

第1章 OHyMoS.NETのインストール手順

1.1 OHyMoS.NETの実行環境

OHyMoS.NETの実行には、事前にMicrosoft社のホームページで提供されている.NetFrameWorkをインストールしておく必要がある。.NetFrameWorkのインストール手順はWindowsUpdateなどを利用して導入することができる。また、各種プログラム言語を用いて要素モデルを開発する場合には、言語に応じた環境が必要になる、3章にて詳しく解説する。

1.2 OHyMoS.NET のダウンロード

OHyMoS.NET は京都大学大学院工学研究科 水文・水資源研究室にて最新版をダウンロードすることができる。ダウンロード手順は以下の通り

京都大学大学院工学研究科 水文・水資源研究室 (<http://hywr.kuciv.kyoto-u.ac.jp/>) にアクセスする。トップページ右側より OHyMoS の項目の下にある OHyMoS.NET を選択する。



OHyMoS.NET のページのダウンロードの項目より、OHyMoS.NET Version ~ を選択し、対象をファイルに保存する。

保存した ohymosnet.zip を適当なフォルダに解凍する。(スクリーンショットでは c:/ohymos)

水文モデル構築のためのフレームワーク「OHyMoS.NET」

OHyMoS - OHyMoS.NET

OHyMoS.NETの紹介

水文モデルはいくつかの水文素過程モデルから成り立っている。
 現在では、その要素となるモデルをユーザがプログラミングでき、かつその要素モデルを組み合わせて全体のシミュレーションモデルを容易に構成することを支援するシステムOHyMoSが存在する。
 OHyMoSの仕様に關しては、[OHyMoSのページ](#)を参照のこと。
 OHyMoS.NETでは、これらの仕様を再現しつつ1要素モデルの作成に複数のプログラミング言語を使用できる、
 2要素モデルをシステムと分類して作成・管理・提供できる、
 3データベースに接続できる
 といった点を実現した、水文モデル構築のためのフレームワークである。

現在のところGUIもたず、CUIで動作する。
 動作環境として.NET Frameworkを必要とし、開発言語はC#を使用している。

動作環境

Microsoft社より提供されている、.NET Frameworkをインストールしておく必要があります。
 データベースなどの機能を用いるためには別途個別のソフトが必要となります。

ダウンロード

- [OHyMoS.NET Version 1.00](#)
- [OHyMoS.NET APIDoc](#)

導入方法

以下のページを参考にOHyMoS.NETをインストールしてください。

免責事項

このソフトウェアの使用に
 いかなる問題が生じた場合
 バージョンアップや不具合
 この文書の内容および、
 責任を負いません。

関連論文、URL

- 鈴木俊朗：流出系の構造的モデリングシステムの開発, 京都大学大学院工学研究科土木工学専攻修士論文, 1994.
- 高植琢馬・椎葉充晴・市川温：構造的モデリングシステムを用いた流出シミュレーション, 水工学論文集, 第39巻, 1995.
- 佐藤芳洋：流出系の構造的モデリングシステムOHyMoS.Jの開発とその応用, 京都大学大学院工学研究科土木システム工学専攻修士論文, 2004.
- .NET Framework : Microsoft .NET Framework ホーム <http://msdn.microsoft.com/ja-jp/netframework/> 2008.

1.3 OHyMoS.NET の計算テスト

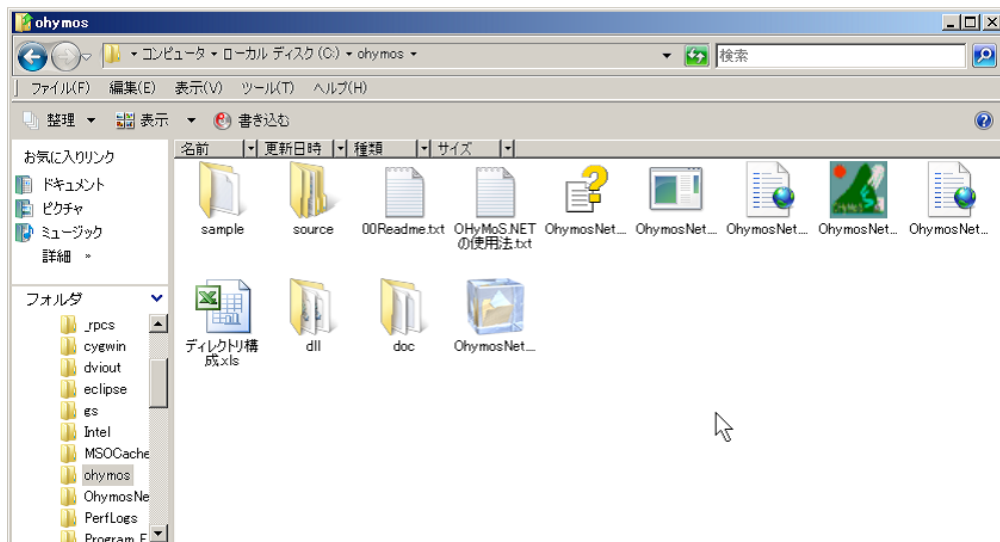
ダウンロードが完了したのち、コマンドプロンプトなどを用いて、適切に設置されたか計算テストを行う

