iRIC Developer's manual Documentation

リリース ***3.0.0***

iRIC organization

2017 年 **07** 月 **27** 日

目次

第 1 章

このマニュアルについて

このマニュアルでは、以下のユーザに必要な情報を解説します。

- iRIC 上で動作するソルバーの開発者
- iRIC 上で動作する格子生成プログラムの開発者

ソルバー開発者の方は、まずは 2 章を読んで、ソルバー開発の流れについて理解してください。その後、必要に応じて 5 章、6 章、7 章を参照してください。

格子生成プログラム開発者の方は、まずは 4 章を読んで格子生成プログラム開発の流れについて理解してください。その後、必要に応じて 5 章、6 章、7 章を参照してください。

第 2 章

ソルバーの開発手順

2.1 概要

ソルバーは、格子、計算条件などに基づいて河川シミュレーションを実行し、計算結果を出力するプログラムです。

iRIC 上で動作するソルバーを開発するには、表 2.1 に示すようなファイルを作成、配置する必要があります。

表 2.1 に示したファイルは iRIC インストール先の下の 「solvers」 フォルダの下に自分が開発するソルバー専用のフォルダを作成し、その下に配置します。

表 2.1　ソルバー関連ファイル一覧

ファイル名	説明
definition.xml	ソルバー定義ファイル。英語で記述する。
solver.exe (例)	ソルバーの実行モジュール。ファイル名はソルバー開発者が任意に選べる。
translation_ja_JP.ts など	ソルバー定義ファイルの辞書ファイル。
README	ソルバーの説明ファイル
LICENSE	ソルバーのライセンス情報ファイル

各ファイルの概要は以下の通りです。

2.1.1 definition.xml

ソルバーに関する以下の情報を定義するファイルです。

- 基本情報
- 計算条件
- 格子属性

iRIC はソルバー定義ファイルを読み込むことで、そのソルバーに必要な計算条件、格子を作成するためのインターフェースを提供し、そのソルバー用の計算データファイルを生成します。ソルバー定義ファイルは、すべて英語で記述します。

2.1.2 ソルバー

河川シミュレーションを実行するプログラムです。iRIC で作成した計算条件と格子を読みこんで計算を行い、結果を出力します。

計算条件、格子、結果の入出力には、iRIC が生成する計算データファイルを使用します。ただし、計算データファイルで入出力を行えないデータについては、任意の外部ファイルを入出力に使うこともできます。

FORTRAN, C 言語、C++ 言語のいずれかの言語で開発します。この章では、FORTRAN で開発する例を説明します。

2.1.3 translation_ja_JP.ts など

ソルバー定義ファイルで用いられている文字列のうち、ダイアログ上などに表示される文字列を翻訳して表示するための辞書ファイルです。日本語 (translation_ja_JP.ts)、韓国語 (translation_ka_KR.ts) など言語ごとに別ファイルとして作成します。

2.1.4 README

ソルバーに関する説明を記述するテキストファイルです。iRIC で新しいプロジェクトを開始する時にソルバーを選択する画面で、説明欄に表示されます。

2.1.5 LICENSE

ソルバーのライセンスについて記述するテキストファイルです。iRIC で 新しいプロジェクトを開始する時にソルバーを選択する画面で、ライセンス欄に表示されます。

iRIC、ソルバー、関連ファイルの関係を 図 2.1 に示します。

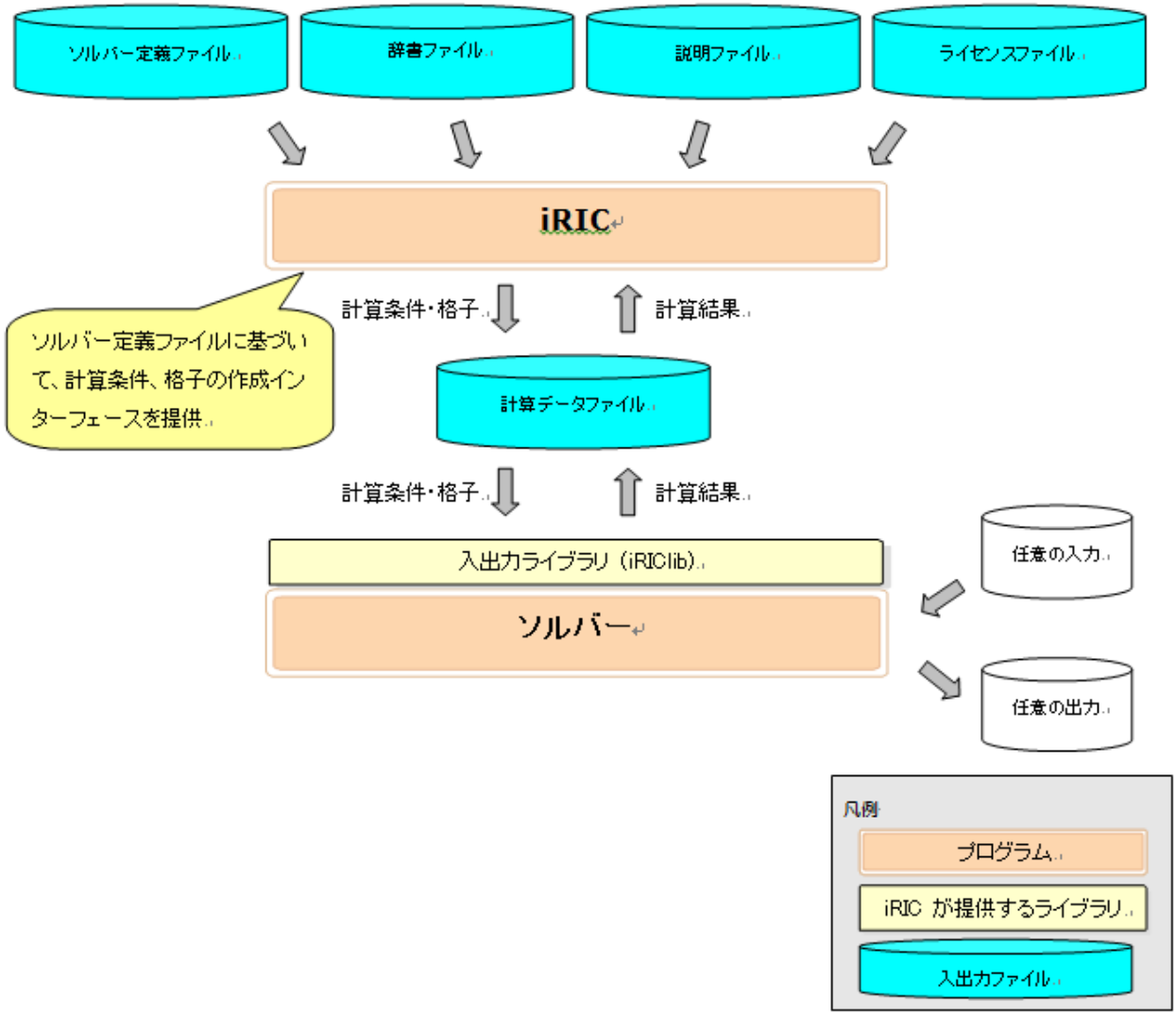

図 2.1　iRIC、ソルバー、関連ファイルの関係図

この章では、この節で説明した各ファイルを作成する手順を順番に説明します。

2.2 フォルダの作成

iRIC のインストールフォルダ の下にある 「solvers」フォルダの下に、開発するソルバーのための専用のフォルダを作成します。今回は、 「example」 というフォルダを作成します。

2.3 ソルバー定義ファイルの作成

ソルバー定義ファイルを作成します。

ソルバー定義ファイルは、ソルバーに関する 表 2.2 に示す情報を定義します。

表 2.2　ソルバー定義ファイルで定義する情報

項目	説明	必須
基本情報	ソルバーの名前、開発者、リリース日など	○
計算条件	ソルバーの実行に必要な計算条件	○
格子属性	計算格子の格子点もしくは格子セルに与える属性	○
境界条件	計算格子の格子点もしくは格子セルに与える境界条件	

ソルバー定義ファイルは、マークアップ言語の一種である XML 言語で記述します。XML 言語の基礎については 5.5 を参照してください。

この節では、ソルバー定義ファイルを、表 2.2 に示した順で作成していきます。

2.3.1 基本情報の作成

ソルバーの基本情報を作成します。 リスト 2.1 に示すようなファイルを作り、 フォルダの作成 (ページ 5) で作成した 「example」 フォルダの下に 「definition.xml」 の名前で保存します。

リスト 2.1　基本情報を記述したソルバー定義ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<SolverDefinition
  name="samplesolver"
  caption="Sample Solver 1.0"
  version="1.0"
  copyright="Example Company"
  release="2012.04.01"
  homepage="http://example.com/"
  executable="solver.exe"
  iterationtype="time"
  gridtype="structured2d"
>
  <CalculationCondition>
  </CalculationCondition>
  <GridRelatedCondition>
  </GridRelatedCondition>
</SolverDefinition>
```

この時点では、ソルバー定義ファイルの構造は 図 2.2 に示すようになっています。

要素	備考
SolverDefinition	基本情報を登録。
CalculationCondition	計算条件を定義。現在は空。
GridRelatedCondition	格子属性を定義。現在は空。

図 2.2　ソルバー定義ファイルの構造

正しくソルバー定義ファイルが作成できているか確認します。

iRIC を起動します。 図 2.3 に示すダイアログが表示されますので、「新しいプロジェクト」ボタンを押します。図 2.4 に示すダイアログが表示されますので、ソルバーのリストに 「Sample Solver」 があるか確認します。あったらそれをクリックし、右側に先ほど指定した属性が正しく表示されるか確認します。

なお、このダイアログでは、以下の属性については表示されません。

- name
- executable
- iterationtype
- gridtype

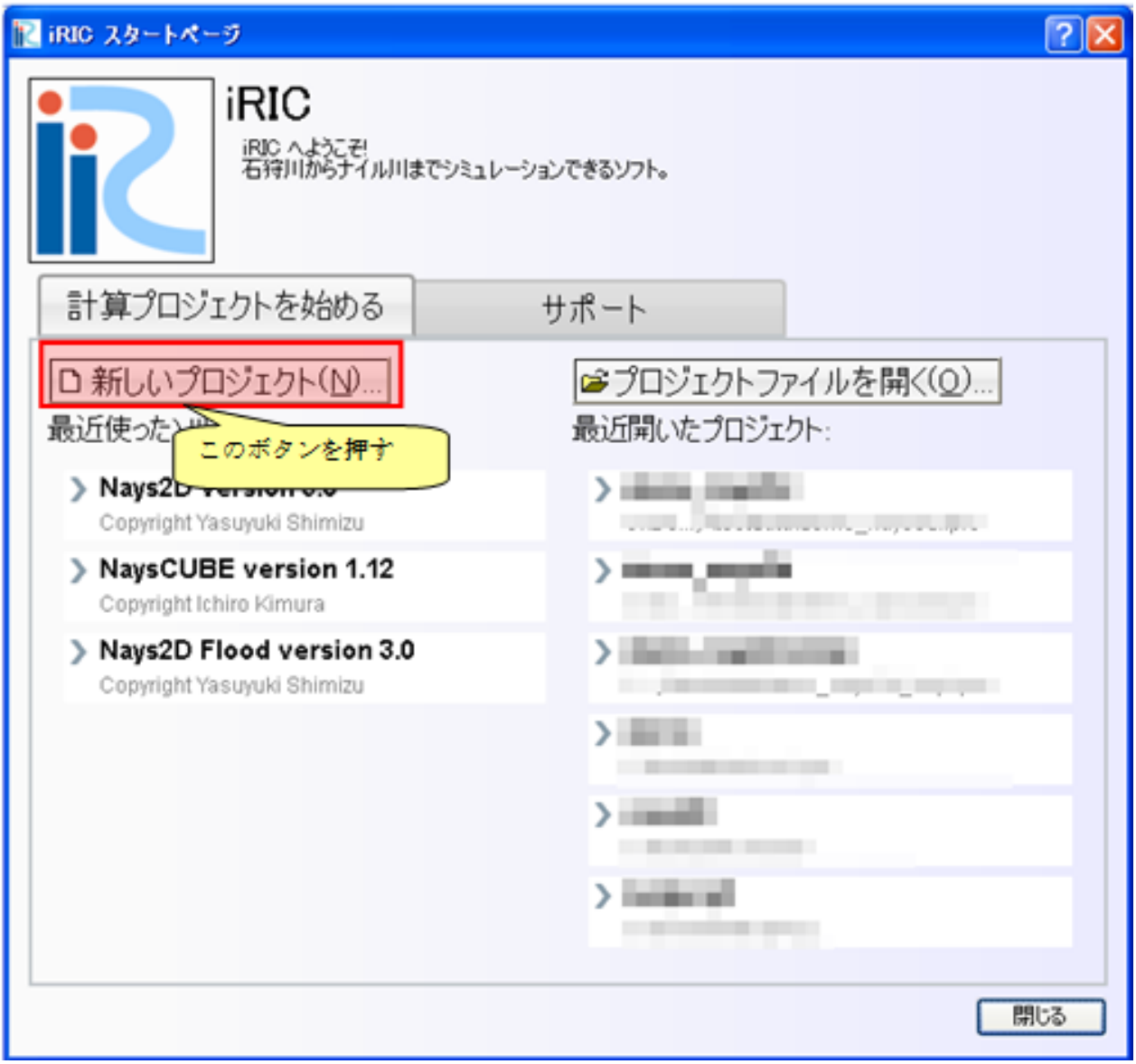

図 2.3　iRIC のスタートダイアログ

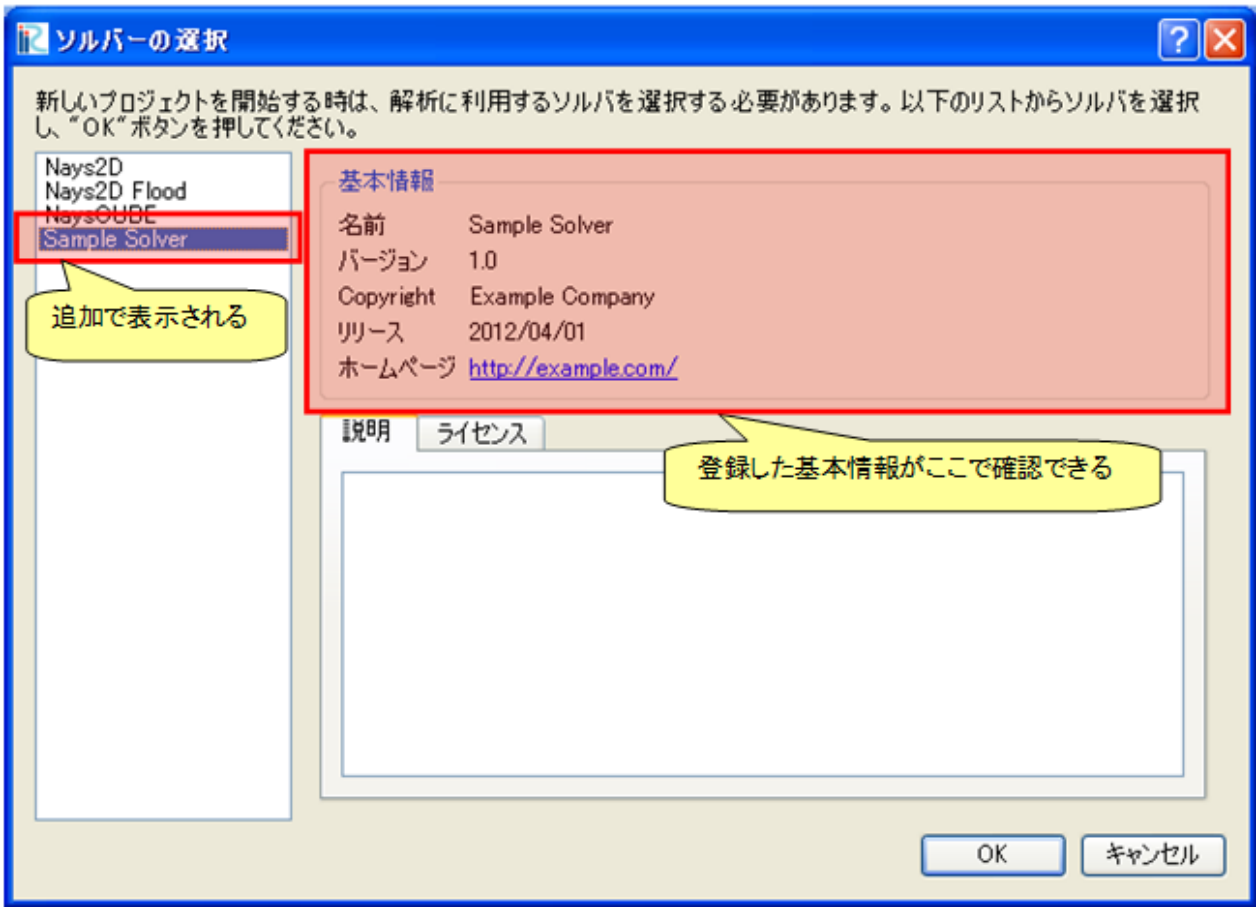

図 2.4　ソルバー選択ダイアログ

なお、ここで記述する name 属性と version 属性については、ソルバーのバージョンアップの際に気をつける必要があります。バージョンアップの際の注意点については 5.5 節を参照してください。

2.3.2 計算条件の定義

計算条件を定義します。計算条件は、ソルバー定義ファイルの CalculationCondition 要素で定義します。 基本情報の作成 (ページ 6) で作成したソルバー定義ファイルに追記し、 リスト 2.2 に示すようなファイルにし、保存します。追記した部分を強調して示しました。

リスト 2.2　計算条件を追記したソルバー定義ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<SolverDefinition
  name="samplesolver"
  caption="Sample Solver"
  version="1.0"
  copyright="Example Company"
  release="2012.04.01"
```

```
  homepage="http://example.com/"
  executable="solver.exe"
  iterationtype="time"
  gridtype="structured2d"
>
  <CalculationCondition>
    <Tab name="basic" caption="Basic Settings">
      <Item name="maxIteretions" caption="Maximum number of Iterations">
        <Definition valueType="integer" default="10">
        </Definition>
      </Item>
      <Item name="timeStep" caption="Time Step">
        <Definition valueType="real" default="0.1">
        </Definition>
      </Item>
    </Tab>
  </CalculationCondition>
  <GridRelatedCondition>
  </GridRelatedCondition>
</SolverDefinition>
```

この時点では、ソルバー定義ファイルの構造は 図 2.5 に示すようになっています。

要素	備考
SolverDefinition	基本情報を登録。
CalculationCondition	計算条件を定義。
Tab	計算条件のグループを定義。
Item	計算条件の名前を定義。
Definition	計算条件の属性を定義。
Item	計算条件の名前を定義。
Definition	計算条件の属性を定義。
GridRelatedCondition	格子属性を定義。現在は空。

図 2.5　ソルバー定義ファイルの構造

正しくソルバー定義ファイルが作成できているか確認します。

iRIC を起動します。図 2.3 に示すダイアログが表示されますので、「新しいプロジェクト」ボタンを押して、ソルバーのリストから 「Sample Solver」 をクリックし、 「OK」 ボタンを押します。図 2.6 に示すダイアログが表示されますが、 「OK」 ボタンを押して進みます。

図 2.6　警告ダイアログ 表示例

プリプロセッサが表示されますので、以下の操作を行います。

メニュー: –> 計算条件 (C) –> 設定 (S)

