Text Parser Library

Ver. 1.1

Design Description

Center of Research on Innovative Simulation Software
Institute of Industrial Science
the University of Tokyo

http://www.iis.u-tokyo.ac.jp/

October 2012

目次

1 概要

C/C++,Frotran などで記述されたシミュレーションソフトウェアにおいて, パラメータを記述, 取得する機能を提供するクラスライブラリを構築する. ユーザが指定するアスキーファイルに記述されたパラメータを読み込み, クラスライブラリ内部にそのパラメータを保持し, シミュレータから取り出せる機能を提供する. さらに, マルチプラットホームで動作することにより, 多くのシミュレーションプログラムでの利用が可能になる.

2 パラメータの記述方式

本ライブラリが扱うパラメータデータベースはノードの階層構造にリーフ（ラベル/値のペア）を格納したものである. パラメータデータベースの主な構成要素はラベル, ノード, リーフ, 値である.

2.1 ラベル

ラベルは, ノードまたはリーフを表す識別子である. 以下にラベルの仕様を示す.

- ノードとリーフの識別子である.

  [ノード]
  ラベル `{ }`
  [リーフ]
  ラベル = 値

- “=” または “{” または @dep (後述) 内の “==” または @dep “!=” の記号の左側にある, 空白を途中に含まない文字列とする. 記号との間に改行や空白があっても構わない.
  - 二重引用符 [“] で囲まれた文字列は, 二重引用符で囲まれていない文字列と等値である.
  - 通常使用できる文字は [a-zA-Z0-9_-] である.
  - ラベルはノードパス的な表記（以下, ラベルパスと呼ぶ）も可能であり, 絶対パス, 相対パスで表すことが出来, パスの記述用に [/.] も使用出来る.
  - 文字列の比較において大文字/小文字の区別はしない.

  以下に例を示す.

  [通常]
  `foo` **または** `"foo"` **または** `FOO` **または** `"FOO"`
  この 4 **つの記法は全て同じラベルになる.**
  [絶対パス]
  `/foo/baz` **または** `"/foo/baz"`
  [相対パス]
  `./foo/baz` **または** `"./foo/baz"`
  `foo/baz` **または** `"foo/baz"`
  `../bar` **または** `"../bar"`

- 同じノード階層内にあるノードには同一のラベルを付ける事はできない. 又, 同じノード階層内にあるリーフには 同一のラベルを付ける事はできない.

可能な例

```
foo {
    qux { baz = "val1" }
    qux { x = "val2" }
}
```

エラーになる例

```
foo {
    qux = 1
    qux = 2   // エラー
}
```

- 同じノード階層内に同一ラベルのノードとリーフが存在してはならない.

```
foo {
    qux { baz = "val1"}
    qux = "val2"   // エラー
}
```

- 絶対パスに同じラベルが複数含まれてはならない.

```
qux {
    qux {         // エラー
         baz = "val1"
    }
}
qux {
    qux = "val1"   // エラー
}
```

- ラベル文字列末尾に"[@]"と書くことで, 配列形式のラベルを定義できる. 配列形式のラベルはパースされた時点で@がノード内で初期化された一意な配列添え字に変換される.

```
 foo {
     bar[@] { param[@]=1,
              param[@]=2
     }
```

```
     bar[@] { param[@]=3,
              param[@]=4
     }
}
```

パース後，以下の様に変換

```
foo {
    bar[0] { param[0]=1,
             param[1]=2
    }
    bar[1] { param[0]=3,
             param[1]=4
    }
}
```

- 同一ノード内でノードとリーフに同じ配列形式ラベルを定義することは出来ない.

```
foo {
    param[@] { abc = 1 }
    param[@]=1    // エラー
}
```

- 絶対パスに同じ配列ラベルが複数含まれてはならない.

```
param[@] {
         param[@] {
                  param[@] = 1
         }
}
```

- ラベルごとにインデントを揃える必要はない. 下記の 2 例は等価である.

例 1

```
one {
   two  {
                three = 1
         }
}
```

例 2

```
one {
two{
three = 1
}
}
```

2.2 ノード

ノードは, ラベルで名前, 位置を定義されたもので, リーフまたはノードを保持するものである. 以下にノードの仕様を示す.

- ノードは, リーフ又はノードを保持する. ノードはラベルで始まり, その後の左中括弧"{"でノード内要素が始まり, 右中括弧"}"で終わる. ノードはネストすることができる. 下記 3 例はいずれも等価である.

ラベル { }
(例)

```
foo {
}
foo  {    }
foo
{
}
```

- 相対パス, 絶対パスによる定義が可能である. 相対ノードパスは, 一階層上を表す "../" と 当該のノードを表す "./" が使用できる.