cygwin 環境がインストールされている場合、cygwin を用いてもよい。

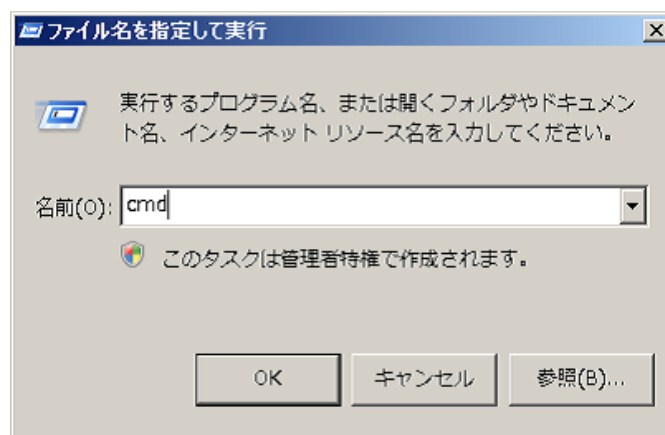
Windows 環境であれば、スタートメニュー プログラム アクセサリからコマンド プロンプトを起動する

SS:スタートメニューからコマンドプロンプトを選択する SS

「ファイル名を指定して実行」から cmd と入力することでコマンド プロンプトを起動しても



よい。

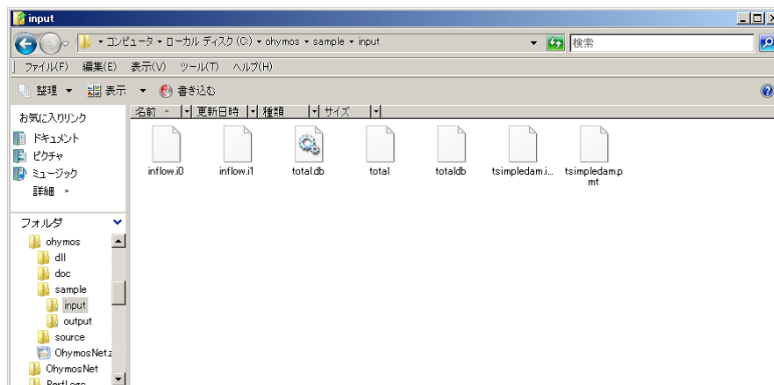


計算テストには、付属されているサンプルファイルを使用する。

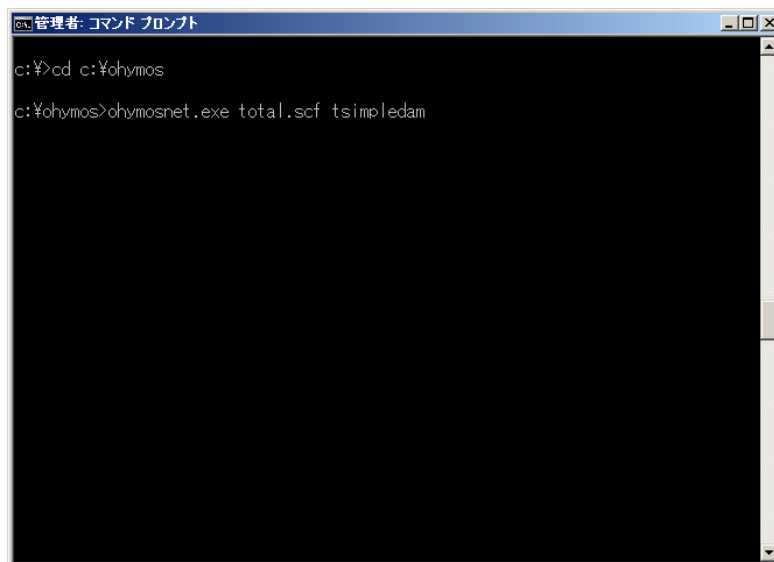
sample フォルダの input フォルダの中に計算に用いるデータがあるので、これらを ohymosnet.exe があるフォルダに移動する。