すると、 図 2.7 に示すダイアログが表示されます。 リスト 2.2 で追記した内容に従って表示されているのが分かります。

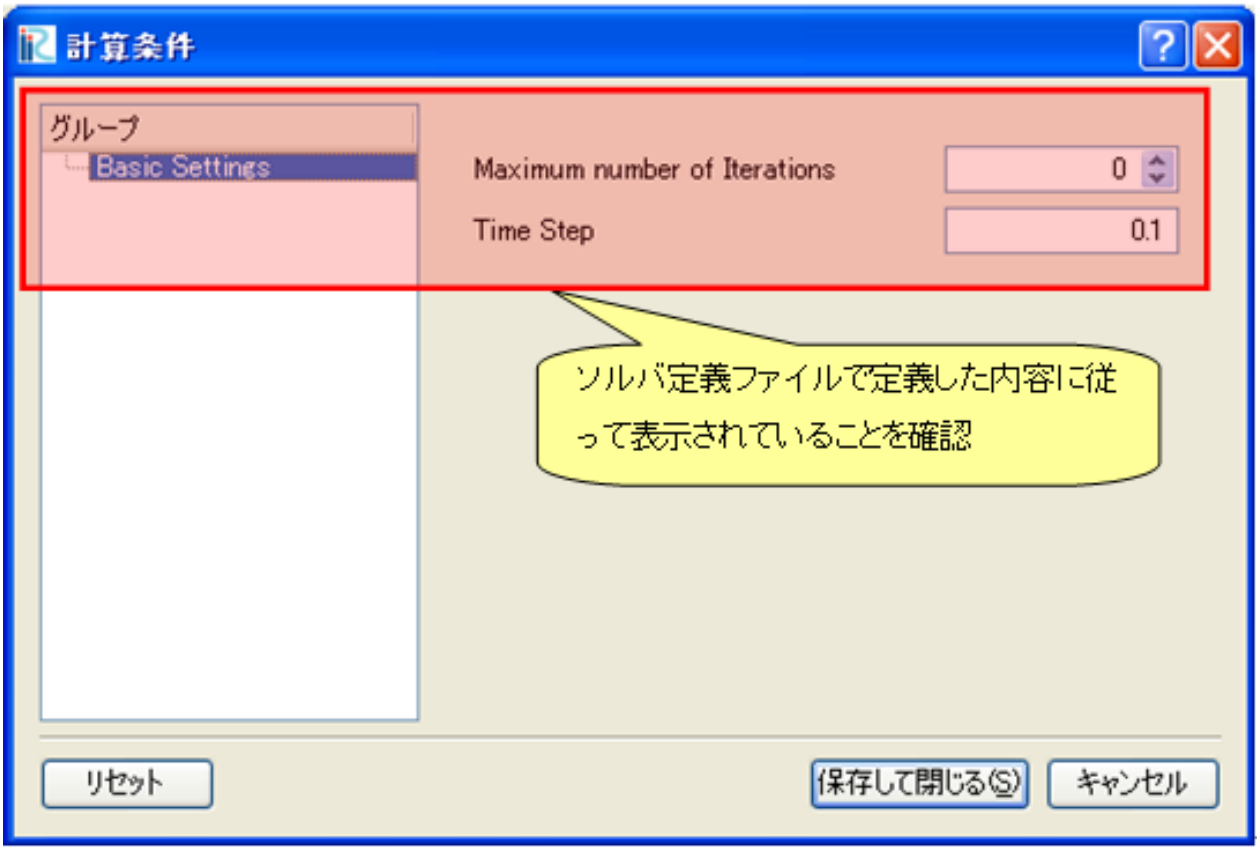

図 2.7　計算条件設定ダイアログ 表示例

グループを増やして、さらに計算条件を追加します。Basic Settings の Tab 要素 のすぐ下に、「Water Surface Elevation」 というグループを追加して保存します。追記したソルバー定義ファイルの抜粋を、 リスト 2.3 に示します。追記した部分を強調して示しました。

リスト 2.3　計算条件を追記したソルバー定義ファイルの例 (抜粋)

```
(前略)
    </Tab>
    <Tab name="surfaceElevation" caption="Water Surface Elevation">
      <Item name="surfaceType" caption="Type">
        <Definition valueType="integer" default="0">
          <Enumeration caption="Constant" value="0" />
          <Enumeration caption="Time Dependent" value="1" />
        </Definition>
      </Item>
      <Item name="constantSurface" caption="Constant Value">
        <Definition valueType="real" default="1">
          <Condition type="isEqual" target="surfaceType" value="0"/>
        </Definition>
      </Item>
      <Item name="variableSurface" caption="Time Dependent Value">
        <Definition valueType="functional">
          <Parameter valueType="real" caption="Time(s)"/>
          <Value valueType="real" caption="Elevation(m) "/>
          <Condition type="isEqual" target="surfaceType" value="1"/>
        </Definition>
      </Item>
    </Tab>
  </CalculationCondition>
  <GridRelatedCondition>
  </GridRelatedCondition>
</SolverDefinition>
```

この時点では、ソルバー定義ファイルの構造は図 2.8 に示すようになっています。

要素	備考
SolverDefinition	基本情報を登録。
CalculationCondition	計算条件を定義。
Tab	計算条件のグループを定義。(Basic Settings)
(略)	
Tab	計算条件のグループを定義。(Water Surface Elevation)
Item	計算条件の名前を定義。
Definition	計算条件の属性を定義。
Enumeration	計算条件に指定できる選択肢を定義。
Enumeration	計算条件に指定できる選択肢を定義。
Item	計算条件の名前を定義。
Definition	計算条件の属性を定義。
Condition	この計算条件が有効になる条件を定義
Item	計算条件の名前を定義。
Definition	計算条件の属性を定義。
Parameter	関数型の計算条件のパラメータを定義
Value	関数型の計算条件の値を定義
Condition	この計算条件が有効になる条件を定義
GridRelatedCondition	格子属性を定義。現在は空。

図 2.8　ソルバー定義ファイルの構造

正しくソルバー定義ファイルが作成できているか確認します。先ほどと同じ手順でダイアログを表示します。

「Water Surface Elevation」 というグループがリストに表示されているのが分かります。また、 「Constant Value」は、「Type」で「Constant」を選択している時のみ、「Time Dependent Value」は、「Type」で「Time Dependent」を選択している時のみ有効です。

ダイアログの表示例を 図 2.9 に示します。

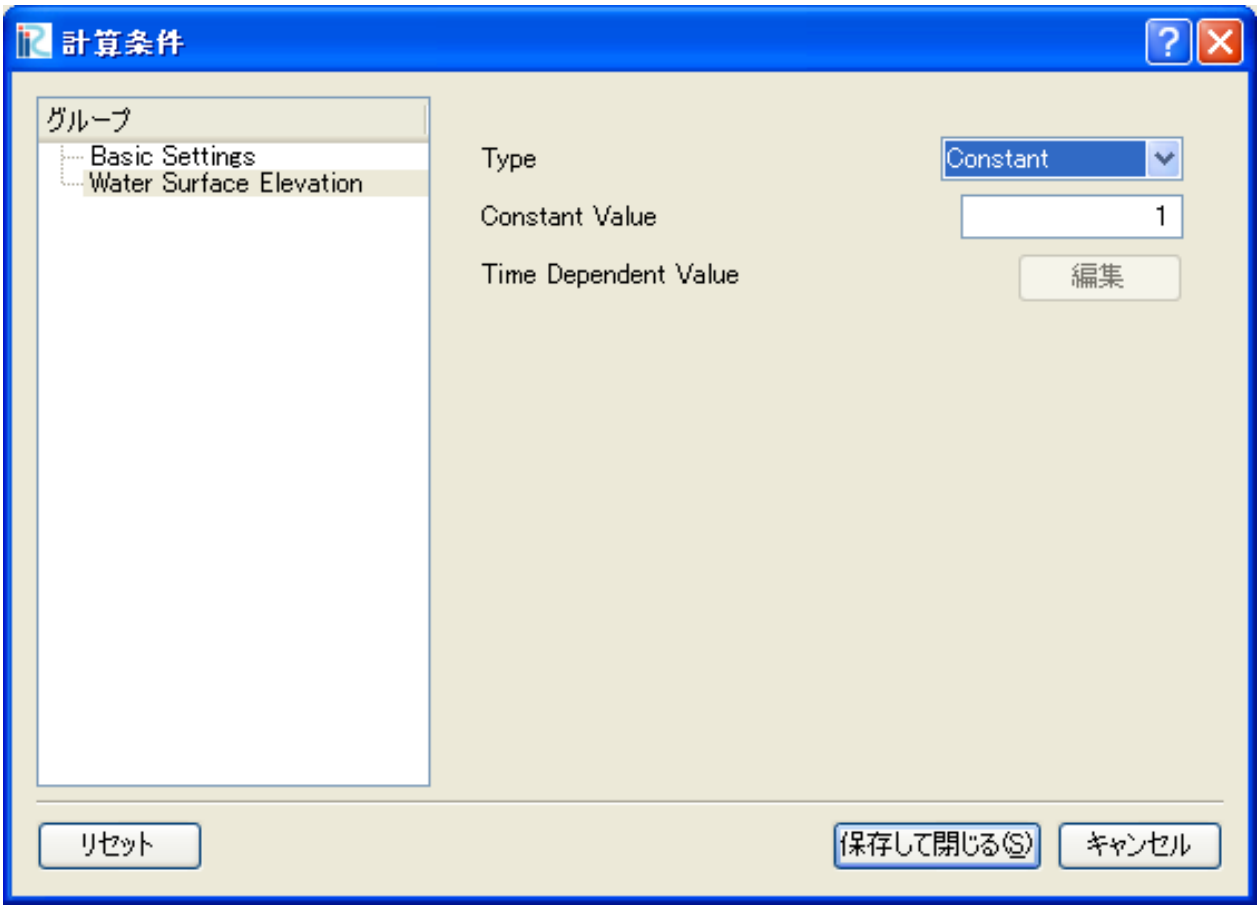

図 2.9　ソルバー定義ファイルの構造

計算条件の定義についてまとめると、以下の通りです。

- 計算条件のグループは Tab 要素で、計算条件は Item 要素で指定します。
- Definition 要素以下の構造は、計算条件の種類 (例: 整数、実数、整数からの選択、関数型) によって異なります。計算条件の種類ごとの記述方法とダイアログ上での表示については 5.3.1 を参照して下さい。
- 計算条件には、 Condition 要素で依存関係を定義できます。Condition 要素では、その計算条件が有効になる条件を指定します。 Condition 要素の定義方法の例は、5.3.2 を参照して下さい。
- この例では、計算条件のダイアログを単純なリスト形式で作成しましたが、グループボックスを使うなどしてダイアログのレイアウトをカスタマイズすることができます。ダイアログのレイアウトのカスタマイズ方法については 5.3.3 を参照して下さい。

2.3.3 格子属性の定義

格子属性を定義します。格子属性は、ソルバー定義ファイルの GridRelatedCondition 要素で定義します。 計算条件の定義 (ページ 9) で作成したソルバー定義ファイルに追記し、 GridRelatedCondition 要素にリスト 2.4 に示すよ

うに追記し、保存します。追記した部分を強調して示しました。

リスト 2.4　格子属性を追記したソルバー定義ファイルの例 (抜粋)

```
(前略)
  </CalculationCondition>
  <GridRelatedCondition>
    <Item name="Elevation" caption="Elevation">
      <Definition position="node" valueType="real" default="max" />
    </Item>
    <Item name="Obstacle" caption="Obstacle">
      <Definition position="cell" valueType="integer" default="0">
        <Enumeration value="0" caption="Normal cell" />
        <Enumeration value="1" caption="Obstacle" />
      </Definition>
    </Item>
    <Item name="Rain" caption="Rain">
      <Definition position="cell" valueType="real" default="0">
        <Dimension name="Time" caption="Time" valueType="real" />
      </Definition>
    </Item>
  </GridRelatedCondition>
</SolverDefinition>
```

正しくソルバー定義ファイルが作成できているか確認します。

iRIC を起動して、ソルバー 「Sample Solver」 の新しいプロジェクトを開始します。すると、 図 2.10 に示すような画面が表示されます。さらに、格子を作成したりインポートしたりすると、図 2.11 のようになります。

なお、格子の作成やインポートの方法が分からない場合、ユーザマニュアルを参照して下さい。

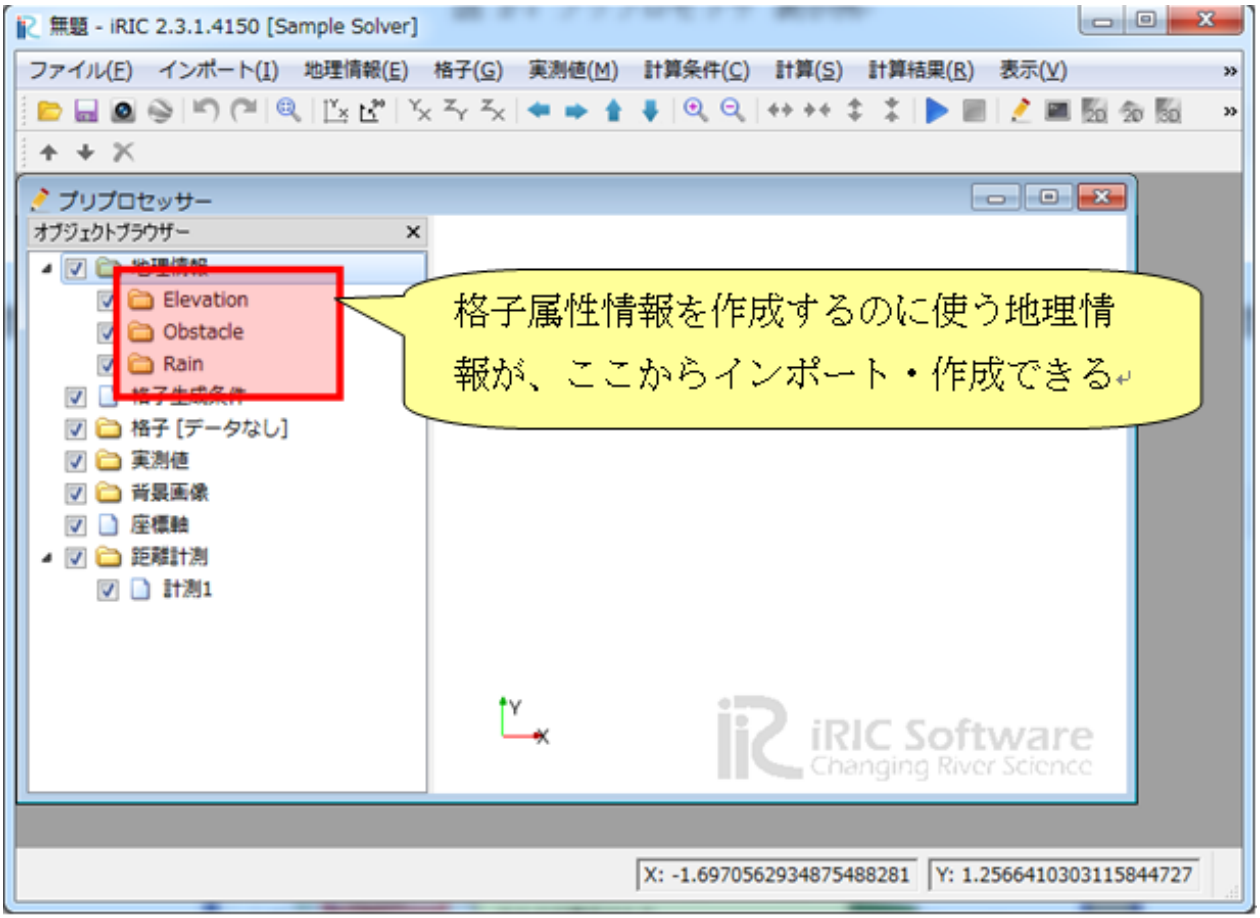

図 2.10　プリプロセッサ 表示例

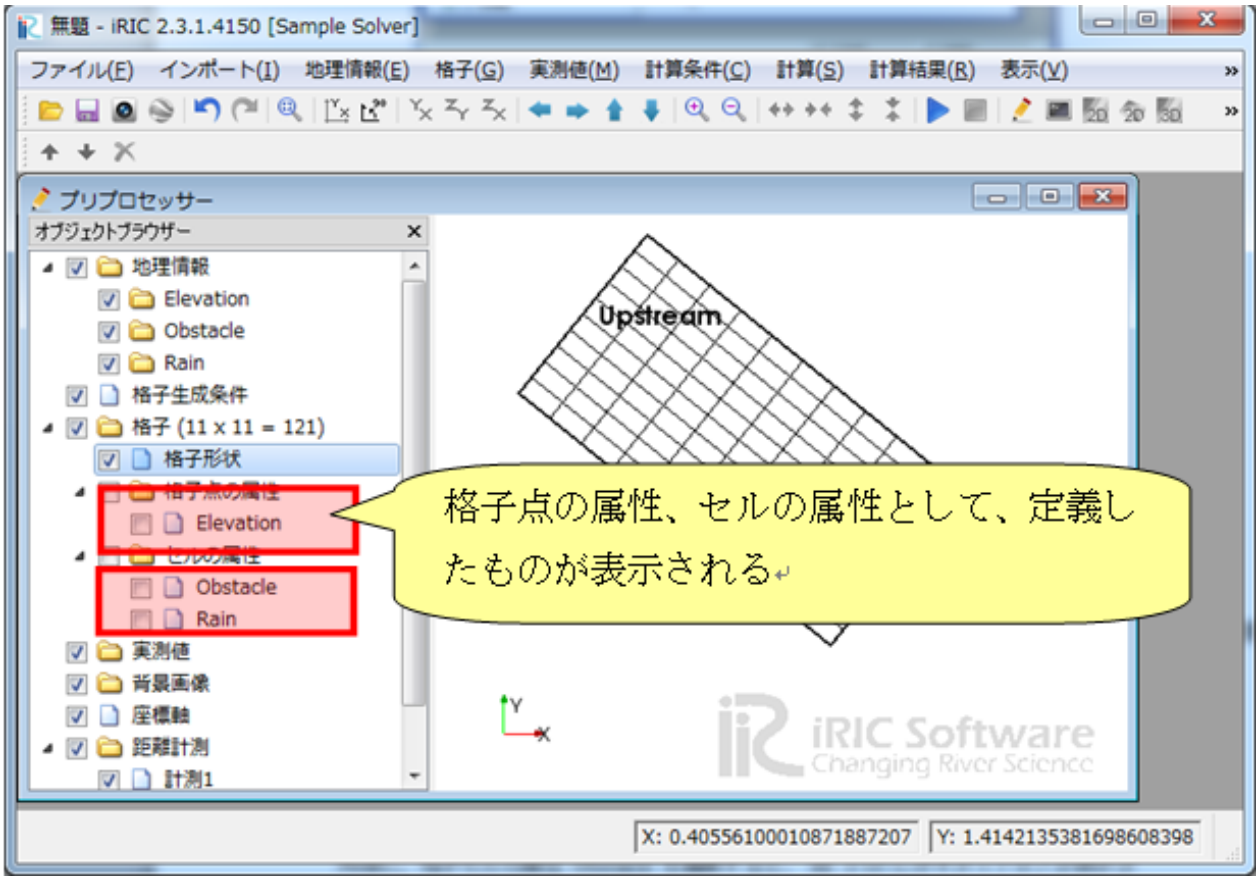

図 2.11　プリプロセッサ 表示例 (格子生成後)

以下の手順で格子点の属性 Elevation を編集すると、 図 2.12 に示すダイアログが表示され、実数の値を入力できることが確認できます。

- オブジェクトブラウザで、 「格子」 –> 「格子点の属性」 -> 「Elevation」 を選択します。
- 描画領域で、マウスクリックで格子点を選択します。
- 右クリックメニューを表示し、 「編集」 を選択します。

図 2.12　格子点の属性 「Elevation」 の編集ダイアログ

同様に、格子セルの属性 「Obstacle」 を編集すると、 図 2.13 に示すダイアログが表示され、 リスト 2.4 で指定した選択肢から値を選択できることが確認できます。

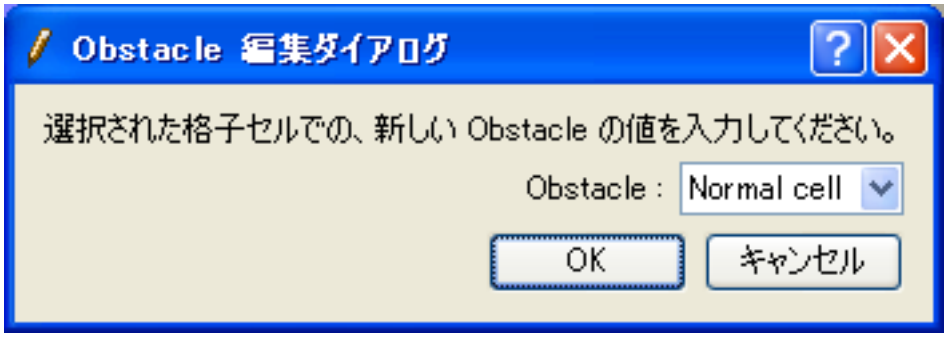

図 2.13　格子セルの属性 「Obstacle」 の編集ダイアログ

格子属性の定義についてまとめると、以下の通りです。

- 格子属性は、Item 要素で指定します。
- Item 要素以下の構造は計算条件の Item と基本的には同じですが、以下の違いがあります。
  - 属性を格子点で定義するか、セルで定義するかを position 属性で指定します。
  - 文字列、関数型、ファイル名、フォルダ名を指定することはできません。
  - 依存関係を指定することはできません。
  - Dimension 要素を用いて、次元を定義することができます。

格子属性については、iRIC では特別な名前が定義されており、特定の目的で使用される属性ではその名前を使用する必要があります。特別な格子属性の名前については 7.3.1 を参照してください。

2.3.4 境界条件の定義

境界条件を定義します。境界条件は、ソルバー定義ファイルの BoundaryCondition 要素で定義します。なお、境界条件の定義は必須ではありません。

格子属性の定義 (ページ 14) で作成したソルバー定義ファイルに追記し、BoundaryCondition 要素を リスト 2.5 に示すように追記し、保存します。追記した部分を強調して示しました。

リスト 2.5　境界条件を追記したソルバー定義ファイルの例 (抜粋)

```
(前略)
  </GridRelatedCondition>
  <BoundaryCondition name="inflow" caption="Inflow" position="node">
    <Item name="Type" caption="Type">
      <Definition valueType="integer" default="0" >
        <Enumeration value="0" caption="Constant" />
```

```
        <Enumeration value="1" caption="Variable" />
      </Definition>
    </Item>
    <Item name="ConstantDischarge" caption="Constant Discharge">
      <Definition valueType="real" default="0">
        <Condition type="isEqual" target="Type" value="0"/>
      </Definition>
    </Item>
    <Item name="FunctionalDischarge" caption="Variable Discharge">
      <Definition conditionType="functional">
        <Parameter valueType="real" caption="Time"/>
        <Value valueType="real" caption="Discharge(m3/s)"/>
        <Condition type="isEqual" target="Type" value="1"/>
      </Definition>
    </Item>
  </BoundaryCondition>
</SolverDefinition>
```