絶対パス

```
/foo/qux {
}
```

相対パス 次の 2 例は等価である.

例 1

```
foo {
    ./qux/abc  {  //  ./は省略可能
       ../cde/ {    }
    }
```

```
}
```

例 2

```
foo {
    ./qux/abc { }
    ./qux/cde { }
}
```

- 一つのノードを分割して定義することが可能

```
foo {
    baz=1
}
foo {
    abc=2
}
```

- ノードの階層ごとに括弧でくくるが, その位置を揃える必要はない. また, 改行の有無に左右されない. 下記の 2 例は等価である.

例 1

```
one { two{ three = 1  }
  }
```

例 2

```
one          {
 two{       three = 1
   }
        }
```

2.3 リーフ

本ライブラリではノード内要素のラベル/値のペアをリーフと呼ぶ. 以下にリーフの仕様を示してある.

- リーフはラベルと値が代入演算子”=”で接続されたものである. 複数のリーフはカンマ”,”もしくはスペース” ”や改行によって区切られる.

ラベル = 値

```
qux="val"
param0=1, param1=2
param0=1 param1=2
```

- 相対パス, 絶対パスによる定義

絶対パス

```
/foo/qux="val1"
```

相対パス

```
./bar/baz=1   //   ./は省略可能
```

- リーフはパラメータファイル内でどのような順で記述されても良い.

2.4 値

リーフでラベルとペアになっているものを値と呼ぶ. 値には, 5 種類あり, それぞれ次のような仕様になっている.

(a) 文字列
- 二重引用符""で囲まれた ASCII 文字列
- 使用できる文字は [a-zA-Z0-9_-] である.
- 文字列の比較において大文字/小文字の区別はしない.

文字列の例

```
"ON" "on"
```

(b) 数値
- 整数
  - “(+/-) 0-9” の “+/-” の後にスペースが有っても可
  - 余分な 0 が有っても可

  数値の例

  ```
  123
  0
  + 123
  - 0
  01
  ```
- 浮動小点数
  - “(+/-) 0-9.(0-9)”, “(+/-) .0-9 ” の “+/-” の後にスペースが有っても可

浮動小数点の例

```
0.10
+ 0.2
- 0.3
3.
.123
```

- 指数
  - “(+/-) 0-9.(0-9) e/d (+/-) 0-9”, “(+/-).0-9 D/E (+/-) 0-9” の “+/-” の後にスペースが有っても可
  - 同じく e/d の前後にスペースが有っても可

指数の例

```
1.0 e 10
+ 0.1 E - 5
- 1.2 d 3
.3 e - 4
```

(c) ベクトル

- 文字列もしくは数値の順序付けされたセットである.
- 左括弧”（”で始まり, 右括弧”）”で終わる.
- 値はカンマ”, ”で区切られる.
- １つのベクトル内の値の要素は全て同じ型でなければならない.
- 未定義値（UNDEF）のベクトルも可能.
- ベクトルのネストはできない.

ベクトルの例

```
(1.0 e 10, 1.0 e 10 , 1.0 e 10 )
(+ 0.1 E - 5 ,- 0.3 E - 5 )
(- 1.2 d 3,- 1.2, 3 ,.3 e - 4)
("ONE","TWO","THREE")
( UNDEF, UNDEF, UNDEF )
```

(d) 依存関係付き値

- "@dep"に続き指定されるC言語ライクな三項演算子による条件式に応じて当該の値を制御可能である.
- "@dep"に続く左括弧”（”で始まり, 右括弧”）”で終わる部分に条件式を記述する.
- 条件式の左辺は依存するパラメータのラベルパス, 右辺はその取りうる値である.
- ラベルパスは相対パスも可能.
- ラベルパスとして配列形式ラベル（例：”param[@]”）を使うことは出来ない.
- 条件式の右辺の値は一つの文字列又は数値に限る.
- 条件式の右辺の値としてベクトルや未定義値, 別の依存関係付き値を指定することは出来ない.
- 条件演算子には"=="と"!="が利用できる.
- 条件式は"&&"と"||"演算子で接続し, 複数指定可能で括弧の順に処理する.

- 条件式に無駄な括弧が有っても可能.
- 条件式の右辺の値が数値の場合は double に変換して"=="と"!="を判定する.
- 条件式に続いて"?"の次に条件式が「真」の時の値, 次に":"に続いて条件式が「偽」の時の値を記述する.
- 値はベクトルでも可能. ":"の左右でベクトルの次元が異なっても良い.
- ":"の左右で値の型が異なっても良い.
- 又, 値は未定義値でも可能だが, 値として別の依存関係付き値を指定することは出来ない.
- 依存関係付き値で使用されるラベルが定義されていない場合は全パラメータのパース終了後に再度評価を行なう.
- 最終的に定義されていないラベルを参照する依存関係付き値には未定義値 (UNDEF) をセットし, 警告を表示する.