コマンドプロンプトによるテストの様子を示す。

まずはカレントディレクトリを OHyMoS.NET を解凍した場所に移す。(サンプルの場合、`cd c:\ohymos` とする)



入力【cd c:\ohymos】

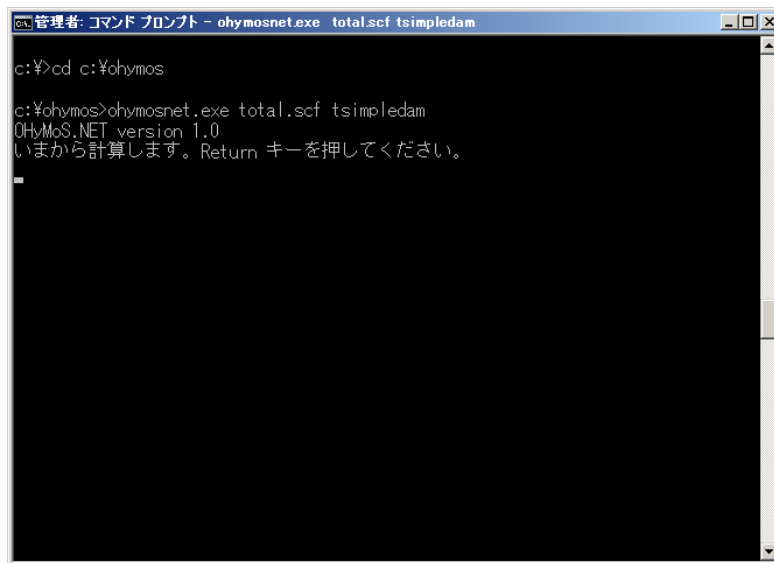


OHyMoS.NET に起動オプションとして構造定義ファイル名とパラメータ・初期状態量ファイル名を与え、実行する。

入力【ohymosnet.exe total.scf tsimpledam】

Return キーを押すことを要求されるので続行する。

実行時にエラーが出る場合は sample の input フォルダからすべてのファイルを移動したかを確認する。



```
管理: コマンド プロンプト - ohymosnet.exe total.scf tsimpledam
c:¥>cd c:¥ohymos
c:¥ohymos>ohymosnet.exe total.scf tsimpledam
OHyMoS.NET version 1.0
いまから計算します。Return キーを押してください。
```

フォルダ内に以下のファイルがあればよい

・inflow.i0 inflow.i1 total.scf tsimpledam.ist tsimpledam.pmt

実行すると計算の方式を尋ねられる、1 の GO_BEYOND_TERMINAL_TIME を選択し、続行する。

入力【1】

計算開始時刻に関する情報が表示され、時刻の指定を要求される。

ここでは計算開始時刻を3つの数字 (year,sec,frac) で指定する。

サンプルでは入力情報に従い、2000 50000 0 を指定する。

入力【2000 50000 0】

中間時刻の指定を要求される。

ここでは中間時刻を2つの数字 (sec,frac) で指定する。

```
管理: コマンド プロンプト - ohymosnet.exe total.scf tsimpledam
c:\>cd c:\%ohymos
c:\%ohymos>ohymosnet.exe total.scf tsimpledam
OHyMoS.NET version 1.0
いまから計算します。Return キーを押してください。

Wait. Now, I am working.
# Key in simulation option.
# TimeControlOption
#   current selection: GO_BEYOND_TERMINAL_TIME
#   select 1 or 2
#     1. GO_BEYOND_TERMINAL_TIME (Default)
#     2. DO_NOT_GO_BEYOND_TERMINAL_TIME

>1
```

```
管理: コマンド プロンプト - ohymosnet.exe total.scf tsimpledam
c:\>cd c:\%ohymos
c:\%ohymos>ohymosnet.exe total.scf tsimpledam
OHyMoS.NET version 1.0
いまから計算します。Return キーを押してください。

Wait. Now, I am working.
# Key in simulation option.
# TimeControlOption
#   current selection: GO_BEYOND_TERMINAL_TIME
#   select 1 or 2
#     1. GO_BEYOND_TERMINAL_TIME (Default)
#     2. DO_NOT_GO_BEYOND_TERMINAL_TIME

>1

File: inflow.i0 newest data time : 2000 50000 0
File: inflow.i1 newest data time : 2000 50000 0
# Key in time step or terminal time for 'total system'.
# current time (year sec frac) = 2000 0 0
# time step (day hour min sec frac) or terminal time (year sec frac)

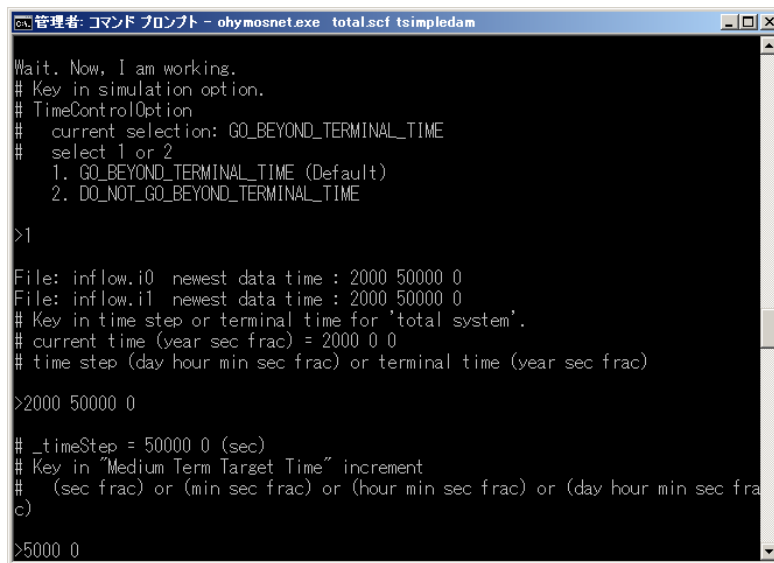
>2000 50000 0
```