正しくソルバー定義ファイルが作成できているか確認します。

iRIC を起動して、ソルバー 「Sample Solver」 の新しいプロジェクトを開始します。格子を作成したりインポートしたりすると、 図 2.14 のようになります。なお、格子の作成やインポートの方法が分からない場合、ユーザマニュアルを参照して下さい。

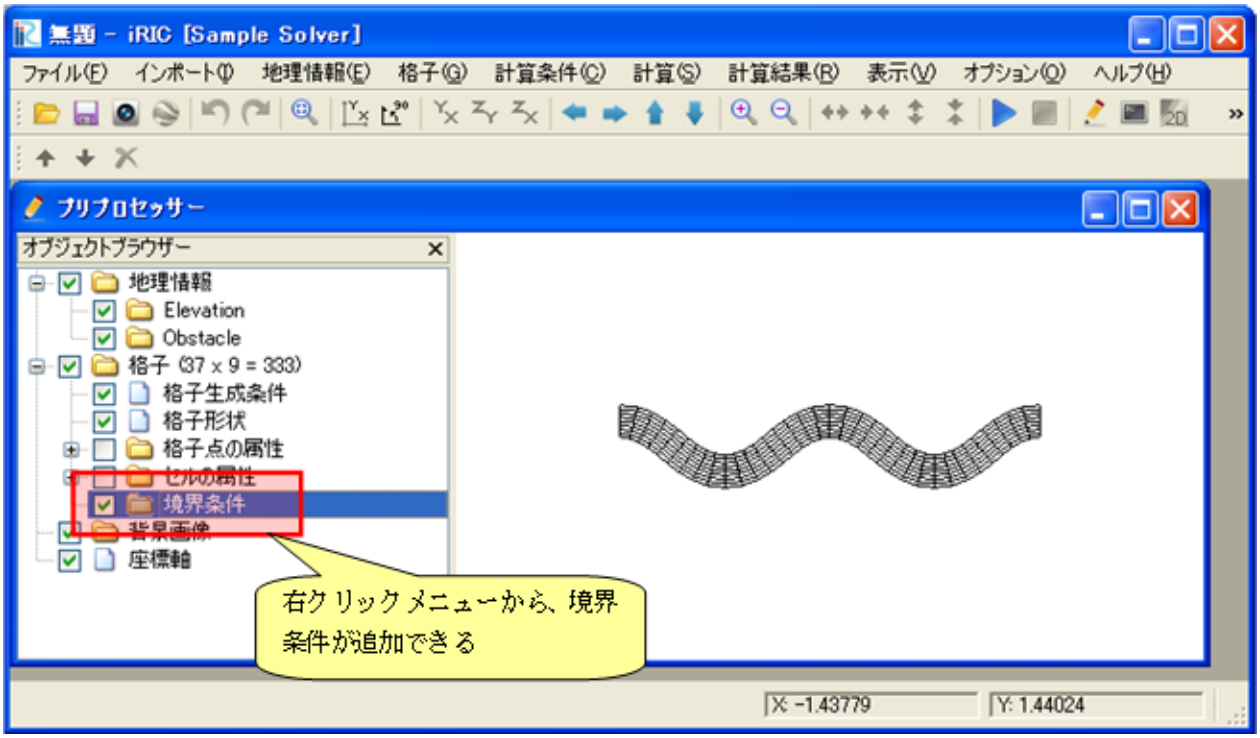

図 2.14　プリプロセッサ 表示例 (格子作成後)

右クリックメニューから「新しい Inflow の追加」を選択すると、図 2.15 に示すダイアログが表示され、境界条件を定義することが出来ます。

図 2.15　境界条件の編集ダイアログ

境界条件を定義した後、格子点を選択して右クリックメニューから「追加」を選択することで流入口にする格子点を設定できます。設定後の画面表示例を図 2.16 に示します。

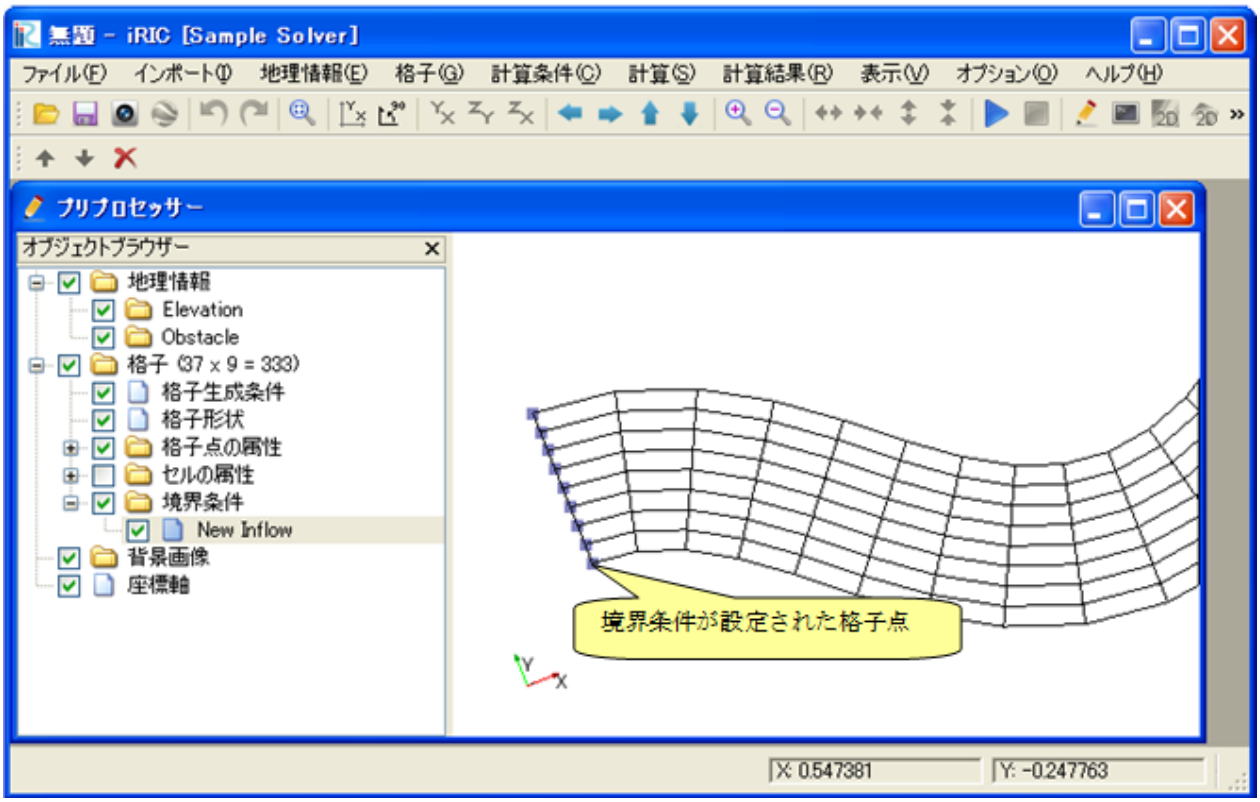

図 2.16　境界条件を設定した格子の表示例

境界条件の定義についてまとめると、以下の通りです。

- 境界条件は、 BoundaryCondition 要素で指定します。
  - Item 要素以下の構造は計算条件の Item と基本的には同じです。計算条件と同様、依存性なども定義できます。

2.4 ソルバーの作成

ソルバーを作成します。この例では、ソルバーは FORTRAN 言語で開発します。

iRIC と連携するソルバーを開発するには、ソルバー定義ファイルに従って iRIC が生成する計算データファイルを、計算条件、格子、結果の入出力に利用する必要があります。

iRIC が生成する計算データファイルは、CGNS ファイルという形式です。CGNS ファイルの入出力には、iRIClib というライブラリを使用します。

この節では、 ソルバー定義ファイルの作成 (ページ 5) で作成したソルバー定義ファイルに従って iRIC が生成する計算データファイルを読みこむソルバーを開発する流れを説明します。このソルバーで行われる入出力処理を 表 2.3 に示します。

表 2.3　ソルバーの入出力の処理の流れ

処理の内容	必須
計算データファイルを開く	○
内部変数の初期化	○
計算条件の読み込み	○
計算格子の読み込み	○
時刻 (もしくはループ回数) の出力	○
計算結果の出力	○
計算データファイルを閉じる	○

この節では、ソルバーを以下の手順で開発していきます。

1. 骨組みの作成
2. 計算データファイルを開く処理、閉じる処理の記述
3. 計算条件、計算格子の読み込み処理の記述
4. 時刻、計算結果の出力処理の記述

2.4.1 骨組みの作成

まずは、ソルバーの骨組みを作成します。リスト 2.6 に示すソースコードを作成して、sample.f90 という名前で保存します。この時点では、ソルバーは何もしていません。

このソースコードをコンパイルします。コンパイル方法は、コンパイラによって異なります。gfortran, Intel Fortran Compiler でのコンパイル方法を 7.2.1 で解説していますので、参考にしてください。

リスト 2.6　サンプルソルバー ソースコード

```
program SampleProgram
  implicit none
  include 'cgnslib_f.h'
  include 'iriclib_f.h'

  write(*,*) "Sample Program"
  stop
end program SampleProgram
```

コンパイルが成功したら、できた実行プログラムをフォルダの作成 (ページ 5) で作成したフォルダにコピーし、名前を 基本情報の作成 (ページ 6) で executable 属性に指定した名前 (この例なら 「solver.exe」) に変更してください。またこの時、ソルバーの実行に必要な DLL も同じフォルダにコピーしてください。

iRIC からソルバーが正しく起動できるか確認します。

「Example Solver」 をソルバーに用いるプロジェクトを新しく開始し、以下の操作を行って下さい。

メニュー: 計算 (C) –> 実行 (R)

ソルバーコンソールが起動され、 図 2.17 に示すように「Sample Program」 という文字列が表示されれば、ソルバーを iRIC から正しく起動できています。

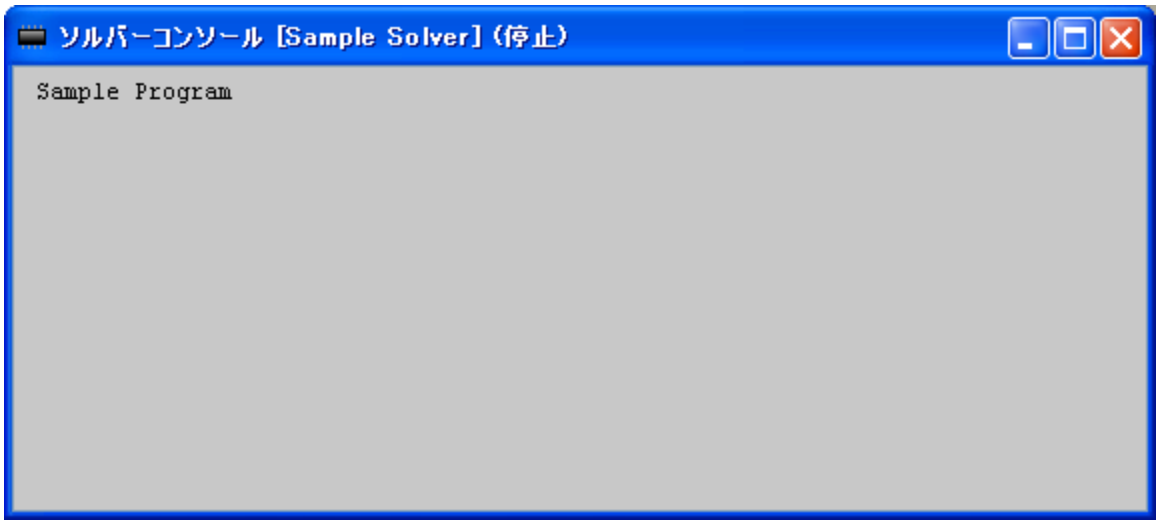

図 2.17　ソルバーコンソール表示例

2.4.2 計算データファイルを開く処理、閉じる処理の記述

計算データファイルを開く処理、閉じる処理を記述します。

ソルバーは、処理開始時に計算データファイルを開き、終了時に計算データファイルを閉じる必要があります。

iRIC は引数として計算データファイルのファイル名を渡すため、そのファイルを開きます。

引数の数と引数を取得するための方法は、コンパイラによって異なります。gfortran, Intel Fortran Compiler での引数の取得方法を 7.1 で説明していますので、参考にしてください。ここでは、Intel Fortran Compiler でコンパイルする場合の方法で記述します。

計算データファイルを開く処理と閉じる処理を追記したソースコードをリスト 2.7 に示します。強調してで示したのが追記した部分です。

リスト 2.7　計算データファイルを開く処理、閉じる処理を追記したソースコード

```
1  program SampleProgram
2    implicit none
3    include 'cgnslib_f.h'
4    integer:: fin, ier
5    integer:: icount, istatus
6    character(200)::condFile
7
```

```
  write(*,*) "Sample Program"

  icount = nargs()
  if ( icount.eq.2 ) then
    call getarg(1, condFile, istatus)
  else
    write(*,*) "Input File not specified."
    stop
  endif

  ! 計算データファイルを開く
  call cg_open_f(condFile, CG_MODE_MODIFY, fin, ier)
  if (ier /=0) stop "*** Open error of CGNS file ***"

  ! 内部変数の初期化
  call cg_iric_init_f(fin, ier)
  if (ier /=0) STOP "*** Initialize error of CGNS file ***"
  ! オプションの設定
  call iric_initoption_f(IRIC_OPTION_CANCEL, ier)
  if (ier /=0) STOP "*** Initialize option error***"

  ! 計算データファイルを閉じる
  call cg_close_f(fin, ier)
  stop
end program SampleProgram
```

骨組みの作成 (ページ 22) と同様に、ファイルのコンパイルと、実行プログラムの配置を行います。

骨組みの作成 (ページ 22) と同様の手順で、iRIC からソルバーが正しく起動できるか確認します。エラーメッセージが表示されずに終了すれば成功です。

この節で追加した関数の詳細については、6.3.2, 6.3.3, 6.3.15 を参照してください。

2.4.3 計算条件、計算格子、境界条件の読み込み処理の記述

計算条件、計算格子、境界条件の読み込み処理を記述します。

iRIC は、 ソルバー定義ファイルの作成 (ページ 5) で作成したソルバー定義ファイルに従って、計算条件、格子、格子属性、境界条件を計算データファイルに出力しますので、ソルバー定義ファイルでの記述に対応するように、計算条件、計算格子、境界条件の読み込み処理を記述します。

計算条件、計算格子の読み込み処理を追記したソースコードをリスト 2.8 に示します。強調して示したのが追記した部分です。

リスト 2.8　計算データファイルを開く処理、閉じる処理を追記したソースコード

```
program SampleProgram
  implicit none
  include 'cgnslib_f.h'
  integer:: fin, ier
  integer:: icount, istatus
  character(200)::condFile
  integer:: maxiterations
  double precision:: timestep
  integer:: surfacetype
  double precision:: constantsurface
  integer:: variable_surface_size
  double precision, dimension(:), allocatable:: variable_surface_time
  double precision, dimension(:), allocatable:: variable_surface_elevation

  integer:: isize, jsize
  double precision, dimension(:,:), allocatable:: grid_x, grid_y
  double precision, dimension(:,:), allocatable:: elevation
  integer, dimension(:,:), allocatable:: obstacle

  integer:: inflowid
  integer:: inflow_count
  integer:: inflow_element_max
  integer:: discharge_variable_sizemax
  integer, dimension(:), allocatable:: inflow_element_count
  integer, dimension(:,:,:), allocatable:: inflow_element
  integer, dimension(:), allocatable:: discharge_type
  double precision, dimension(:), allocatable:: discharge_constant
  integer, dimension(:), allocatable:: discharge_variable_size
  double precision, dimension(:,:), allocatable:: discharge_variable_time
  double precision, dimension(:,:), allocatable:: discharge_variable_value

  write(*,*) "Sample Program"

  ! (略)

  ! 内部変数の初期化
  call cg_iric_init_f(fin, ier)
  if (ier /=0) STOP "*** Initialize error of CGNS file ***"
  ! オプションの設定
  call iric_initoption_f(IRIC_OPTION_CANCEL, ier)
  if (ier /=0) STOP "*** Initialize option error***"

  ! 計算条件の読み込み
  call cg_iric_read_integer_f("maxIteretions", maxiterations, ier)
  call cg_iric_read_real_f("timeStep", timestep, ier)
  call cg_iric_read_integer_f("surfaceType", surfacetype, ier)
  call cg_iric_read_real_f("constantSurface", constantsurface, ier)
```

```
48
49    call cg_iric_read_functionalsize_f("variableSurface", variable_surface_size, ier)
50    allocate(variable_surface_time(variable_surface_size))
51    allocate(variable_surface_elevation(variable_surface_size))
52    call cg_iric_read_functional_f("variableSurface", variable_surface_time, variable_
      ↪surface_elevation, ier)
53
54    ! 格子のサイズを調べる
55    call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
56
57    ! 格子を読み込むためのメモリを確保
58    allocate(grid_x(isize,jsize), grid_y(isize,jsize))
59    ! 格子を読み込む
60    call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)
61
62    ! 格子点で定義された属性 のメモリを確保
63    allocate(elevation(isize, jsize))
64    allocate(obstacle(isize - 1, jsize - 1))
65
66    ! 属性を読み込む
67    call cg_iric_read_grid_real_node_f("Elevation", elevation, ier)
68    call cg_iric_read_grid_integer_cell_f("Obstacle", obstacle, ier)
69
70    ! 流入口の数に従って、境界条件を保持するメモリを確保。
71    allocate(inflow_element_count(inflow_count))
72    allocate(discharge_type(inflow_count), discharge_constant(inflow_count))
73    allocate(discharge_variable_size(inflow_count))
74
75    ! 流入口に指定された格子点の数と、時間依存の流入量のサイズを調べる
76    inflow_element_max = 0
77    do inflowid = 1, inflow_count
78      ! 流入口に指定された格子点の数
79      call cg_iric_read_bc_indicessize_f('inflow', inflowid, inflow_element_
      ↪count(inflowid))
80      if (inflow_element_max < inflow_element_count(inflowid)) then
81        inflow_element_max = inflow_element_count(inflowid)
82      end if
83      ! 流入口の時間依存の流入量のデータの数
84      call cg_iric_read_bc_functionalsize_f('inflow', inflowid, 'FunctionalDischarge',␣
      ↪discharge_variable_size(inflowid), ier);
85      if (discharge_variable_sizemax < discharge_variable_size(inflowid)) then
86        discharge_variable_sizemax = discharge_variable_size(inflowid)
87      end if
88    end do
89
90    ! 流入口に指定された格子点と、時間依存の流入量を保持するメモリを確保。
91    allocate(inflow_element(inflow_count, 2, inflow_element_max))
92    allocate(discharge_variable_time(inflow_count, discharge_variable_sizemax))
93    allocate(discharge_variable_value(inflow_count, discharge_variable_sizemax))
```

```
94
95    ! 境界条件の読み込み
96    do inflowid = 1, inflow_count
97      ! 流入口に指定された格子点
98      call cg_iric_read_bc_indices_f('inflow', inflowid, inflow_
    ↪element(inflowid:inflowid,:,:), ier)
99      ! 流入量の種類 (0 = 一定、1 = 時間依存)
100     call cg_iric_read_bc_integer_f('inflow', inflowid, 'Type', discharge_
    ↪type(inflowid:inflowid), ier)
101     ! 流入量 (一定)
102     call cg_iric_read_bc_real_f('inflow', inflowid, 'ConstantDischarge', discharge_
    ↪constant(inflowid:inflowid), ier)
103     ! 流入量 (時間依存)
104     call cg_iric_read_bc_functional_f('inflow', inflowid, 'FunctionalDischarge',␣
    ↪discharge_variable_time(inflowid:inflowid,:), discharge_variable_
    ↪value(inflowid:inflowid,:), ier)
105   end do
106
107   ! 計算データファイルを閉じる
108   call cg_close_f(fin, ier)
109   stop
110 end program SampleProgram
```

計算条件などを読み込む関数に渡す引数が、計算条件の定義 (ページ 9), 格子属性の定義 (ページ 14) でソルバー定義ファイルに定義した Item 要素の name 属性と一致していることに注目してください。

なお、ソルバー定義ファイルで定義する計算条件、格子、格子属性と、それを読み込むための iRIClib の関数の対応関係については、 5.3.1 を参照してください。

