyを選択
- Runを実行



```
Program arguments:  
-dir C:\works\eclipse-main\HascTool\classes -o C:\works\eclipse-main\HascTool\HascToolBeansList.xml
```

作成したFilterの使用

- Eclipseを再起動
 - HascToolBeansList.xml更新
 - HASCToolを実行
 - HascToolを右クリック> Run As> Run Configurations> HascToolの起動
 - Block Class List
 - 新しいフィルタがリストに追加
 - 選択するとブロックが生成
- HASCXBDファイルの編集



スマートフォンでのセンシング

iOSのセンサーAPI

- Core Motion Framework
 - iOS 4.0以降用のAPI
 - 加速度、ジャイロ、磁気
 - 姿勢（ロール、ピッチ、ヨー） – 複数センサーを統合
- Core Location Framework
 - 位置情報、方角（磁気）
- UIAccelerometer クラス
 - 加速度

iOS サンプルコード

- Core Motion で加速度を取得する例
 - 他のセンサの場合も同様

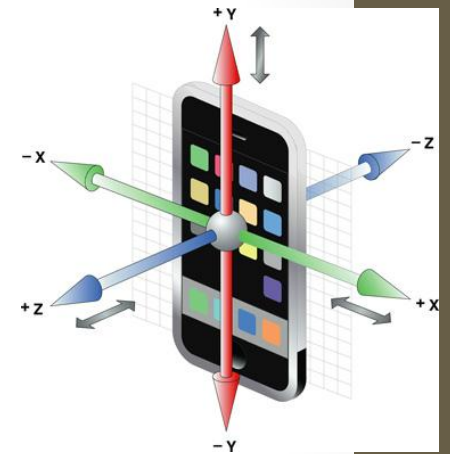
```
CMMotionManager *manager = [[CMMotionManager alloc] init];
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
```

```
// サンプリングレート 100Hz
manager.accelerometerUpdateInterval = 1.0/100;
```

```
// 加速度取得開始
```

```
[manager startAccelerometerUpdatesToQueue:queue withHandler:
^(CMAccelerometerData *data, NSError *error) {
    // 加速度データを取得した時に呼び出されるブロック
    NSLog(@"%f,%f,%f,%f", data.timestamp,
        data.acceleration.x, data.acceleration.y, data.acceleration.z);
}
];
```

Event Handling Guide for iOS より



AndroidのセンサーAPI

- android.hardware パッケージ
 - SensorManager, Sensor, SensorEventクラス
 - SensorEventListener インタフェース
- 様々な種類のセンサが定義されている
 - 加速度、ジャイロ、磁気、光、温度、圧力、...
- サンプリングレート (delay) の指定が4段階
 - FASTEST, GAME, UI, NORMAL

Android サンプルコード

- 加速度を取得する例
 - 他のセンサの場合も同様

```
protected void onResume() {
    super.onResume();
    mManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    // 加速度センサを取得
    mAccelerometer = mManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    // FASTESTレートで、加速度センサをサンプリング開始
    mManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_FASTEST);
}

// センサ値が変化した時に呼び出される (SensorEventListenerインタフェース)
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        Log.d("ACC", event.timestamp + "," + event.values[0] + "," + event.values[1] + "," + event.values[2]);
    }
}
```

Android 補足

- 加速度の単位がiOSと異なる
 - iOSは (G), Androidは (m/s^2)
- サンプリングの頻度
 - 値がある程度変化しないと、onSensorChanged() が呼び出されない
 - 呼び出される頻度は、機種により異なる
- 外部ストレージ (SDカード) への保存が遅い
 - ログの書き込み速度よりも、加速度データの来る速度のほうが速いことも