サンプルでは 5000 0 を指定する。

入力【5000 0】

中間時刻の指定を要求される。

ここでは中間時刻を 2 つの数字 (sec,frac) で指定する。



```
管理: コマンド プロンプト - ohymosnet.exe total.scf tsimpledam
Wait. Now, I am working.
# Key in simulation option.
# TimeControlOption
# current selection: GO_BEYOND_TERMINAL_TIME
# select 1 or 2
1. GO_BEYOND_TERMINAL_TIME (Default)
2. DO_NOT_GO_BEYOND_TERMINAL_TIME

>1

File: inflow.i0 newest data time : 2000 50000 0
File: inflow.i1 newest data time : 2000 50000 0
# Key in time step or terminal time for 'total system'.
# current time (year sec frac) = 2000 0 0
# time step (day hour min sec frac) or terminal time (year sec frac)

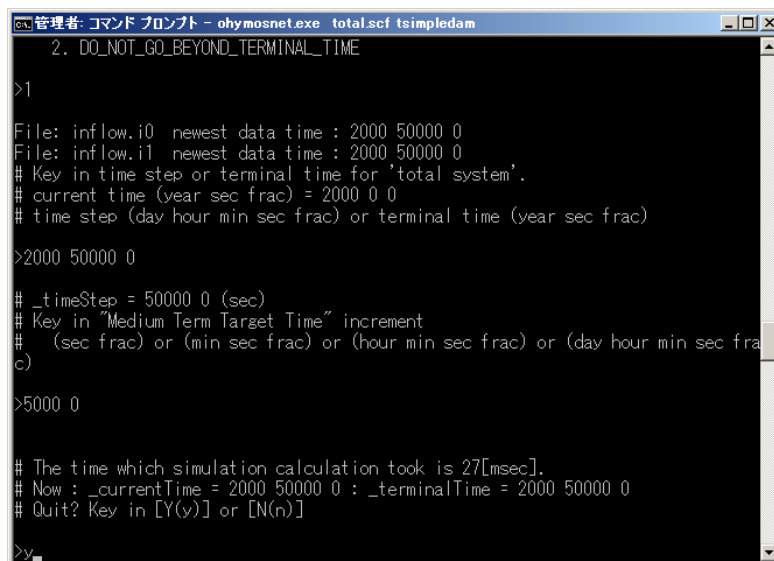
>2000 50000 0

# _timeStep = 50000 0 (sec)
# Key in "Medium Term Target Time" increment
# (sec frac) or (min sec frac) or (hour min sec frac) or (day hour min sec frac)

>5000 0
```