また、計算条件、計算格子、格子属性の読み込みに使う関数の詳細については、6.3.5, 6.3.6 を参照してください。

2.4.4 時刻、計算結果の出力処理の記述

時刻、計算結果の出力処理を記述します。

時間依存の方程式を解くソルバーの場合、タイムステップの数だけ時刻、計算結果の出力を繰り返します。

また、時刻、計算結果の出力のたびにユーザがソルバーの実行を中止していないか確認し、中止していたら実行を中止します。

なお、ソルバーが出力する計算結果についてはソルバー定義ファイルには記述しませんので、ソルバー定義ファイルとの対応関係を気にせず記述できます。

時刻、計算結果の出力処理を追記したソースコードをリスト 2.9 に示します。強調して示したのが追記した部分です。

リスト 2.9　時刻、計算結果の出力処理を追記したソースコード

```
  ! (略)
  integer:: isize, jsize
  double precision, dimension(:,:), allocatable:: grid_x, grid_y
  double precision, dimension(:,:), allocatable:: elevation
  integer, dimension(:,:), allocatable:: obstacle
  double precision:: time
  integer:: iteration
  integer:: canceled
  integer:: locked
  double precision, dimension(:,:), allocatable:: velocity_x, velocity_y
  double precision, dimension(:,:), allocatable:: depth
  integer, dimension(:,:), allocatable:: wetflag
  double precision:: convergence

  ! (略)

  ! 属性を読み込む
  call cg_iric_read_grid_real_node_f("Elevation", elevation, ier)
  call cg_iric_read_grid_integer_cell_f("Obstacle", obstacle, ier)

  allocate(velocity_x(isize,jsize), velocity_y(isize,jsize), depth(isize,jsize),␣
↪wetflag(isize,jsize))
  iteration = 0
  time = 0
  do
    time = time + timestep
    ! (ここで計算を実行。格子の形状も変化)

    call iric_check_cancel_f(canceled)
    if (canceled == 1) exit
    call iric_check_lock_f(condFile, locked)
    do while (locked == 1)
      sleep(1)
      call iric_check_lock_f(condFile, locked)
    end do
    call iric_write_sol_start_f(condFile, ier)
    call cg_iric_write_sol_time_f(time, ier)
    ! 格子を出力
    call cg_iric_write_sol_gridcoord2d_f (grid_x, grid_y, ier)
    ! 計算結果を出力
    call cg_iric_write_sol_real_f ('VelocityX', velocity_x, ier)
    call cg_iric_write_sol_real_f ('VelocityY', velocity_y, ier)
    call cg_iric_write_sol_real_f ('Depth', depth, ier)
    call cg_iric_write_sol_integer_f ('Wet', wetflag, ier)
    call cg_iric_write_sol_baseiterative_real_f ('Convergence', convergence, ier)
    call cg_iric_flush_f(condFile, fin, ier)
    call iric_write_sol_end_f(condFile, ier)
```

```
    iteration = iteration + 1
    if (iteration > maxiterations) exit
  end do

  ! 計算データファイルを閉じる
  call cg_close_f(fin, ier)
  stop
end program SampleProgram
```

時刻、計算結果の出力に使う関数の詳細については、6.3.10, 6.3.12 を参照してください。計算実行中に格子形状が変化する場合、6.3.11 で説明する関数を使用してください。

計算結果については、iRIC では特別な名前が定義されており、特定の目的で使用される結果ではその名前を使用する必要があります。特別な計算結果の名前については 7.3.2 を参照してください。

2.5 ソルバー定義ファイルの辞書ファイルの作成

ソルバー定義ファイルで用いられている文字列のうち、ダイアログ上などに表示される文字列を翻訳して表示するための辞書ファイルを作成します。

まず、iRIC から、以下のメニューを起動します。すると、ソルバー定義ファイルの辞書更新ウィザードが表示されます。ダイアログの表示例を、図 2.18 〜 図 2.20 に示します。

メニュー: オプション (O) –> 辞書ファイルの作成・更新 (C)

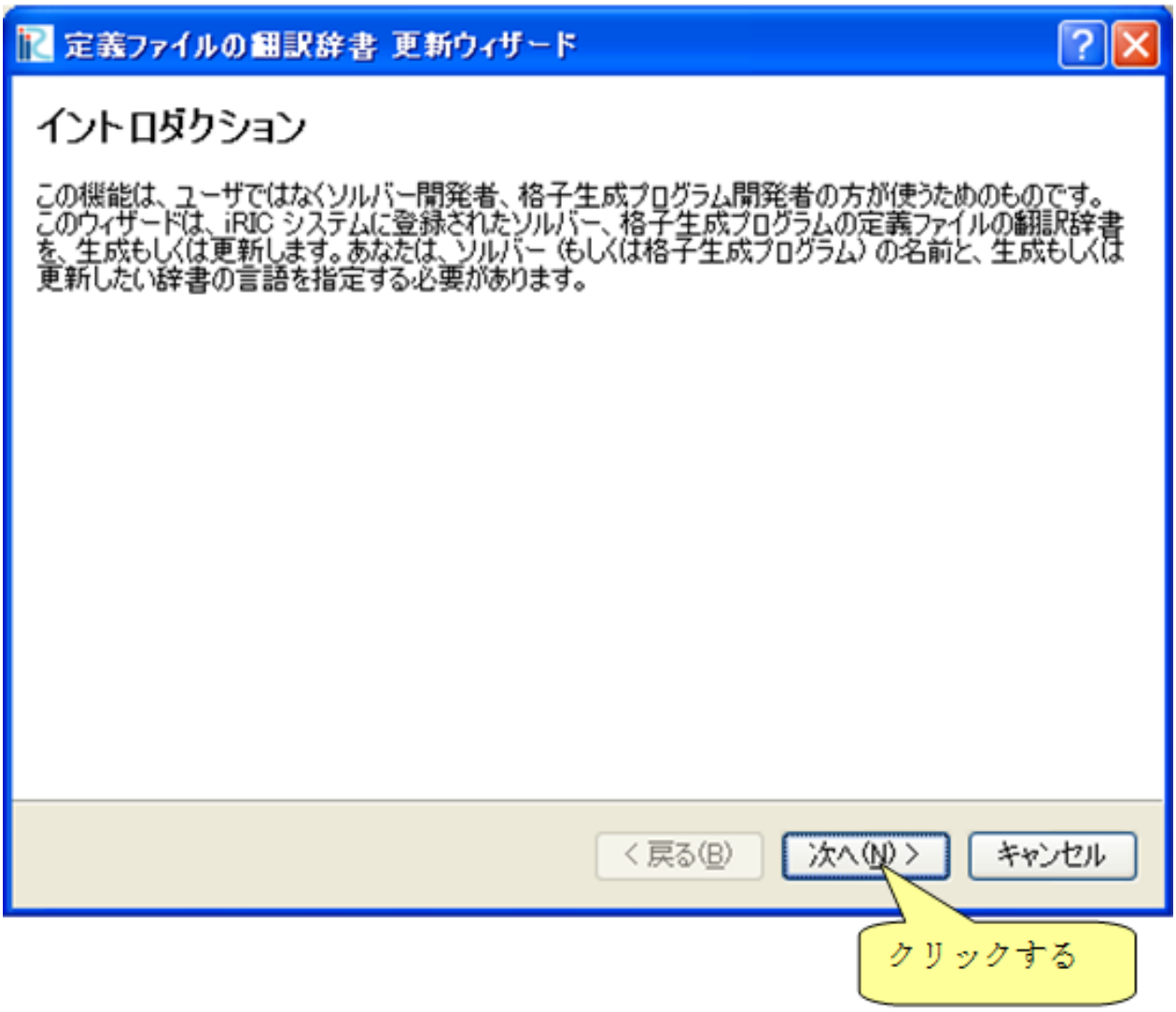

図 2.18　ソルバー定義ファイルの辞書更新ウィザード 表示例 (1 ページ目)

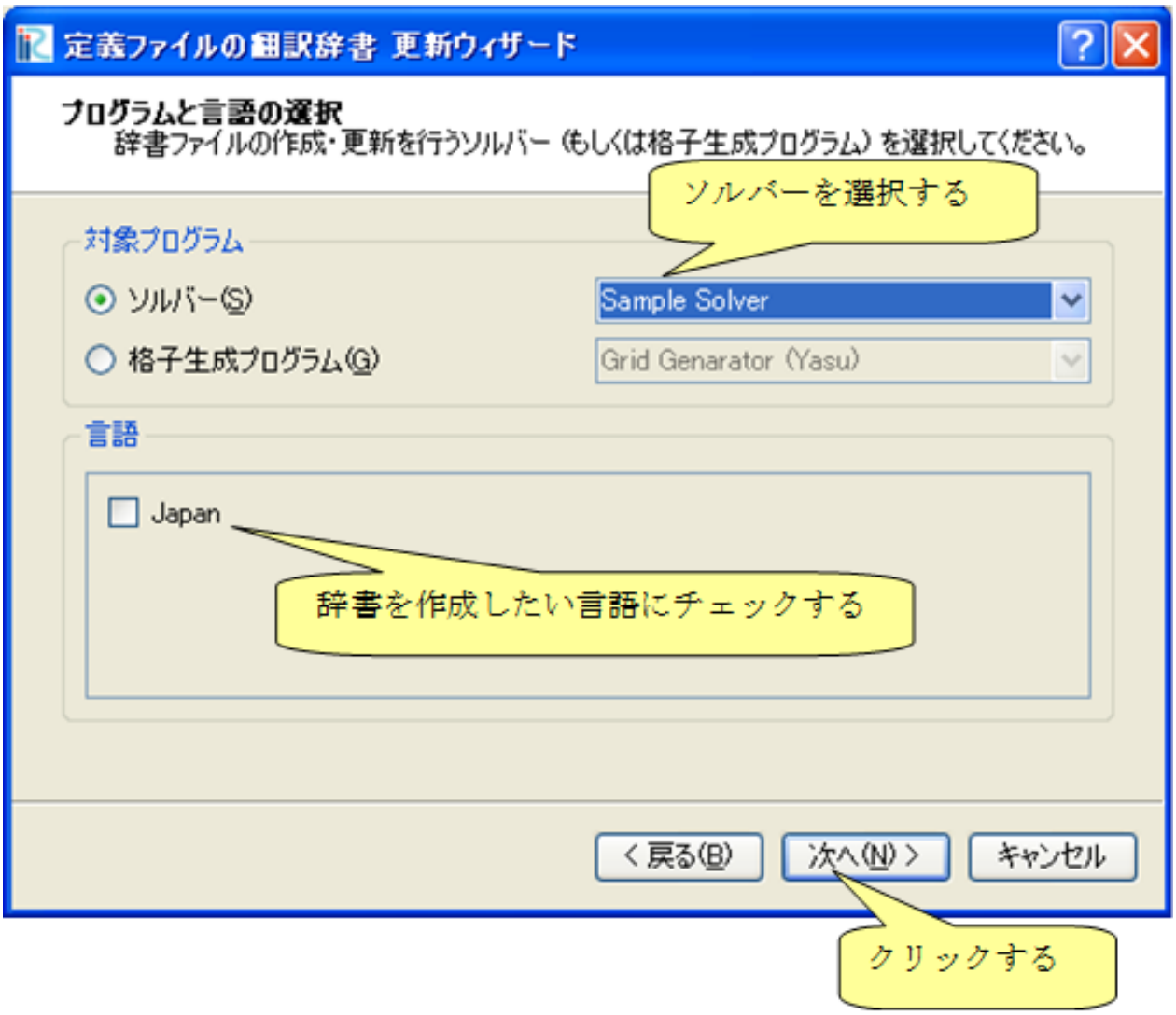

図 2.19　ソルバー定義ファイルの辞書更新ウィザード 表示例 (2 ページ目)

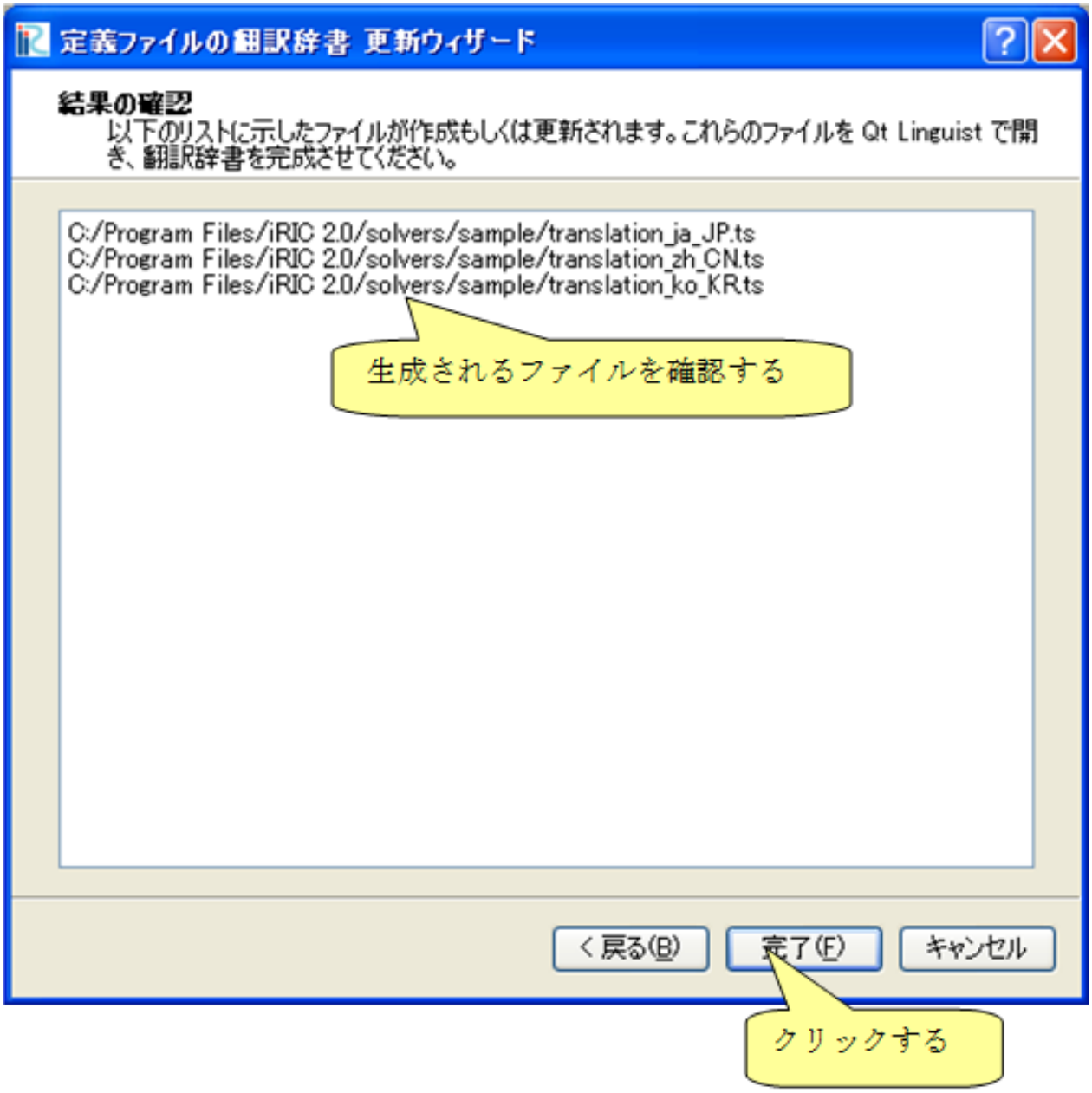

図 2.20　ソルバー定義ファイルの辞書更新ウィザード 表示例 (3 ページ目)

辞書ファイルは、ソルバー定義ファイルと同じフォルダに作成されます。作成された辞書ファイルは、翻訳前の英語のみが含まれています。辞書ファイルはテキストファイルですので、テキストエディタなどで開いて編集します。辞書ファイルは、文字コードに UTF-8 を指定して保存してください。

辞書ファイルの編集例を、 リスト 2.10 、リスト 2.11 に示します。例に示したように、translation 要素の中に翻訳後の文字列を追記してください。

表 2?16 ソルバー定義ファイルの辞書ファイルの一部 (編集前)

リスト 2.10　ソルバー定義ファイルの辞書ファイルの一部 (編集前)

```
<message>
  <source>Basic Settings</source>
  <translation></translation>
</message>
```

リスト 2.11　ソルバー定義ファイルの辞書ファイルの一部 (編集後)

```
<message>
  <source>Basic Settings</source>
  <translation>基本設定</translation>
</message>
```

なお、辞書ファイルは、Qt に付属する Qt Linguist を利用して編集することもできます。Qt Linguist の画面表示例を 図 2?18 に示します。Qt Linguist は、以下の URL からダウンロードできる Qt に含まれています。

https://www.qt.io/download/

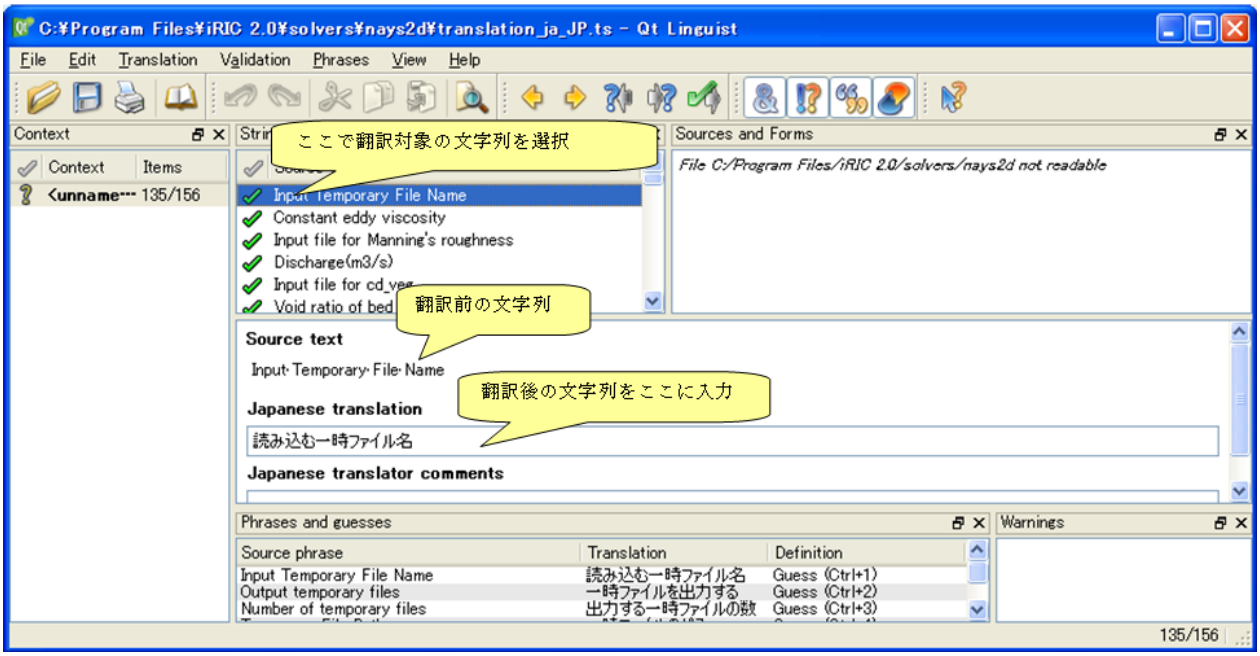

図 2.21　Qt Linguist 画面表示例

翻訳が完了したら、iRIC を確認したい言語に切り替えてから iRIC を起動し直し、正しく翻訳されて表示されるか確認します。翻訳完了後のプリプロセッサ、計算条件設定ダイアログの表示例をそれぞれ図 2.22, 図 2.23 に示します。