依存関係付き値の書式

`@dep ( 条件式 ) ? 値 : 値`

条件式

ラベル == 値

ラベル != 値

条件式 && 条件式

条件式 || 条件式

依存関係付き値の例（例）

```
@dep (swt == "on") ? 1 : 2
@dep ((swt == "on")) ? 1 : 2
@dep ( a == 1) ? (1,2) : (0,1,2)
@dep ( a == 1) ? (1,2) : ("a","b")
@dep ( a == 1) ? 5 : UNDEF
@dep ( (swt1==0) && (swt2==1) ) ? "a" : "b"
@dep ( (swt1==0) || (swt2!=1) ) ? "a" : "b"
```

(e) 未定義値

- ラベルの値に対し未定義であることを明示するため未定義値（UNDEF）を記述することが出来る.
- 未定義値はパラメータのパースの際に警告を表示する. また, パラメータを取得する際に未定義値だったら警告を表示する.

未定義値の例

```
baz=UNDEF
```

2.5 改行

パラメータの記述は要素の単位で改行が可能である. 途中で改行できないのはラベル, 文字列の値, 数値, 未定義値 (UNDEF), 依存関係の"@dep"である.

2.6 コメント

コメントは C/C++ ライクな/*〜*/, 及び//〜が利用できる.

3 機能

本ライブラリは以下の機能を有する.

3.1 パラメータファイルの読み込み

パラメータパースクラスをインスタンス化し, ロード命令によりファイルをオープン, クラス内部のパラメータデータ構造にパラメータを一括読み込みする.

3.2 パラメータ依存関係のチェック

パラメータに依存関係が指定されている場合, その正当性をチェックする. 不適な場合には適切なコメントやライン番号などを標準エラー出力する. 依存性のチェックは, 一旦すべてのパラメータを読み込んだ後で, 逐次処理する.

3.3 パラメータの保持

STL コンテナクラスを用いて, パラメータ階層構造を保持. 値は文字列として保持する.

3.4 パラメータの型変換

取得したパラメータの型を変換する. 整数, 浮動小数点への変換機能を提供する. 本ライブラリを利用するシミュレータ開発者は, パラメータの構造, 値の型などを知っているので, 目的のパラメータを取得後, 適切な型に変換するのはシミュレータ側コードで行う.

3.5 パラメータデータ構造の破棄

パラメータを取得した後, パラメータデータが不要になれば, パラメータパースクラスのインスタンスを破棄せずに, 内部のパラメータデータ構造を破棄することができる.

3.6 パラメータファイルへの書き出し

クラスインスタンス内のパラメータデータ構造を指定したファイルにパラメータファイル形式で書き出すことができる.

4 動作環境

本ライブラリは多くのプラットフォームで動作することを想定しているが, 最小限の変更及び設定により, 以下の環境で動作するように設計する.

1 Linux x86_64/i386
2 Mac OS X (10.6 Snow Leopard, 10.7 Lion)
3 Microsoft Windows7 64bit 版 32bit 版

5 スレッドセーフ

本ライブラリは複数スレッドからの同時アクセスによる不具合の発生を防ぐため, データベースのインスタンス生成からパラメータファイルの読み込み終了まで及びパラメータデータ構造の破棄後にデータの取得をロックする機構を設けている.

5.1 MPI 対応版

本ライブラリには, MPI を用いた並列プログラムでの利用を想定した, MPI 対応版がある. MPI 対応版では, rank 数 0 のプロセスが指定されたファイルからパラメータの記述を読み込み, その他のプロセスに読み込んだ内容を分配する. 全てのプロセスが読み込んだ内容をパースし, プロセスごとにデータを保持する. その際, プロセスごとに非 MPI 版と同様のデータ取得のロックを行い, そのプロセスが管理する複数のスレッドからの同時アクセスによる不具合の発生を防ぐ機構を設けている.

6 アップデート情報

本文書のアップデート情報について記す.

Version 1.1 2012/10/9

- get_instance() に加え、通常の new() にも対応.
- 値の書き換え機能を追加.

Version 1.0 2012/6/16

- Version 1.0 リリース.

Version 0.9c 2012/5/28

- Version 0.9c リリース. MPI 対応版の記述を追加. リーフの記述を一部修正.

Version 0.9b 2012/5/2

- Version 0.9b リリース. 例のインデントを調整.

Version 0.9a 2012/4/28

- Version 0.9a リリース.