サンプルでは 5000 0 を指定する。

入力【5000 0】



```
管理: コマンド プロンプト - ohymosnet.exe total.scf tsimpledam
2. DO_NOT_GO_BEYOND_TERMINAL_TIME

>1

File: inflow.i0 newest data time : 2000 50000 0
File: inflow.i1 newest data time : 2000 50000 0
# Key in time step or terminal time for 'total system'.
# current time (year sec frac) = 2000 0 0
# time step (day hour min sec frac) or terminal time (year sec frac)

>2000 50000 0

# _timeStep = 50000 0 (sec)
# Key in "Medium Term Target Time" increment
# (sec frac) or (min sec frac) or (hour min sec frac) or (day hour min sec frac)

>5000 0

# The time which simulation calculation took is 27[msec].
# Now : _currentTime = 2000 50000 0 : _terminalTime = 2000 50000 0
# Quit? Key in [Y(y)] or [N(n)]

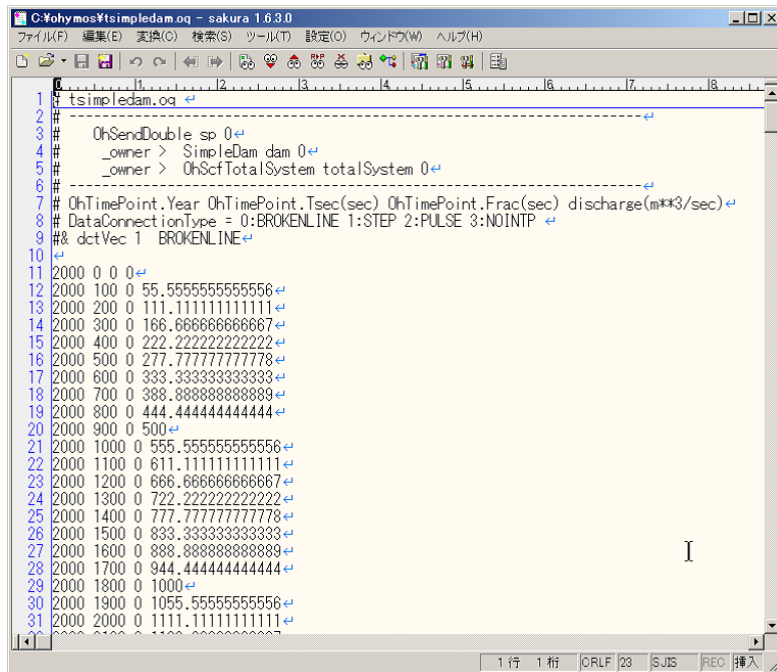
>y
```

計算にかかった時間などのメッセージが出れば計算完了である。

計算結果が出力され、計算をさらに終了するかを問われる。

計算を終了する場合 Y を指定する。

入力【y】



```
C:\ohyomos\tsimpledam.oq - sakura 1.6.3.0
1 # tsimpledam.oq
2 #
3 # OhSendDouble sp 0
4 # _owner > SimpleDam dam 0
5 # _owner > OhScfTotalSystem totalSystem 0
6 #
7 # OhTimePoint.Year OhTimePoint.Tsec(sec) OhTimePoint.Frac(sec) discharge(m**3/sec)
8 # DataConnectionType = 0:BROKENLINE 1:STEP 2:PULSE 3:NOINTP
9 #& dctVec 1 BROKENLINE
10 #
11 2000 0 0 0
12 2000 100 0 55.5555555555556
13 2000 200 0 111.111111111111
14 2000 300 0 166.666666666667
15 2000 400 0 222.222222222222
16 2000 500 0 277.777777777778
17 2000 600 0 333.333333333333
18 2000 700 0 388.888888888889
19 2000 800 0 444.444444444444
20 2000 900 0 500
21 2000 1000 0 555.555555555556
22 2000 1100 0 611.111111111111
23 2000 1200 0 666.666666666667
24 2000 1300 0 722.222222222222
25 2000 1400 0 777.777777777778
26 2000 1500 0 833.333333333333
27 2000 1600 0 888.888888888889
28 2000 1700 0 944.444444444444
29 2000 1800 0 1000
30 2000 1900 0 1055.55555555556
31 2000 2000 0 1111.111111111111
32 2000 2100 0 1166.66666666667
```

出力ファイルとして

tsimpledam.oq (計算出力結果 1)

tsimpledam.os (計算出力結果 2)

tsimpledam.tsd (計算の中間状態を保存したファイル)

tsimpledam.tst (計算の中間時間を保存したファイル)

がフォルダ内にできていることを確認する。

ファイルができていれば計算テストは完了である。

第2章 OHyMoS.NETの要素モデル作成

2.1 要素モデルを作る前に

OHyMoS.NET に新たな要素モデルを追加するには、dll(ダイナミックリンクライブラリ)を作成し、それを追加する。OHyMoS をインストールしたディレクトリに新たな要素モデルの dll を入れる場所を事前に作成しておくのが望ましい。