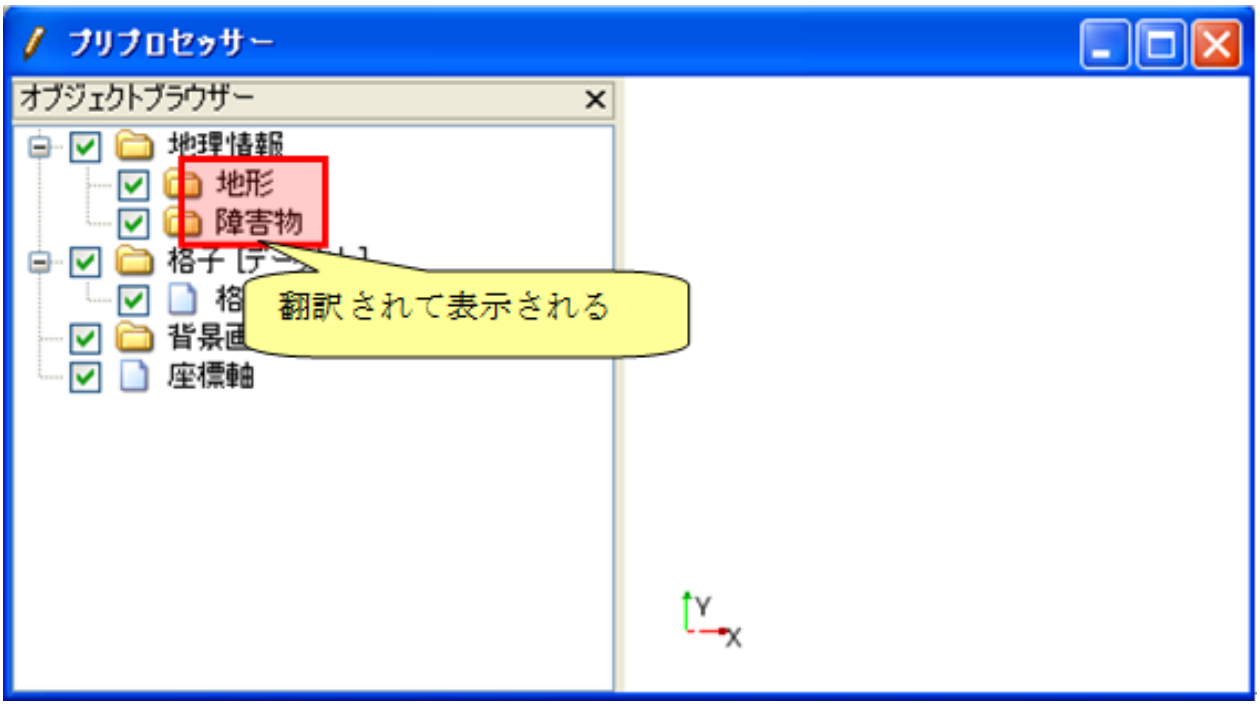

図 2.22　翻訳完了後のプリプロセッサ 表示例

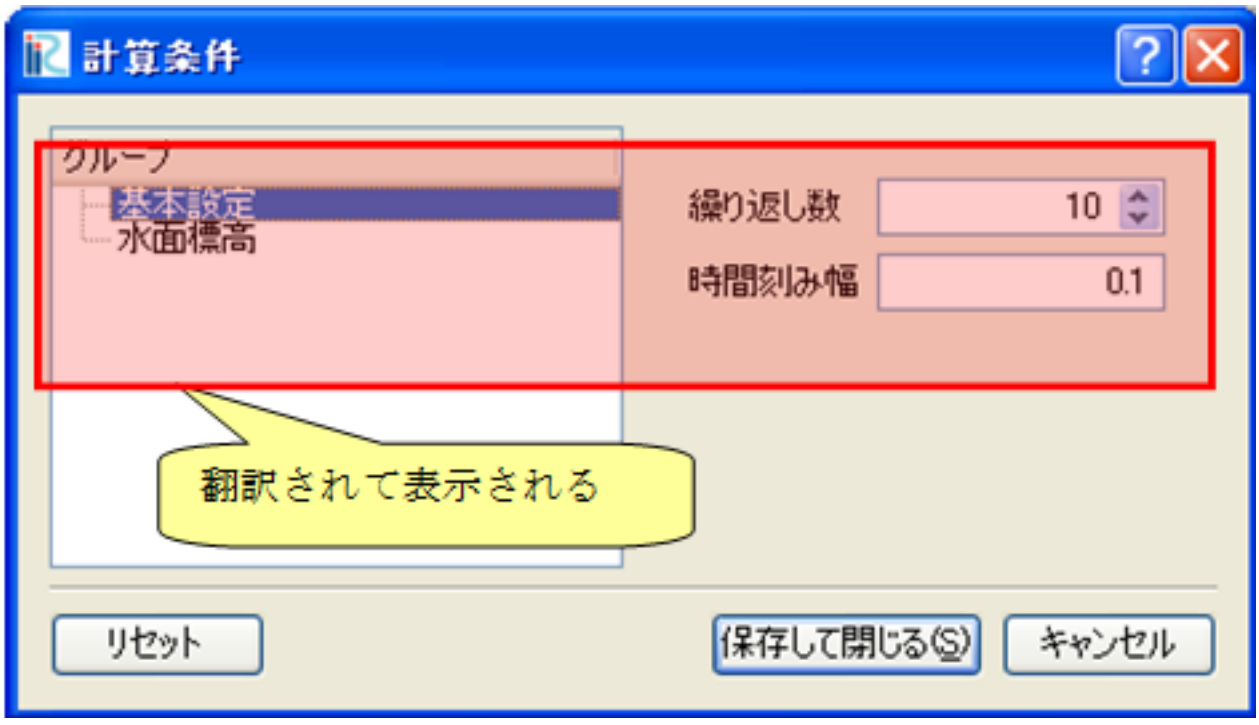

図 2.23　翻訳完了後の計算条件設定ダイアログ 表示例

2.6 説明ファイルの作成

ソルバーの概要などについて説明するファイルを作成します。

README というファイル名のテキストファイルを、フォルダの作成 (ページ 5) で作成したフォルダの下に作成します。文字コードは UTF-8 にします。

説明ファイルは、以下の例のようなファイル名で言語ごとに用意します。言語ごとの説明ファイルがない場合、README が使用されます。

- 英語: README
- 日本語: README_ja_JP

「README_」 以降につく文字列は、辞書ファイルの 「translation_*****.ts」 の「*****」 の部分と同じですので、日本語以外の説明ファイルを作る際のファイル名は、辞書ファイルのファイル名を参考にして決めて下さい。

説明ファイルの内容は、iRIC 上で新規プロジェクトを作成する際のソルバー選択ダイアログで、説明タブに表示されます。ファイルを作成したら、iRIC 上で正しく表示されるか確認して下さい。ダイアログの表示例を、図 2.24 に示します。

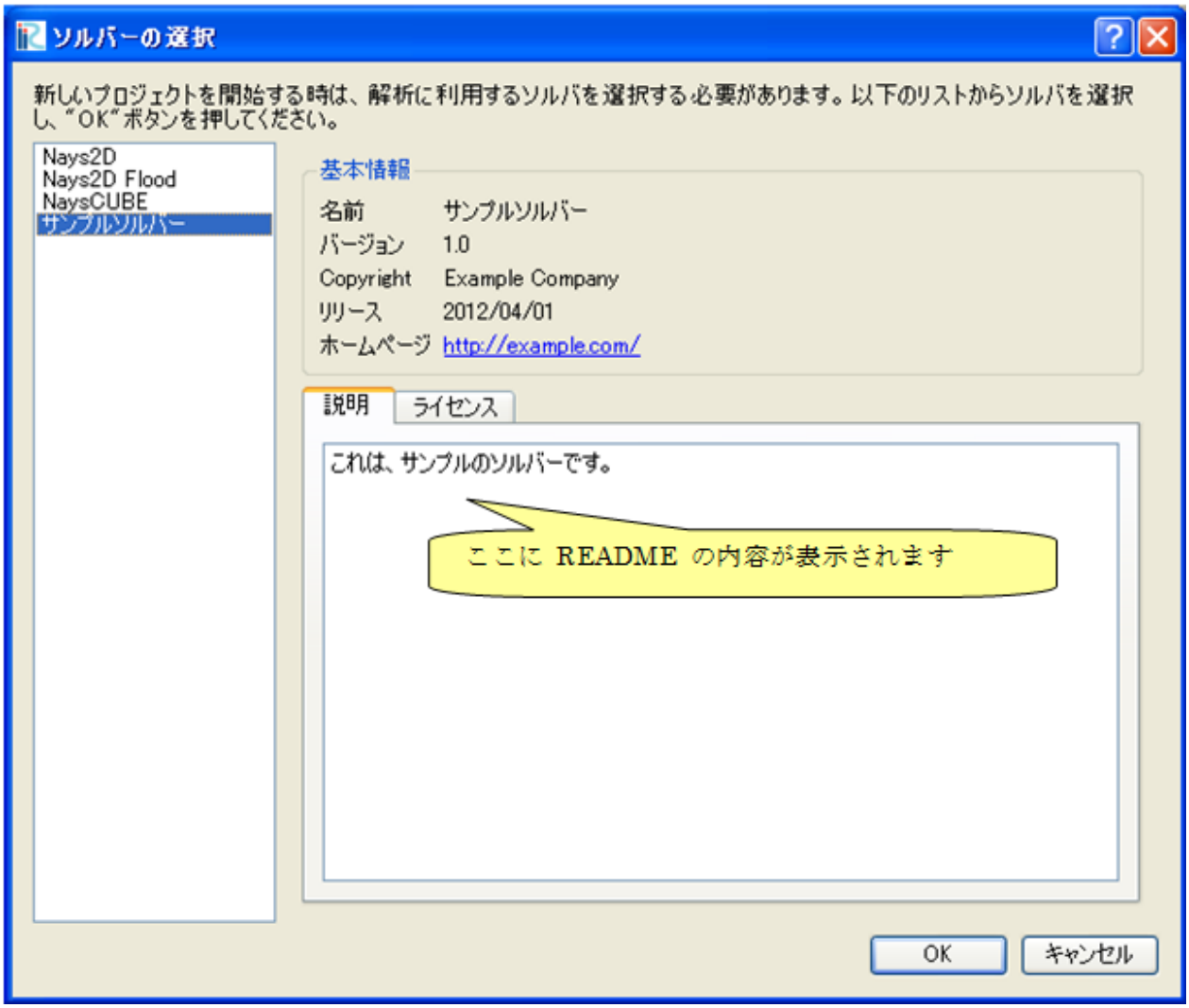

図 2.24　ソルバー選択ダイアログ 表示例

2.7 ライセンス情報ファイルの作成

ソルバーの利用ライセンスについて説明するファイルを作成します。

LICENSE というファイル名のテキストファイルを、フォルダの作成 (ページ 5) で作成したフォルダの下に作成します。文字コードは UTF-8 にします。

ライセンス情報ファイルは、以下の例のようなファイル名で言語ごとに用意します。言語ごとのライセンスファイルがない場合、 LICENSE が使用されます。

- 英語: LICENSE
- 日本語: LICENSE_ja_JP

「LICENSE_」 以降につく文字列は、辞書ファイルの「translation_*****.ts」 の 「*****」 の部分と同じですので、

日本語以外の説明ファイルを作る際のファイル名は、辞書ファイルのファイル名を参考にして決めて下さい。

ライセンス情報ファイルの内容は、iRIC 上で新規プロジェクトを作成する際のソルバー選択ダイアログで、ライセンスタブに表示されます。ファイルを作成したら、iRIC 上で正しく表示されるか確認して下さい。ダイアログの表示例を、 図 2.25 に示します。

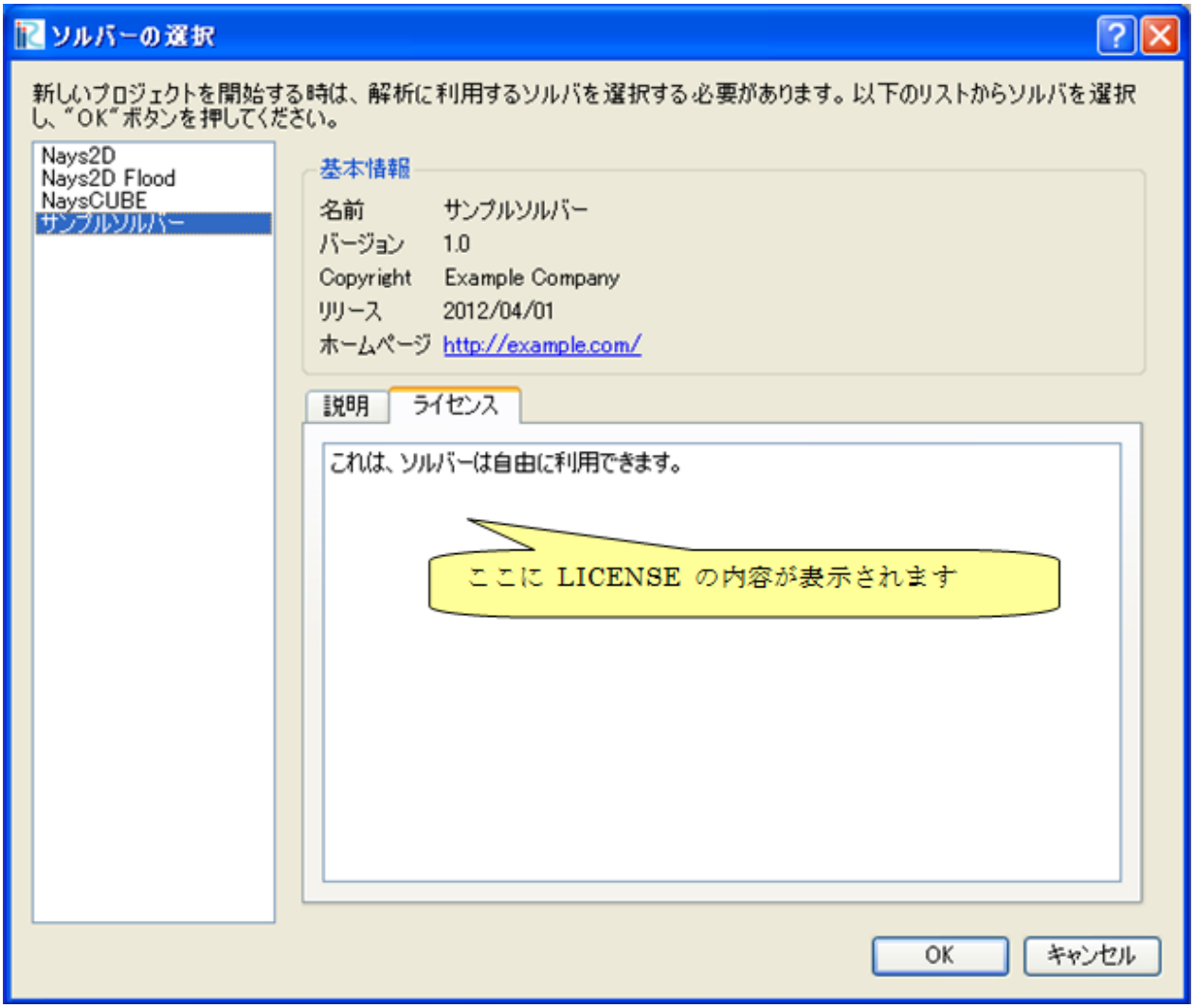

図 2.25　ソルバー選択ダイアログ 表示例

第 3 章

計算結果分析ソルバーの開発手順

3.1 概要

iRIC では、既存の CGNS ファイルの計算結果を読み込み、分析（・加工）することができます。分析結果は、新たな CGNS ファイルに書き出すことができます。計算結果分析ソルバーの開発手順は、通常のソルバー開発手順と同様です (ソルバーの開発手順 (ページ 3) 参照)。

ここでは、計算結果分析ソルバーを FORTRAN で開発する例を説明します。

一つのソルバーで複数の CGNS ファイルを扱う場合、操作対象の CGNS ファイルを指定するために、ソルバーの開発手順 (ページ 3) で使用した関数とは別の関数を用います (6.4.1 参照) 。複数 CGNS ファイル用の関数は、末尾が 「_mul_f」 で終わっており、ファイル ID を第一引数とします。また、計算結果読み込み用に既存の CGNS を開く際は、cg_iric_init_f の代わりに cg_iric_initread_f を用いて初期化を行います。複数の CGNS ファイルを扱ったソースコードの例をリスト 3.1 に示します。

リスト 3.1　複数 CGNS ファイルを扱ったソースコード（抜粋）

```
! （前略）

! ファイルオープン、初期化
call cg_open_f(cgnsfile, CG_MODE_MODIFY, fin1, ier)
call cg_iric_init_f(fin1, ier)

! （略）

! 計算条件の読み込み等
call cg_iric_read_functionalsize_mul_f(fin1, 'func', param_func_size, ier)

! （略）

!ファイルオープン、初期化（計算結果読み込み用）
call cg_open_f(param_inputfile, CG_MODE_READ, fin2, ier)
call cg_iric_initread_f(fin2, ier)

```

```
! （略）

! 計算結果の読み込み等
call cg_iric_read_sol_count_mul_f(fin2, solcount, ier)

! （略）

! 計算結果の分析等

! （略）

! 分析結果等の出力
call cg_iric_write_sol_time_mul_f(fin1, t, ier)

! （略）

! ファイルのクローズ
call cg_close_f(fin1, ier)
call cg_close_f(fin2, ier)

! （後略）
```

既存の CGNS の計算結果をもとに、「魚の生息しやすさ」を算出するソルバーのソースコードをリスト 3.2 に示します。

リスト 3.2　既存の CGNS ファイルを読み込み、分析するソルバーのソースコード

```
program SampleProgram2
  implicit none
  include 'cgnslib_f.h'

  integer icount
  character(len=300) cgnsfile

  integer:: fin1, fin2, ier, istatus

  character(len=300) param_inputfile
  integer:: param_result
  character(len=100) param_resultother
  integer:: param_func_size
  double precision, dimension(:), allocatable:: param_func_param
  double precision, dimension(:), allocatable:: param_func_value
  character(len=100) resultname

  integer:: isize, jsize
  double precision, dimension(:,:), allocatable:: grid_x, grid_y
  double precision, dimension(:,:), allocatable:: target_result
  double precision, dimension(:,:), allocatable:: analysis_result
  double precision:: tmp_target_result
```

```
  double precision:: tmp_analysis_result

  integer:: i, j, f, solid, solcount, iter
  double precision:: t

  ! Intel Fortran 用の記述。
  icount = nargs()
  if (icount.eq.2) then
    call getarg(1, cgnsfile, istatus)
  else
    write(*,*) "Input File not specified."
    stop
  end if

  ! CGNS ファイルのオープン
  call cg_open_f(cgnsfile, CG_MODE_MODIFY, fin1, ier)
  if (ier /=0) STOP "*** Open error of CGNS file ***"
  ! 内部変数の初期化
  call cg_iric_init_f(fin1, ier)

  ! 計算条件を読み込む
  call cg_iric_read_string_mul_f(fin1, 'inputfile', param_inputfile, ier)
  call cg_iric_read_integer_mul_f(fin1, 'result', param_result, ier)
  call cg_iric_read_string_mul_f(fin1, 'resultother', param_resultother, ier)

  call cg_iric_read_functionalsize_mul_f(fin1, 'func', param_func_size, ier)
  allocate(param_func_param(param_func_size), param_func_value(param_func_size))
  call cg_iric_read_functional_mul_f(fin1, 'func', param_func_param, param_func_value,␣
↪ier)

  if (param_result .eq. 0) resultname = 'Depth(m)'
  if (param_result .eq. 1) resultname = 'Elevation(m)'
  if (param_result .eq. 2) resultname = param_resultother

  ! 指定された CGNS ファイルから、格子を読み込む
  call cg_open_f(param_inputfile, CG_MODE_READ, fin2, ier)
  if (ier /=0) STOP "*** Open error of CGNS file 2 ***"
  call cg_iric_initread_f(fin2, ier)

  ! 格子を読み込む
  call cg_iric_gotogridcoord2d_mul_f(fin2, isize, jsize, ier)
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  call cg_iric_getgridcoord2d_mul_f(fin2, grid_x, grid_y, ier)

  ! 読み込んだ格子を cgnsfile に出力する
  call cg_iric_writegridcoord2d_mul_f(fin1, isize, jsize, &
    grid_x, grid_y, ier)

  ! 計算結果を読み込んで加工するためのメモリを確保
```

```
  allocate(target_result(isize, jsize), analysis_result(isize, jsize))

  ! 計算結果を処理
  call cg_iric_read_sol_count_mul_f(fin2, solcount, ier)

  do solid = 1, solcount
    ! 計算結果を読み込み
    call cg_iric_read_sol_time_mul_f(fin2, solid, t, ier)
    call cg_iric_read_sol_real_mul_f(fin2, solid, resultname, &
      target_result, ier)

    ! 読み込んだ計算結果をもとに、魚の生息しやすさを算出する。
    do i = 1, isize
      do j = 1, jsize
        tmp_target_result = target_result(i, j)
        do f = 1, param_func_size
          if ( &
            param_func_param(f) .le. tmp_target_result .and. &
            param_func_param(f + 1) .gt. tmp_target_result) then
            tmp_analysis_result = &
              param_func_value(f) + &
              (param_func_value(f + 1) - param_func_value(f)) / &
              (param_func_param(f + 1) - param_func_param(f)) * &
              (tmp_target_result - param_func_param(f))
          endif
        end do
        analysis_result(i, j) = tmp_analysis_result
      end do
    end do

    ! 処理済みの計算結果を出力
    call cg_iric_write_sol_time_mul_f(fin1, t, ier)
    call cg_iric_write_sol_real_mul_f(fin1, 'fish_existence', analysis_result, ier)
  end do

  ! CGNS ファイルのクローズ
  call cg_close_f(fin1, ier)
  call cg_close_f(fin2, ier)
  stop
end program SampleProgram2
```

第 4 章

格子生成プログラムの開発手順

4.1 概要

格子生成プログラムは、格子生成条件に基づいて、格子を生成するプログラムです。作成したプログラムは、iRIC 上から格子生成アルゴリズムの 1 つとして利用できるようになります。

iRIC 上で動作する格子生成プログラムを開発するには、表 2 1 に示すようなファイルを作成、配置する必要があります。表 2 1 に示した項目のうち、“iRIC 2.0” フォルダと “gridcreators” フォルダは、iRIC をインストールすれば既に作成されています。ソルバー開発者は、“gridcreators” フォルダの下に自分が開発する格子生成プログラム専用のフォルダを作成し、関連するファイルをその下に配置します。

ソルバーは、格子、計算条件などに基づいて河川シミュレーションを実行し、計算結果を出力するプログラムです。

iRIC 上で動作するソルバーを開発するには、表 4.1 に示すようなファイルを作成、配置する必要があります。

表 4.1 に示したファイルは iRIC インストール先の下の 「solvers」 フォルダの下に自分が開発するソルバー専用のフォルダを作成し、その下に配置します。