その後、ユーザ環境変数に「OHYMOS_LIB」という環境変数を作成する。先ほど自分が作成した要素モデルを格納しておきたいディレクトリのパスを OHYMOS_LIB に設定する。このときディレクトリは複数でも構わない。環境変数に設定するときは間をセミコロン (;) で区切ることにする

以後、設定したディレクトリに要素モデル (dll ファイル) を格納することで、使用できる要素モデルが増えていく。

2.2 要素モデルを作成する (C # 編)

要素モデルを作成するには、OHyMoS.NET の規格に従い、.NET 対応言語でソースコードを書き、OHyMoS.NET に同梱されている ([dll] ディレクトリ内) OhymosNet.dll を参照してコンパイルを行う。

コンパイルに用いるソフトには Visual Studio 2005 Express Edition などがある

ここではまず、.NET 対応言語である、C # を用いて要素モデルの作成を行う。サンプルとして内部的な計算を行わない簡易なモデルを用意した。そちらを参照しつつ話を進めていく。

```
using System;
using System.IO;
using OhymosNet;

namespace OhymosNet
{
    public class SampleCSharp:OhElement
    {
        /* 端子モデルを宣言する */
        internal OhReceiveDouble _rP; //受信端子 rp 用
        internal OhSendDouble _sP0; //送信端子 sp 0 用
        internal OhSendDouble _sP1; //送信端子 sp 1 用
        internal OhTime _dt; //計算ステップ時間の管理用
    }
}
```

```
internal double _rpdata;          //データ保持用の状態量

//ScfTotalSystem から呼び出される初期化関数
public override void init(String aObjName, int aObjNum, OhStreamReader aSr)
{
    init(aObjName, aObjNum, null);
}

//実際に初期化に用いられる
public virtual void init(String aObjName, int aObjNum, String aDataStr)
{
    _rP = new OhReceiveDouble(); //受信端子 rp の宣言と初期化を行う
    _rP.init("rp", 0);

    base.init(aObjName, aObjNum, 0, aDataStr);
}

//デフォルトコンストラクタ
public SampleCSharp():base()
{
    _rP = null;
    _sP0 = new OhSendDouble("sp", 0); //送信端子 sp 0 の宣言と初期化を行う
    _sP1 = new OhSendDouble("sp", 1); //送信端子 sp 1 の宣言と初期化を行う
    _className = "SampleCSharp";      //クラスの名前を設定する
}

//パラメータを伴うコンストラクタ
public SampleCSharp(String aObjName, int aObjNum, String aDataStr):base()
{
    _rP = null;
    _sP0 = new OhSendDouble("sp", 0);
    _sP1 = new OhSendDouble("sp", 1);
    _className = "SampleCSharp";
    init(aObjName, aObjNum, aDataStr); //こちらでは初期化も行う
}

//設定した受信端子を全体系に登録する
public override void registerReceivePorts()
{
    register((OhObject)_rP); //登録には register(OhObject objectname) 関数を用いる
}

//設定した送信端子を全体系に登録する
public override void registerSendPorts()
{
    register((OhObject)_sP0); //登録には register(OhObject objectname) 関数を用いる
    register((OhObject)_sP1); //登録には register(OhObject objectname) 関数を用いる
}
```

```

//パラメータファイル (*.pmt) からパラメータを呼び出し設定する
public override void setParameter(OhStreamReader aSw)
{
    double dt = 0.0; //計算ステップ dt (frac)
    try
    {
        dt = Double.Parse(aSw.readWord()); //パラメータファイルから一単語を呼び出し
        Double 型に変換する
        _dt = new OhTime(dt); //OhTime 型として再定義し、_dt に保存する
    }
    catch (IOException e) //readWord() がエラーを投げた場合
    {
        throw new OhError(oPrintString("\t"));
    }
    catch (FormatException e) //Double.Parse() がエラーを投げた場合
    {
        throw new OhError(oPrintString("\t"));
    }

    if (_dt.toDouble() < OhTime.NEGLIGIBLY_SMALL_TIME) //OhTime の用意している計算
    ステップより短い場合はエラーを投げる
    {
        throw new OhError("_dt should be greater than or equal to " + OhTime.NEGLIGIBLY_SMALL_TIME
            + " but it is " + _dt);
    }
}

//途中の計算時間を保存する
public virtual void saveParameter(StreamWriter aSw)
{
    aSw.Write(_dt + NEWLINE);
    aSw.Write(NEWLINE);
}

//初期状態量ファイル (*.ist) から初期状態量を呼び出し設定する
public override void setInitialState(OhStreamReader aSr)
{
    int year = 0; //年 (西暦)
    long tsec = 0L; //秒
    double frac = 0.0; //ミリ秒

    try
    {
        //要素モデルの計算開始時間を読み込む たとえば [ 2008 20000 0 ] であれば 2008 年
        1 月 1 日より 20000 秒経過した時間を指す
        aSr.skipWhites(); //空白部分を読み飛ばす関数
        year = Int32.Parse(aSr.readWord());
        tsec = Int64.Parse(aSr.readWord());
    }
}

```

```

        frac = Double.Parse(aSr.readWord());
    }
    catch (IOException e) //readWord() がエラーを投げた場合
    {
        throw new OhError("Error in setInitialState()." + oPrintString("\t"));
    }
    catch (FormatException e) // Parse() がエラーを投げた場合
    {
        throw new OhError("Error in setInitialState().", oPrintString("\t"));
    }
}

    _currentTime = new OhTimePoint(year, tsec, frac); //読み込んだ数字を OhTimePoint
型として _currentTime に格納する
}

//途中の計算状態を保存する
public override void saveTerminalState(StreamWriter aSw)
{
    aSw.Write(_currentTime + " # _currentTime");
    aSw.Write(NEWLINE);
    aSw.Write(_rpdata + " # _rpdata (dat)");
    aSw.Write(NEWLINE);
}

//シミュレーション開始時の初期出力を設定する
public override bool initialOutput()
{
    //受信端子側から情報を取り出せない場合は計算を行わない
    //ReceivePort.canYouGetData(_currentTime) は _currentTime の情報が受信端子から読
み込めるかをチェックする
    if (_rP.canYouGetData(_currentTime) == false)
    {
        return false;
    }

    //計算ができる場合は送信端子にデータを書き出す
    //CommentString は出力時に書式として書き込まれる
    //sendData(_currentTime,0) はここでは送信端子に、時刻: _currentTime のデータとし
て 0 を書き込んでいる
    _sP0.CommentString = "OhTimePoint.Year OhTimePoint.Tsec(sec) " + "OhTimePoint.Frac(sec) data_1";
    _sP0.sendData(_currentTime, 0);

    _sP1.CommentString = "OhTimePoint.Year OhTimePoint.Tsec(sec) " + "OhTimePoint.Frac(sec) data_2";
    _sP1.sendData(_currentTime, 0);

    return true; //計算が成功したら true を返す
}

//次の計算の時間ステップを計算して返す、ここでは計算の必要がないため、保存されている _dt

```

を返す

```
public override OhTime calculateTimeStep()
{
    return _dt;
}
```

//要素モデルが計算可能な状態にあるかを返す、ここでは継承してきた OhElement の canYouCalculate0()

を用いる

```
public override bool canYouCalculate()
{
    return canYouCalculate0();
}
```

//要素モデルの計算の一連の流れを行う、ここでは継承してきた OhElement の work0() を用いる

```
public override bool work()
{
    return work0();
}
```

//要素モデルに計算を行わせる。受信端子からデータを取得し、それらをもちいて1ステップ分の計算を行い、送信端子にデータを書き出す

//モデルの根幹をなす部分であるため、計算モデルの実装にはここを改造すればよい

```
public override bool calculate()
{
```

OhTimePoint nextTime = OhTimePoint.add(_currentTime, _timeStep); //現在時間と時間ステップから次の計算時刻を割り出す

```
    //予定される次の計算時刻に受信端子のデータがあるかを確認する
    //データがない場合は計算が不能であるため、false を返す
    if (_rP.canYouGetData(nextTime) == false)
    {
        return false;
    }
```

```
    //受信端子から次の計算時刻のデータを受け取る
    _rpdata = _rP.getData(nextTime);
```

/* モデルの1ステップ分の計算をここに記述する */

//送信端子にデータを書き出す。ここでは受信してきたデータをそのまま二つの送信端子に渡

している

```
_sP0.sendData(nextTime, _rpdata);
_sP1.sendData(nextTime, _rpdata);
```

```
return true; //計算が成功したら true を返す
```

```
}
```

//要素モデルの基本情報(ここでは要素名)を返す

```
public override String basicInfo()
```

```
    {  
        return "SampleCSharp";  
    }  
}
```

サンプルは以上 OHyMoS.NET は要素モデルを作成した段階ではまだシミュレーションは実行できない。次項にて、シミュレーションの実行に必要な「構造定義ファイル」と「パラメータ・初期状態量ファイル」について解説する。