表 4.1　格子生成プログラム関連ファイル一覧

ファイル名	説明
definition.xml	格子生成プログラム定義ファイル。英語で記述する。
generator.exe (例)	格子生成プログラムの実行モジュール。ファイル名は開発者が任意に選べる。
translation_ja_JP.ts など	格子生成プログラム定義ファイルの辞書ファイル。
README	格子生成プログラムの説明ファイル

各ファイルの概要は以下の通りです。

4.1.1 definition.xml

格子生成プログラムに関する以下の情報を定義するファイルです。

- 基本情報
- 格子生成条件

iRIC は格子生成プログラム定義ファイルを読み込むことで、格子生成条件を作成するためのインターフェースを提供し、そのプログラム用の格子生成データファイルを生成します。また、この格子生成プログラムが生成する格子に現在使っているソルバーが対応している時のみ、この格子生成プログラムを使えるようにします。

格子生成プログラム定義ファイルは、すべて英語で記述します。

4.1.2 格子生成プログラム

格子を生成するプログラムです。iRIC で作成した格子生成条件を読みこんで格子を生成し、生成した格子を出力します。

格子生成条件、格子の入出力には、iRIC が生成する格子生成データファイルを使用します。

FORTRAN, C 言語、C++ 言語のいずれかの言語で開発します。この章では、FORTRAN で開発する例を説明します。

4.1.3 translation_ja_JP.ts など

格子生成プログラム定義ファイルで用いられている文字列のうち、ダイアログ上に表示される文字列を翻訳して表示するための辞書ファイルです。日本語 (translation_ja_JP.ts)、韓国語 (translation_ka_KR.ts) など言語ごとに別ファイルとして作成します。

4.1.4 README

格子生成プログラムに関する説明を記述するテキストファイルです。iRIC で格子生成アルゴリズムを選択する画面で、説明欄に表示されます。

iRIC、格子生成プログラム、関連ファイルの関係を図 4.1 に示します。

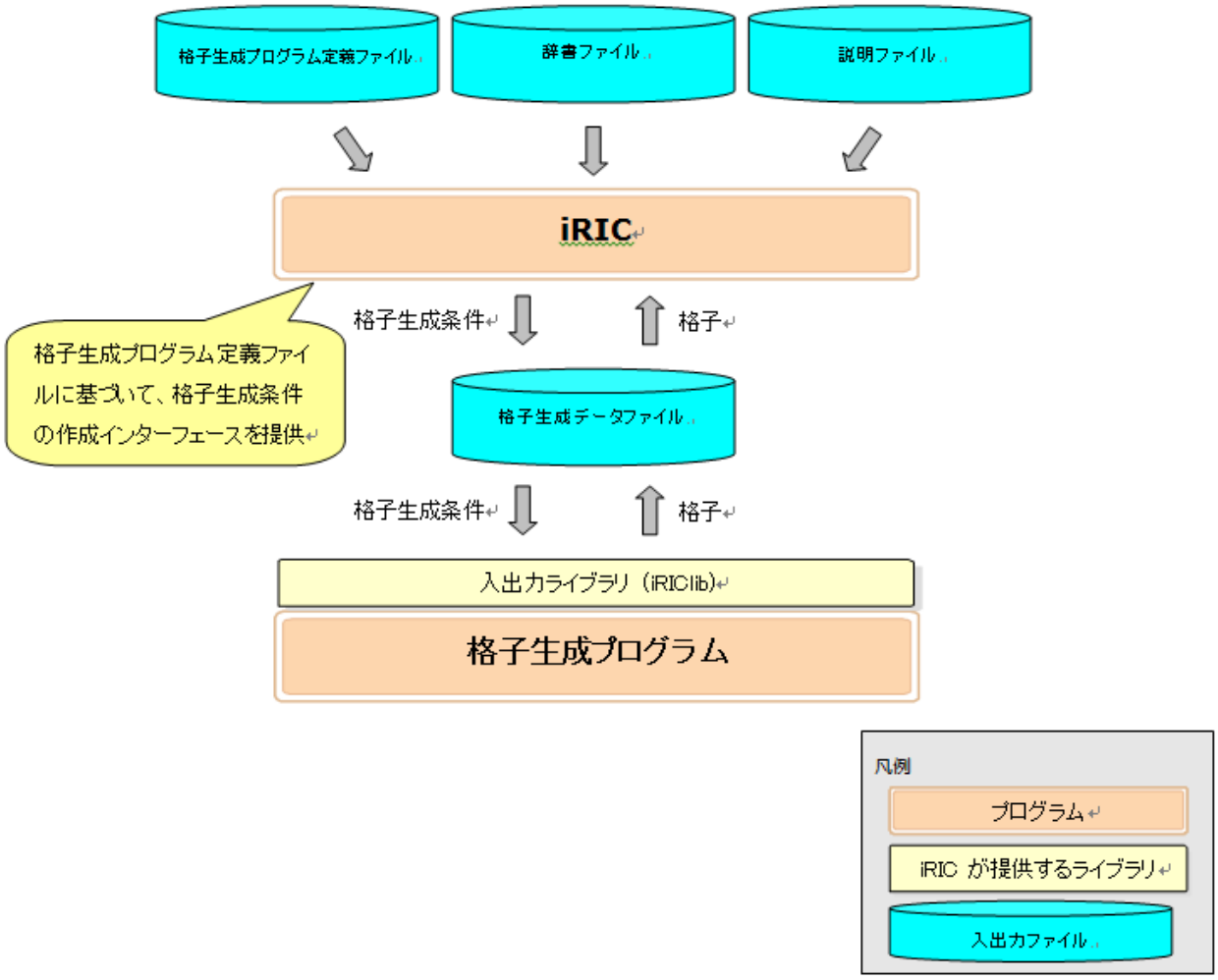

図 4.1　iRIC、格子生成プログラム、関連ファイルの関係図

この章では、この節で説明した各ファイルを作成する手順を、順番に説明します。

4.2 フォルダの作成

iRIC のインストールフォルダ の下にある 「gridcreators」フォルダの下に、開発するソルバーのための専用のフォルダを作成します。今回は、」example」というフォルダを作成します。

4.3 格子生成プログラム定義ファイルの作成

格子生成プログラム定義ファイルを作成します。

格子生成プログラム定義ファイルは、格子生成プログラムに関する 表 4?2 に示す情報を定義します。

ソルバー定義ファイルを作成します。

ソルバー定義ファイルは、ソルバーに関する 表 4.2 に示す情報を定義します。

表 4.2　格子生成プログラム定義ファイルで定義する情報

項目	説明	必須
基本情報	格子生成プログラムの名前、開発者、リリース日など	○
格子生成条件	格子の生成に必要な格子生成条件	○
エラーコード	エラー発生時のコードとメッセージの対応表	

定義ファイルは、マークアップ言語の一種である XML 言語で記述します。XML 言語の基礎については 5.5 を参照してください。

この節では、ソルバー定義ファイルを、表 4.2 に示した順で作成していきます。

4.3.1 基本情報の作成

ソルバーの基本情報を作成します。 リスト 4.1 に示すようなファイルを作り、 フォルダの作成 (ページ 45) で作成した 「example」 フォルダの下に 「definition.xml」 の名前で保存します。

リスト 4.1　基本情報を記述した格子生成プログラム定義ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<GridGeneratorDefinition
  name="samplecreator"
  caption="Sample Grid Creator"
  version="1.0"
  copyright="Example Company"
  executable="generator.exe"
  gridtype="structured2d"
>
  <GridGeneratingCondition>
  </GridGeneratingCondition>
</GridGeneratorDefinition>
```

この時点では、格子生成プログラム定義ファイルの構造は図 4.2 に示すようになっています。

要素	備考
GridGeneratorDefinition	基本情報を登録。
GridGeneratingCondition	格子生成条件を定義。現在は空。

図 4.2　格子生成プログラム定義ファイルの構造

正しく定義ファイルが作成できているか確認します。

iRIC を起動します。 図 4.3 に示すダイアログが表示されますので、「新しいプロジェクト」ボタンを押します。図 4.4 に示すダイアログが表示されますので、 「Nays2D」 を選択して 「OK」 ボタンを押し、新しいプロジェクトを開始します。

次に、メニューから以下の操作を行い、格子生成アルゴリズムの選択画面を表示します。

メニュー: 格子 (C) –> 格子生成アルゴリズムの選択 (S)

格子生成アルゴリズムの選択ダイアログの表示例を図 4.5 に示します。ここに、先ほど作成した定義ファイルで指定した「Sample Grid Creator」が表示されていることを確認します。確認できたら、キャンセルボタンを押します。

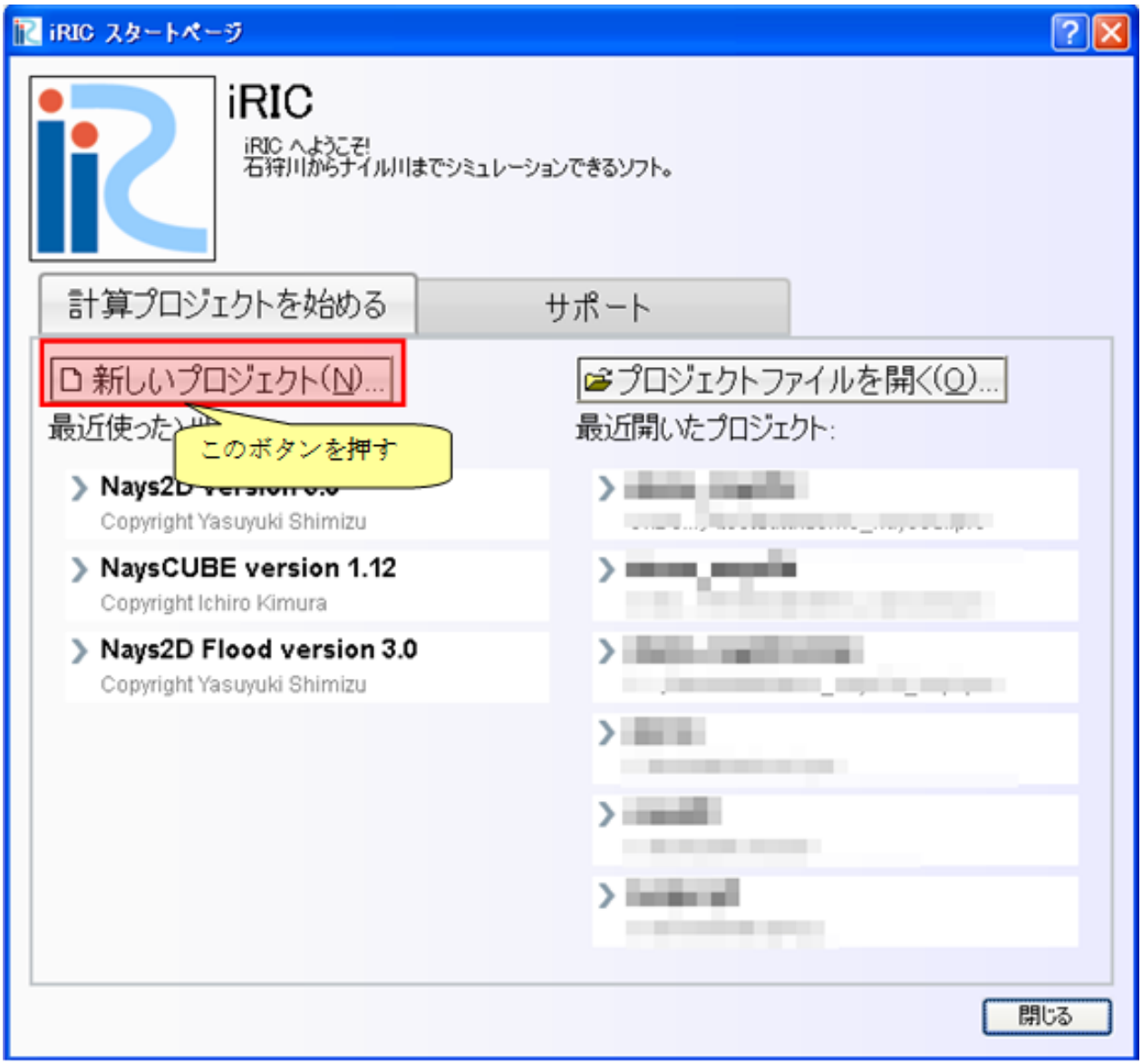

図 4.3　iRIC のスタートダイアログ

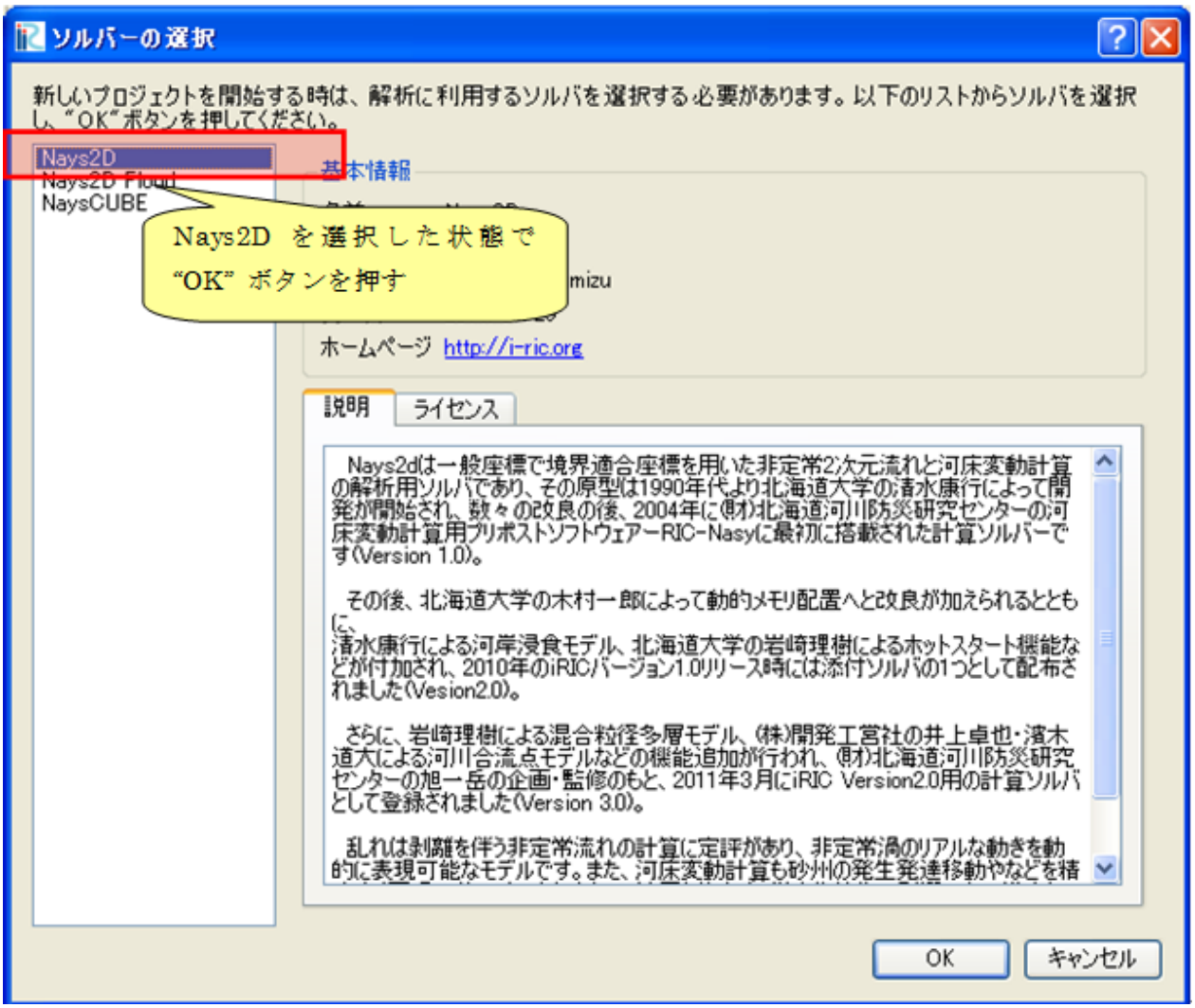

図 4.4　ソルバー選択ダイアログ

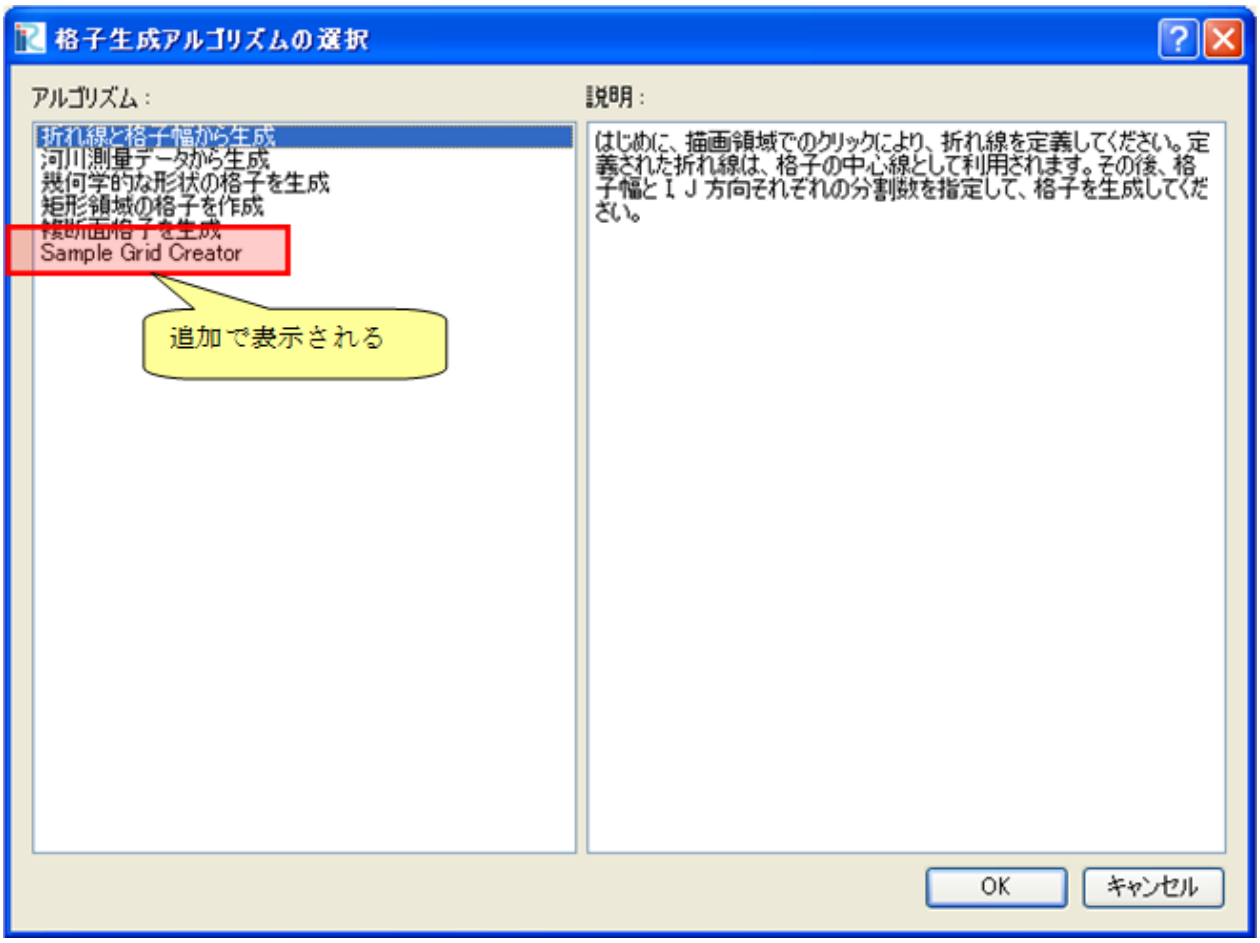

図 4.5　格子生成アルゴリズム選択ダイアログ

4.3.2 格子生成条件の定義

計算条件を定義します。計算条件は、ソルバー定義ファイルの CalculationCondition 要素で定義します。 基本情報の作成 (ページ 46) で作成した格子生成プログラム定義ファイルに追記し、 リスト 4.2 に示すようなファイルにし、保存します。追記した部分を強調して示しました。

リスト 4.2　格子生成条件を追記した格子生成プログラム定義ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<GridGeneratorDefinition
  name="samplecreator"
  caption="Sample Grid Creator"
  version="1.0"
  copyright="Example Company"
  executable="generator.exe"
  gridtype="structured2d"
>
  <GridGeneratingCondition>
```

```
    <Tab name="size" caption="Grid Size">
      <Item name="imax" caption="IMax">
        <Definition valueType="integer" default="10" max="10000" min="1" />
      </Item>
      <Item name="jmax" caption="JMax">
        <Definition valueType="integer" default="10" max="10000" min="1" />
      </Item>
    </Tab>
  </GridGeneratingCondition>
</GridGeneratorDefinition>
```