第3章 シミュレーション実行まで

この章では前節で作成した SampleCSharp を、既存のサンプルに組み込んだシミュレーションを行う

3.1 構造定義ファイルの作成

構造定義ファイルの詳しい解説は～節を参照のこと。
サンプル

```
# total.scf
#
# -----
# 1. Numbers part
# -----

2 # number of input ports
1 # number of components
0 # number of iterator elements
0 # number of iterator sets (number_of_iterartor_sets)
2 # number of output ports

# -----
# 2. Memory allocations part
# -----

[InputPorts]
# id objname classname

0 inflow.i0 OhInputFileDouble
1 inflow.i1 OhInputFileDouble

[Components]
# id objname objnum classname basin_structure_file nin ...

0 dam 0 SimpleDam 2 # 2 is the number of input ports

[Iterators]
# id objname objnum classname
```



```

[OutputPorts]
# id objname linelength classname

0 tsimpledam.oq 80 OhOutputFileDouble
1 tsimpledam.os 80 OhOutputFileDouble

# -----
# 3. Register and connect part
# -----

[RegisterIterators]

[ConnectPorts]

r 0 c 0 rp 0 # connect input port 0 to rp 0 of component 0 (dam)
r 1 c 0 rp 1 # connect input port 1 to rp 1 of component 0 (dam)
c 0 sp 0 s 0 # connect sp 0 of component 0 (dam) to output port 0
c 0 sp 1 s 1 # connect sp 1 of component 0 (dam) to output port 1
z          # end mark

[ConnectComponents]
z # end mark

  変更後

# total.scf
#
# -----
# 1. Numbers part
# -----

2 # number of input ports
2 # number of components
0 # number of iterator elements
0 # number of iterator sets (number_of_iterartor_sets)
4 # number of output ports

# -----
# 2. Memory allocations part
# -----

[InputPorts]
# id objname classname

0 inflow.i0 OhInputFileDouble
1 inflow.i1 OhInputFileDouble

[Components]
# id objname objnum classname basin_structure_file nin ...

```

```

0 sample 0 SampleCSharp # 2 is the number of input ports
1 sample 1 SampleCSharp # 2 is the number of input ports

[Iterators]
# id objname objnum classname

[OutputPorts]
# id objname linelength classname

0 tsimpledam.oq 80 OhOutputFileDouble
1 tsimpledam.os 80 OhOutputFileDouble
2 sample1.txt 80 OhOutputFileDouble
3 sample2.txt 80 OhOutputFileDouble

# -----
# 3. Register and connect part
# -----

[RegisterIterators]

[ConnectPorts]

r 0 c 0 rp 0 # connect input port 0 to rp 0 of component 0 (dam)
c 0 sp 0 s 0 # connect sp 0 of component 0 (dam) to output port 0
c 0 sp 1 s 1 # connect sp 1 of component 0 (dam) to output port 1
c 0 sp 0 c 1 rp 0 # connect input port 0 to rp 0 of component 0 (dam)
c 1 sp 0 s 2 # connect sp 0 of component 0 (dam) to output port 0
c 1 sp 1 s 3 # connect sp 1 of component 0 (dam) to output port 1
z          # end mark

[ConnectComponents]
z # end mark

```

3.2 パラメータファイル (*.pmt) の作成

サンプル

```

# tsimpledam.pmt
#
# parameter file
#
#TestSimpleDam totalSystem 0
OhScfTotalSystem totalSystem 0
{
  SimpleDam dam 0
  {
    2000 0.5 100 # _qPass (m3/sec), _ratio (-), _dt (sec)
    1.0 1.0     # _cRp[0], _cRp[1]
  }
}

```

```
}  
}
```

変更後

```
# tsimpledam.pmt  
#  
# parameter file  
#  
#TestSimpleDam totalSystem 0  
OhScfTotalSystem totalSystem 0  
{  
SampleCSharp sample 0  
{  
100 # current_time  
}  
SampleCSharp sample 1  
{  
100 # current_time  
}  
}
```

3.3 初期状態量ファイル (*.ist) の作成

サンプル

```
# tsimpledam.ist  
#  
#TestSimpleDam totalSystem 0  
OhScfTotalSystem totalSystem 0  
2000 0 0.0  
{  
SimpleDam dam 0  
{  
2000 0 0.0 # current_time  
0. # storage (m^3)  
}  
}
```

変更後

```
# tsimpledam.ist  
#  
#TestSimpleDam totalSystem 0  
OhScfTotalSystem totalSystem 0  
2000 0 0.0  
{  
# SimpleDam dam 0  
# {
```

```
# 2000 0 0.0 # current_time
# 0. # storage (m^3)
# }
```

```
SampleCSharp sample 0
{
2000 0 0.0 # current_time
}
```

```
SampleCSharp sample 1
{
2000 0 0.0 # current_time
}
}
```