この時点では、格子生成プログラム定義ファイルの構造は図 4.6 に示すようになっています。

要素	備考
GridGeneratorDefinition	基本情報を登録。
GridGeneratingCondition	格子生成条件を定義。
Tab	格子生成条件のグループを定義。
Item	格子生成条件の名前を定義。
Definition	格子生成条件の属性を定義。
Item	格子生成条件の名前を定義。
Definition	格子生成条件の属性を定義。

図 4.6　格子生成プログラム定義ファイルの構造

正しく格子生成プログラム定義ファイルが作成できているか確認します。

iRIC を起動し、 基本情報の作成 (ページ 46) と同じ手順で格子生成アルゴリズム選択画面を表示します。「Sample Grid Creator」 を選択し、 「OK」 ボタンを押します。

すると、 図 4.7 に示すダイアログが表示されます。リスト 4.2 で追記した内容に従って、「Grid Size」 というグループが追加されているのが分かります。確認できたら、 「キャンセル」 ボタンを押します。

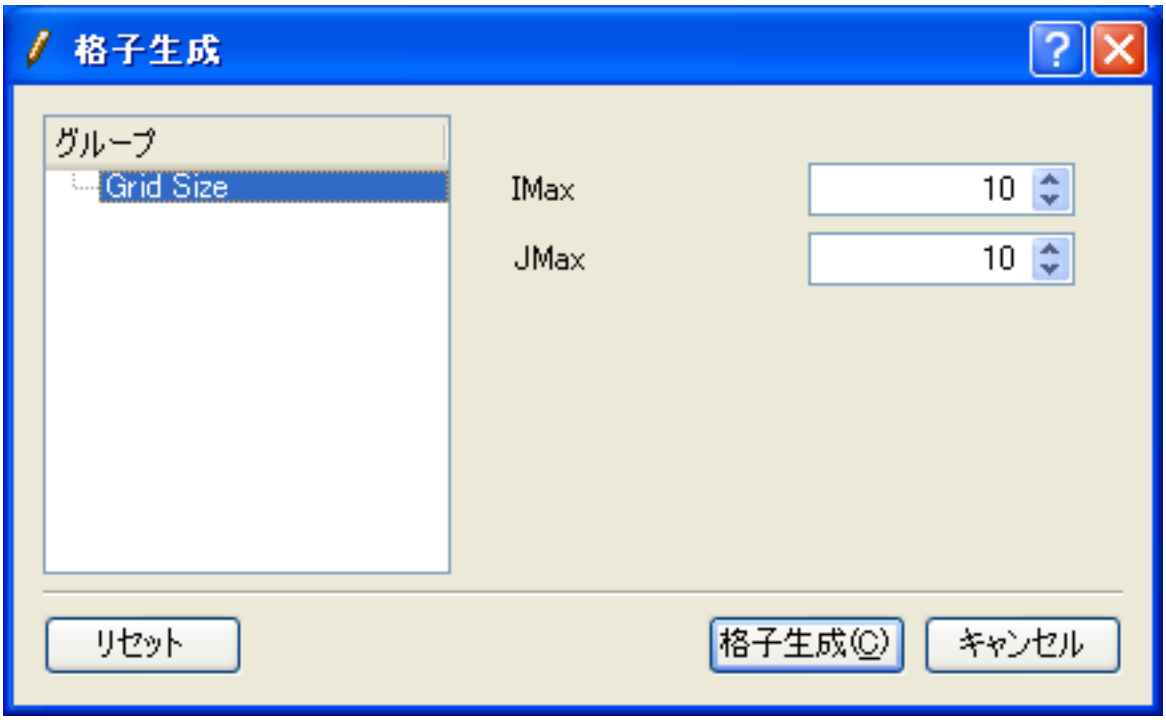

図 4.7　格子生成ダイアログ 表示例

グループを増やして、さらに格子生成条件を追加します。「Grid Size」の Tab 要素 のすぐ下に、「Elevation Output」というグループを追加して保存します。追記した定義ファイルの抜粋を、リスト 4.3 に示します。追記した部分を強調して示しました。

リスト 4.3　格子生成条件を追記した格子生成プログラム定義ファイルの例 (抜粋)

```
    （前略）
    </Tab>
    <Tab name="elevation" caption="Elevation Output">
      <Item name="elev_on" caption="Output">
        <Definition valueType="integer" default="0">
          <Enumeration caption="Enabled" value="1" />
          <Enumeration caption="Disabled" value="0" />
        </Definition>
      </Item>
      <Item name="elev_value" caption="Value">
        <Definition valueType="real" default="0">
          <Condition type="isEqual" target="elev_on" value="1" />
        </Definition>
      </Item>
    </Tab>
  </GridGeneratingCondition>
</GridGeneratorDefinition>
```

この時点では、定義ファイルの構造は図 4.8 に示す通りです。

要素	備考
GridGeneratorDefinition	基本情報を登録。
GridGeneratingCondition	格子生成条件を定義。
Tab	格子生成条件のグループを定義。
(略)	
Tab	格子生成条件のグループを定義。
Item	格子生成条件の名前を定義。
Definition	格子生成条件の属性を定義。
Enumeration	格子生成条件に指定できる選択肢を定義。
Enumeration	格子生成条件に指定できる選択肢を定義。
Item	格子生成条件の名前を定義。
Definition	格子生成条件の属性を定義。
Condition	この格子生成条件が有効になる条件を定義

図 4.8　格子生成プログラム定義ファイルの構造

正しくソルバー定義ファイルが作成できているか確認します。先ほどと同じ手順でダイアログを表示します。

「Elevation Output」というグループがリストに表示され、このグループには 2 つの項目が表示されているのが分かります。また、「Value」は、「Output」で「Enabled」を選択している時のみ有効です。ダイアログの表示例を図 4.9 に示します。

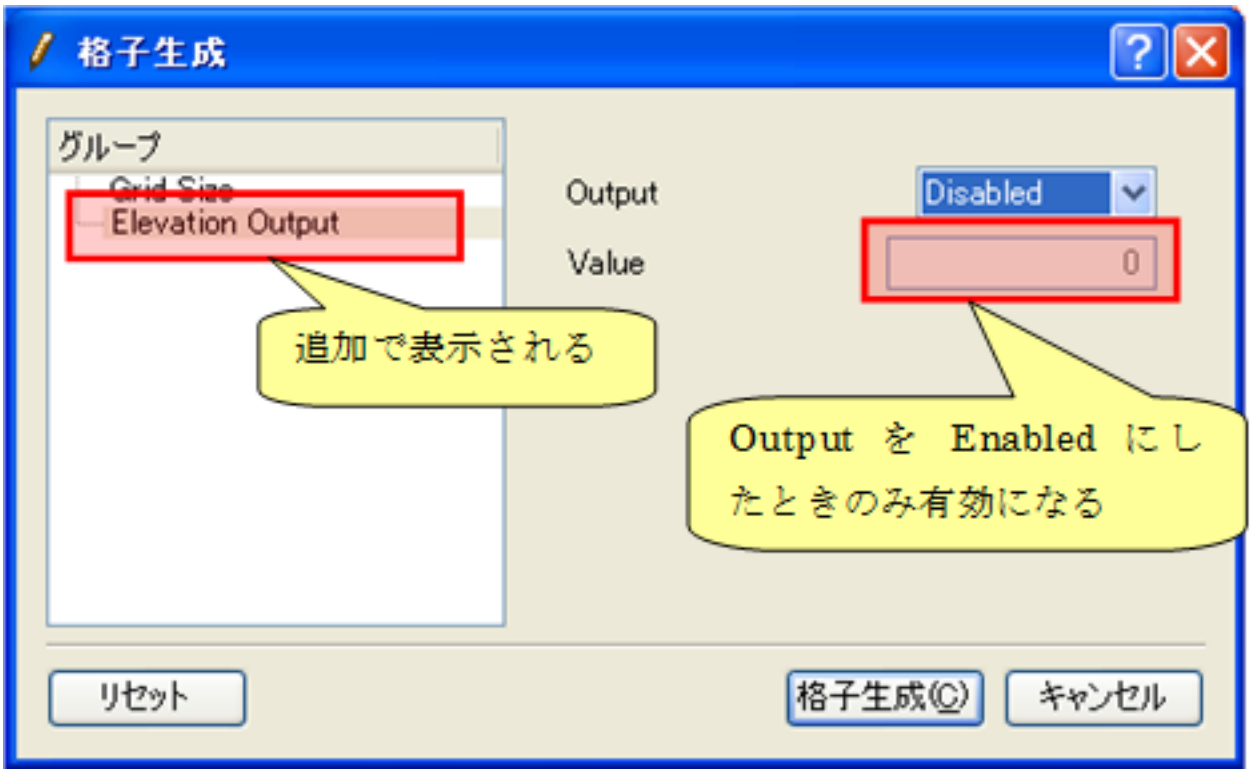

図 4.9　格子生成ダイアログ 表示例

格子生成条件の定義についてまとめると、以下の通りです。

- 格子生成条件のグループは Tab 要素で、格子生成条件は Item 要素でそれぞれ指定します。
- Definition 要素以下の構造は、計算条件の種類 (例: 整数、実数、整数からの選択、関数型) によって異なります。格子生成条件の種類ごとの記述方法と、ダイアログ上での表示については 5.3.1 を参照して下さい。
- 格子生成条件には、 Condition 要素で依存関係を定義できます。 Condition 要素では、その格子生成条件が有効になる条件を指定します。 Condition 要素の定義方法の詳細は、5.3.2 を参照して下さい。
- この例では、格子生成条件のダイアログを単純なリスト形式で作成しましたが、グループボックスを使うなどしてダイアログのレイアウトをカスタマイズすることができます。ダイアログのレイアウトのカスタマイズ方法については 5.3.3 を参照して下さい。

4.3.3 エラーコードの定義

格子生成プログラムで発生するエラーのコードと、対応するメッセージを定義します。エラーコードは、定義ファイルの ErrorCode 要素で定義します。格子生成条件の定義 (ページ 49) で作成した格子生成プログラム定義ファイルに追記し、 リスト 4.4 に示すようなファイルにし、保存します。追記した部分を強調して示しました。

リスト 4.4　エラーコードを追記した格子生成プログラム定義ファイルの例

```
(前略)
      </Item>
    </Tab>
  </GridGeneratingCondition>
  <ErrorCodes>
    <ErrorCode value="1" caption="IMax * JMax must be smaller than 100,000." />
  </ErrorCodes>
</GridGeneratorDefinition>
```

この時点では、定義ファイルの構造は 図 4.10 に示すようになっています。なお、エラーコードの定義は必須ではありません。

要素	備考
GridGeneratorDefinition	基本情報を登録。
GridGeneratingCondition	格子生成条件を定義。
(略)	
ErrorCodes	
ErrorCode	エラーコードを定義。

図 4.10　格子生成プログラム定義ファイルの構造

エラーコードの定義が正しく行えているかの確認は、格子生成プログラムを作成するまで行えません。エラーコードの定義の確認についてはエラー処理の記述 (ページ 61) で行います。

4.4 格子生成プログラムの作成

格子生成プログラムを作成します。この例では、格子生成プログラムは FORTRAN 言語で開発します。

iRIC と連携する格子生成プログラムを開発するには、格子生成プログラム定義ファイルに従って iRIC が生成する格子生成データファイルを、格子生成条件、格子の入出力に利用する必要があります。

iRIC が生成する格子生成データファイルは、CGNS ファイルという形式です。CGNS ファイルの入出力には、iRIClib というライブラリを使用します。

この節では、格子生成プログラム定義ファイルの作成 (ページ 45) で作成した定義ファイルに従って iRIC が生成する格子生成データファイルを読みこんで、格子を生成するプログラムを開発する流れを説明します。

この格子生成プログラムで行われる入出力処理を 表 4.3 に示します。

表 4.3　格子生成プログラムの入出力の処理の流れ

処理の内容
格子生成データファイルを開く
内部変数の初期化
格子生成条件の読み込み
格子の出力
格子生成データファイルを閉じる

この節では、格子生成プログラムを以下の手順で作成します。

1. 骨組みの作成
2. 格子生成データファイルを開く処理、閉じる処理の記述
3. 格子の出力処理の記述
4. 格子生成条件の読み込み処理の記述
5. エラー処理の記述

4.4.1 骨組みの作成

格子生成プログラムの骨組みを作成します。リスト 4.5 に示すソースコードを作成して、sample.f90 という名前で保存します。この時点では、このプログラムは何もしていません。

このソースコードをコンパイルします。コンパイル方法は、コンパイラによって異なります。gfortran, Intel Fortran Compiler でのコンパイル方法を 7.2 で解説していますので、参考にしてください。

リスト 4.5　サンプル格子生成プログラム ソースコード

```
1 program SampleProgram
2   implicit none
3   include 'cgnslib_f.h'
4 end program SampleProgram
```

コンパイルが成功することを確認してください。

4.4.2 格子生成データファイルを開く処理、閉じる処理の記述

格子生成データファイルを開く処理、閉じる処理を記述します。

格子計算プログラムは、処理開始時に格子生成データファイルを開き、終了時に閉じる必要があります。iRIC は引数として格子生成データファイルのファイル名を渡すため、そのファイル名を開きます。

引数の数と引数を取得するための方法は、コンパイラによって異なります。gfortran, Intel Fortran compiler での引数の取得方法を 7.1 で説明していますので、参考にしてください。ここでは、Intel Fortran compiler でコンパイルする場合の方法で記述します。

処理を追記したソースコードを リスト 4.6 に示します。追記した部分を強調して示します。

リスト 4.6　計算データファイルを開く処理、閉じる処理を追記したソースコード

```
program SampleProgram
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier
  integer:: icount, istatus

  character(200)::condFile

  icount = nargs()
  if ( icount.eq.2 ) then
    call getarg(1, condFile, istatus)
  else
    stop "Input File not specified."
  endif

  ! 格子生成データファイルを開く
  call cg_open_f(condFile, CG_MODE_MODIFY, fin, ier)
  if (ier /=0) stop "*** Open error of CGNS file ***"

  ! 内部変数の初期化。戻り値は 1 になるが問題ない。
  call cg_iric_init_f(fin, ier)

  ! 格子生成データファイルを閉じる
  call cg_close_f(fin, ier)
end program SampleProgram
```

骨組みの作成 (ページ 55) と同様に、ファイルのコンパイルを行います。問題なくコンパイルが成功することを確認してください。

この節で追加した関数の詳細については、6.3.2, 6.3.3, 6.3.15 を参照してください。

4.4.3 格子の出力処理の記述

格子の出力処理を記述します。

まずは、iRIC との連携が正しく行えることを確認するため、単純な格子を生成して出力する処理を記述します。

格子を出力する処理を追記したソースコードを リスト 4.7 に示します。追記した部分を強調して示します。

リスト 4.7　格子を出力する処理を追記したソースコード

```
program SampleProgram
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier
  integer:: icount, istatus
  integer:: imax, jmax
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  character(200)::condFile

  icount = nargs()
  if ( icount.eq.2 ) then
    call getarg(1, condFile, istatus)
  else
    stop "Input File not specified."
  endif

  ! 格子生成データファイルを開く
  call cg_open_f(condFile, CG_MODE_MODIFY, fin, ier)
  if (ier /=0) stop "*** Open error of CGNS file ***"

  ! 内部変数の初期化。戻り値は 1 になるが問題ない。
  call cg_iric_init_f(fin, ier)

  imax = 10
  jmax = 10

  ! 格子生成用のメモリを確保
  allocate(grid_x(imax,jmax), grid_y(imax,jmax)

  ! 格子を生成
  do i = 1, imax
    do j = 1, jmax
      grid_x(i, j) = i
      grid_y(i, j) = j
    end do
  end do

  ! 格子を出力
  cg_iric_writegridcoord2d_f(imax, jmax, grid_x, grid_y, ier)

  ! 格子生成データファイルを閉じる
  call cg_close_f(fin, ier)
end program SampleProgram
```

コンパイルしたら、できた実行プログラムを フォルダの作成 (ページ 45) で作成したフォルダにコピーし、名前を 基本情報の作成 (ページ 46) で executable 属性に指定した名前 (この例なら 「generator.exe」) に変更してくださ

い。またこの時、格子生成プログラムの実行に必要な DLL なども同じフォルダにコピーしてください。

この段階で、iRIC から格子生成プログラムが正しく起動できるか確認します。

ソルバーに 「Nays2DH」 を指定して、新しいプロジェクトを開始し、基本情報の作成 (ページ 46) で行ったのと同じ操作で格子生成アルゴリズムに 「Sample Grid Creator」を選択し、格子生成ダイアログを表示します。表示されるダイアログを図 4.11 に示します。

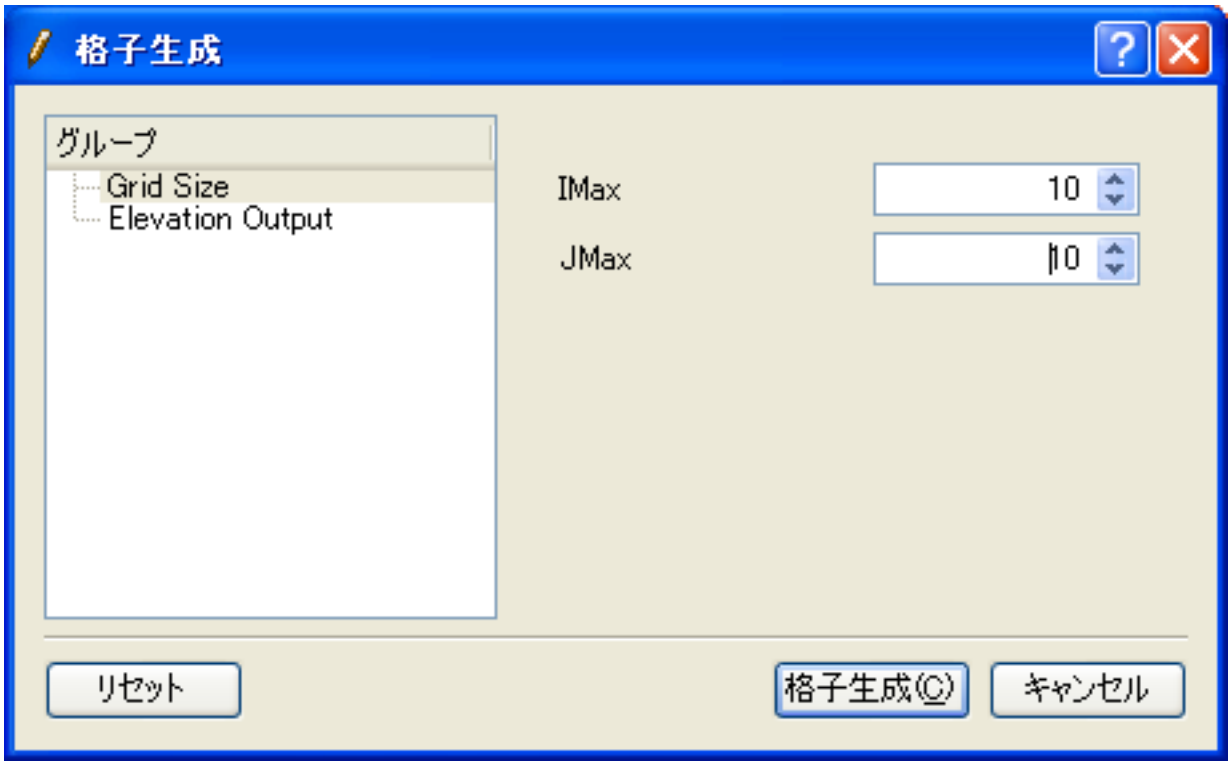

図 4.11　格子生成条件設定ダイアログ 表示例

「格子生成」 ボタンを押します。すると、格子生成プログラムが 10 x 10 の格子を生成し、それが iRIC 上に読み込まれるのが確認できます。「格子生成」 ボタンを押した後のプリプロセッサの表示画面を図 4.12 に示します。

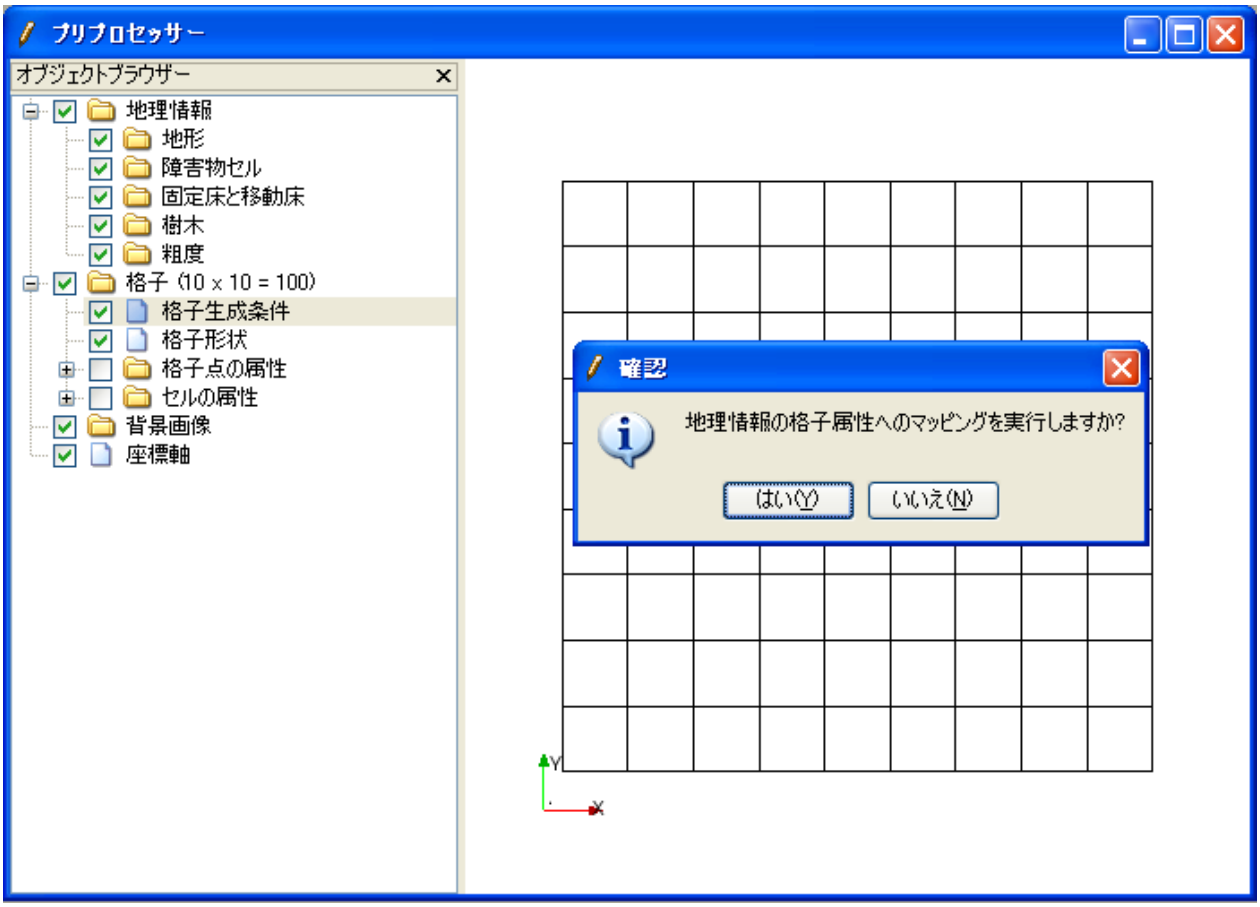

図 4.12　プリプロセッサ表示例

なお、この節で追加した格子出力用の関数の詳細については、6.3.8 を参照してください。ただし 6.3.8 では 3 次元格子の出力用関数についても解説していますが、格子生成プログラムで利用できるのは、2 次元格子の出力用関数だけです。

4.4.4 格子生成条件の読み込み処理の記述

格子生成条件の読み込み処理を記述します。

iRIC は、 格子生成プログラム定義ファイルの作成 (ページ 45) で作成した定義ファイルに従って格子生成条件を格子生成データファイルに出力しますので、それに対応するように格子生成条件の読み込み処理を記述します。

格子生成条件の読み込み処理を追記したソースコードをリスト 4.8 に示します。追記した部分を太字で示します。格子生成条件を読み込む関数に渡す引数が、 格子生成条件の定義 (ページ 49) で定義ファイルに記述した Item 要素の name 属性と一致していることに注目してください。

コンパイルしたら、 格子の出力処理の記述 (ページ 56) の時と同様の手順で格子を生成し、指定した通りの格子生成条件で格子が生成することを確認してください。

定義ファイルで定義する格子生成条件と、それを読み込むための iRIClib の関数の対応関係については、5.3.1 を参照してください。格子生成条件の読み込みに使う関数の詳細については、6.3.5 を参照してください。

リスト 4.8　格子生成条件の読み込み処理を追記したソースコード

```
program SampleProgram
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier
  integer:: icount, istatus
  integer:: imax, jmax
  integer:: elev_on
  double precision:: elev_value
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  double precision, dimension(:,:), elevation

  character(200)::condFile

  icount = nargs()
  if ( icount.eq.2 ) then
    call getarg(1, condFile, istatus)
  else
    stop "Input File not specified."
  endif

  ! 格子生成データファイルを開く
  call cg_open_f(condFile, CG_MODE_MODIFY, fin, ier)
  if (ier /=0) stop "*** Open error of CGNS file ***"

  ! 内部変数の初期化。戻り値は 1 になるが問題ない。
  call cg_iric_init_f(fin, ier)

  ! 格子生成条件の読み込み
  ! 簡潔に記述するため、エラー処理は行っていない
  call cg_iric_read_integer_f("imax", imax, ier)
  call cg_iric_read_integer_f("jmax", jmax, ier)
  call cg_iric_read_integer_f("elev_on", elev_on, ier)
  call cg_iric_read_real_f("elev_value", elev_value, ier)

  ! 格子生成用のメモリを確保
  allocate(grid_x(imax,jmax), grid_y(imax,jmax)
  allocate(elevation(imax,jmax))

  ! 格子を生成
  do i = 1, isize
    do j = 1, jsize
      grid_x(i, j) = i
      grid_y(i, j) = j
      elevation(i, j) = elev_value
```

```
    end do
  end do

  ! 格子を出力
  cg_iric_writegridcoord2d_f(imax, jmax, grid_x, grid_y, ier)
  if (elev_on == 1) then
    cg_iric_write_grid_real_node_f("Elevation", elevation, ier);
  end if

  ! 格子生成データファイルを閉じる
  call cg_close_f(fin, ier)
end program SampleProgram
```

4.4.5 エラー処理の記述

格子生成条件に問題があった場合のエラー処理を記述します。

エラー処理を追記したソースコードを リスト 4.9 に示します。太字で示したのが追記した部分です。追記した部分により、格子の格子点数が 100000 を超えるような imax, jmax を指定した時は、エラーが発生するようにしました。

コンパイルしたら、 格子の出力処理の記述 (ページ 56) の時と同様の手順で格子を生成し、imax x jmax が 100000 より大きくなる条件の時には、 図 4.13 に示すようなダイアログが表示されることを確認してください。例えば、IMax, JMax にそれぞれ 10000 を指定してみてください。

エラー処理に使う関数の詳細については 6.3.14 を参照してください。

リスト 4.9　エラー処理を追記したソースコード (抜粋)

```
! (前略)

  ! 格子生成条件の読み込み
  ! 簡潔に記述するため、エラー処理は行っていない
  call cg_iric_read_integer_f("imax", imax, ier)
  call cg_iric_read_integer_f("jmax", jmax, ier)
  call cg_iric_read_integer_f("elev_on", elev_on, ier)
  call cg_iric_read_real_f("elev_value", elev_value, ier)

  ! エラー処理
  if (imax * jmax > 100000 ) then
    ! 100000 より大きい格子は生成できない
    call cg_iric_write_errorcode(1, ier)
    cg_close_f(fin, ier)
    stop
  endif

  ! 格子生成用のメモリを確保
```

```
19   allocate(grid_x(imax,jmax), grid_y(imax,jmax)
20   allocate(elevation(imax,jmax))
21
22 ! （後略）
```

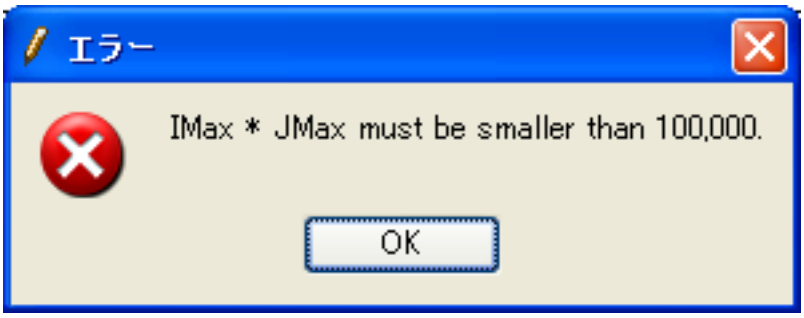

図 4.13　格子生成エラーダイアログ 表示例

4.5 格子生成プログラム定義ファイルの辞書ファイルの作成

格子生成プログラム定義ファイルで用いられている文字列のうち、ダイアログ上に表示される文字列を翻訳して表示するための辞書ファイルを作成します。

まず、iRIC から、以下のメニューを起動します。すると、格子生成プログラム定義ファイルの辞書更新ウィザードが表示されます。ダイアログの表示例を、図 4.14 〜 図 4.16 に示します。

メニュー: オプション (O) –> 辞書ファイルの作成・更新 (C)

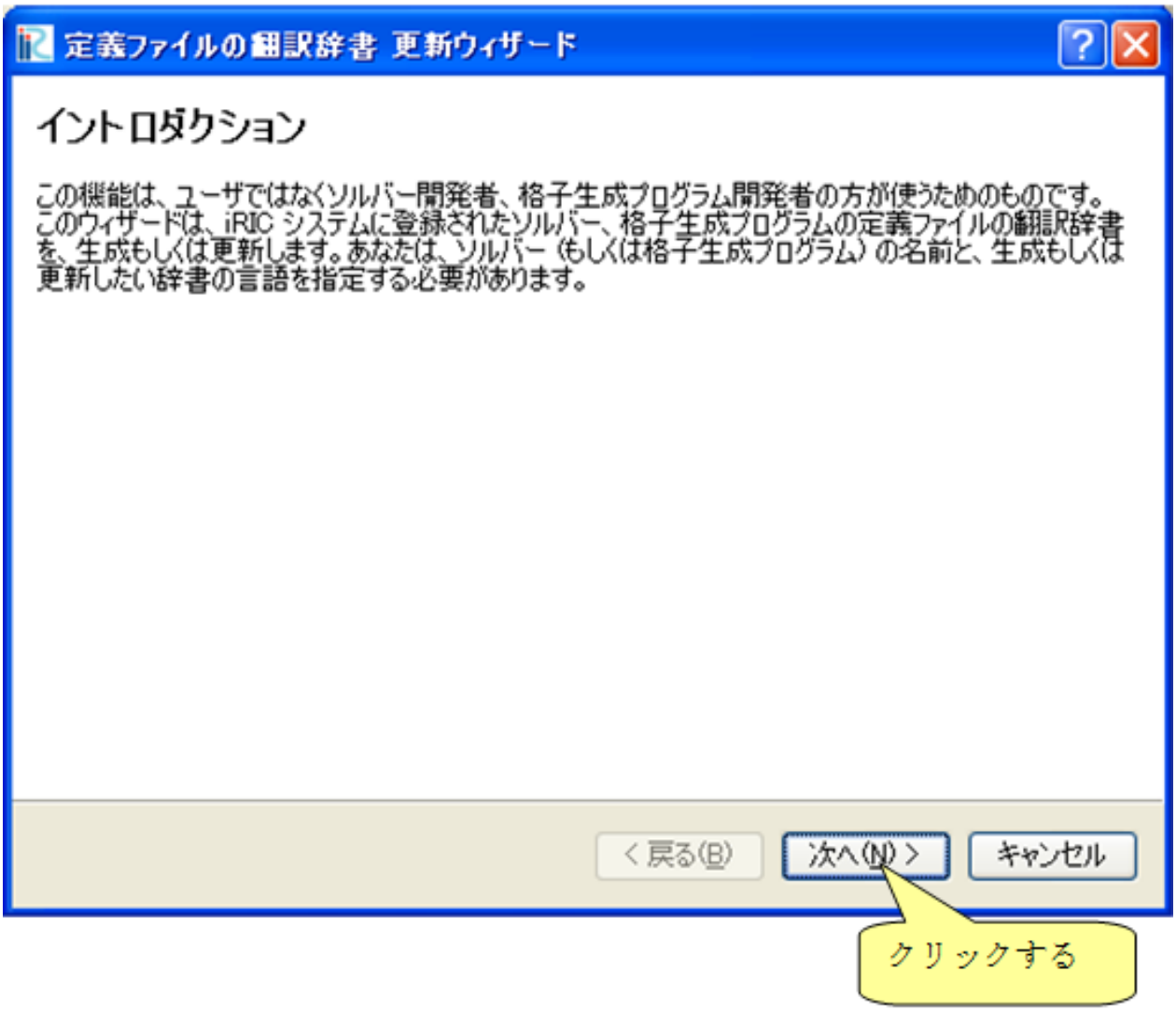

図 4.14　定義ファイルの翻訳辞書 更新ウィザード 表示例 (1 ページ目)

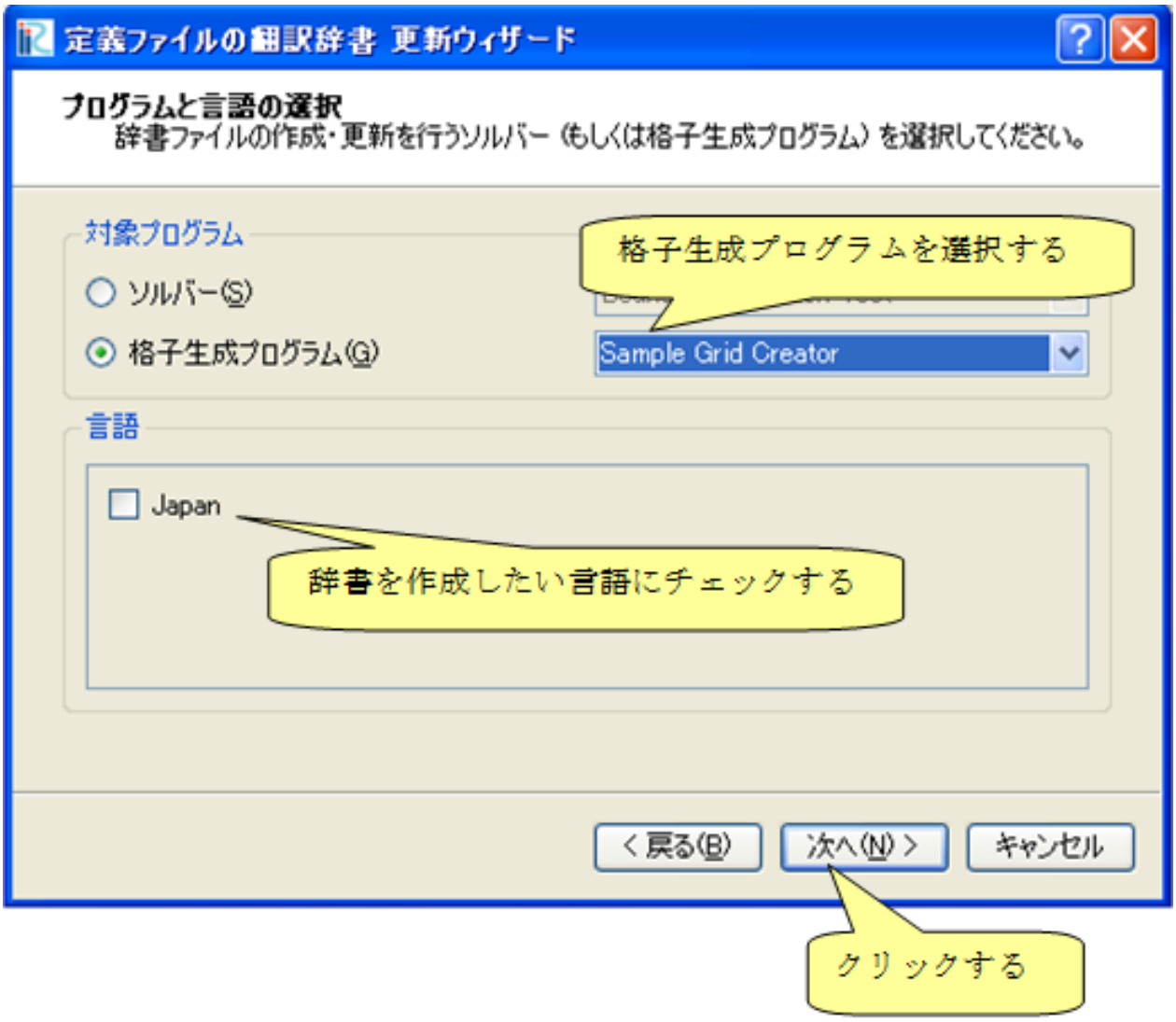

図 4.15　定義ファイルの翻訳辞書 更新ウィザード 表示例 (2 ページ目)

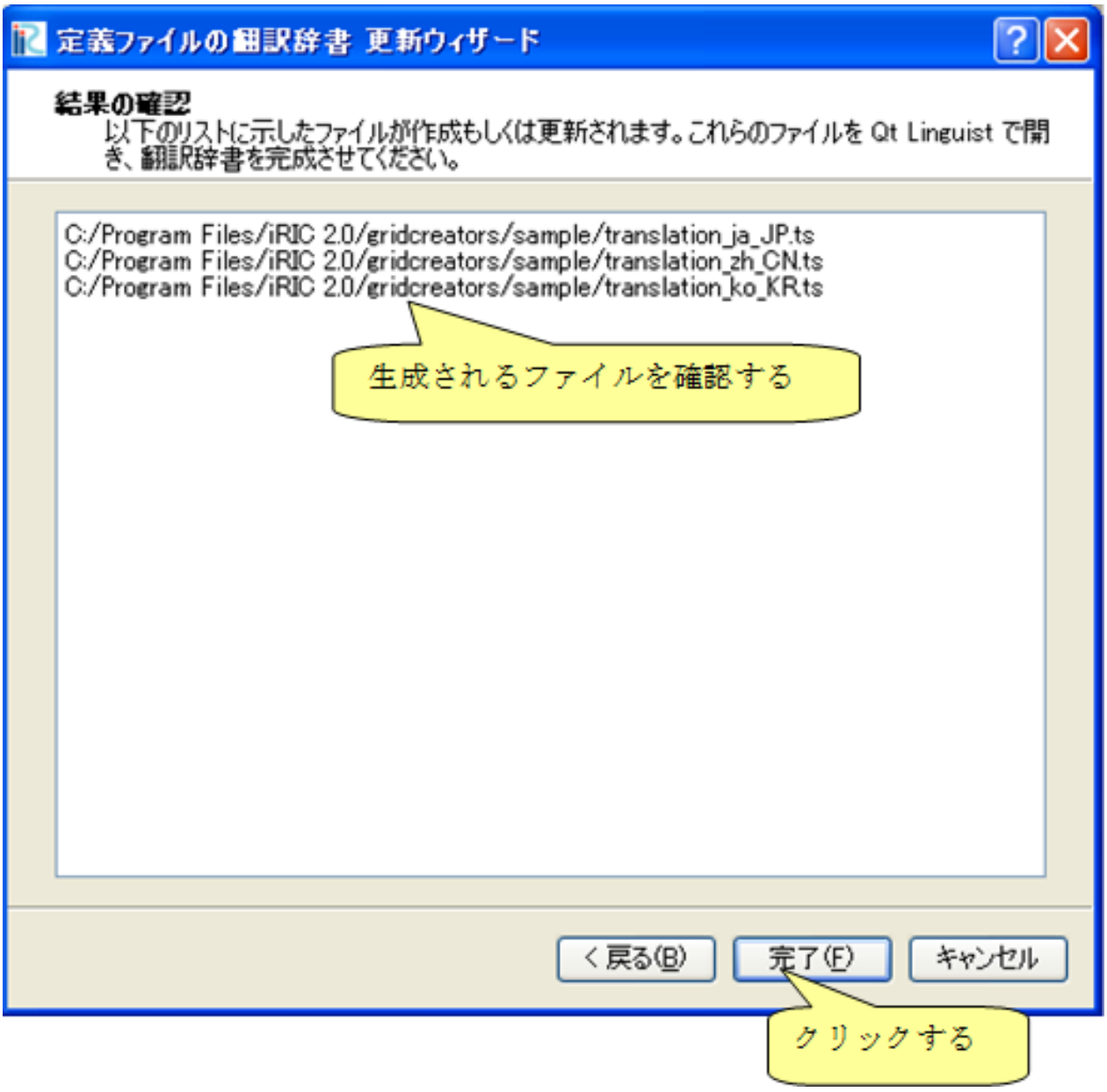

図 4.16　定義ファイルの翻訳辞書 更新ウィザード 表示例 (3 ページ目)

辞書ファイルは、格子生成プログラムソルバー定義ファイルと同じフォルダに作成されます。作成された辞書ファイルは、翻訳前の英語のみが含まれています。辞書ファイルはテキストファイルですので、テキストエディタなどで開いて編集します。辞書ファイルは、UTF-8 で保存してください。

辞書ファイルの編集例を、 リスト 4.10 リスト 4.11 に示します。例に示したように、translation 要素の中に翻訳後の文字列を追記してください。

リスト 4.10　格子生成プログラム定義ファイルの辞書ファイルの一部 (編集前)

```
<message>
  <source>Sample Grid Creator</source>
  <translation></translation>
</message>
```

リスト 4.11　格子生成プログラム定義ファイルの辞書ファイルの一部 (編集後)

```
<message>
  <source>Sample Grid Creator</source>
  <translation>サンプル格子生成プログラム</translation>
</message>
```

なお、辞書ファイルは、Qt に付属する Qt Linguist を利用して編集することもできます。Qt Linguist の画面表示例を図 4.17 に示します。Qt Linguist は、以下の URL からダウンロードできる Qt に含まれています。

https://www.qt.io/download/

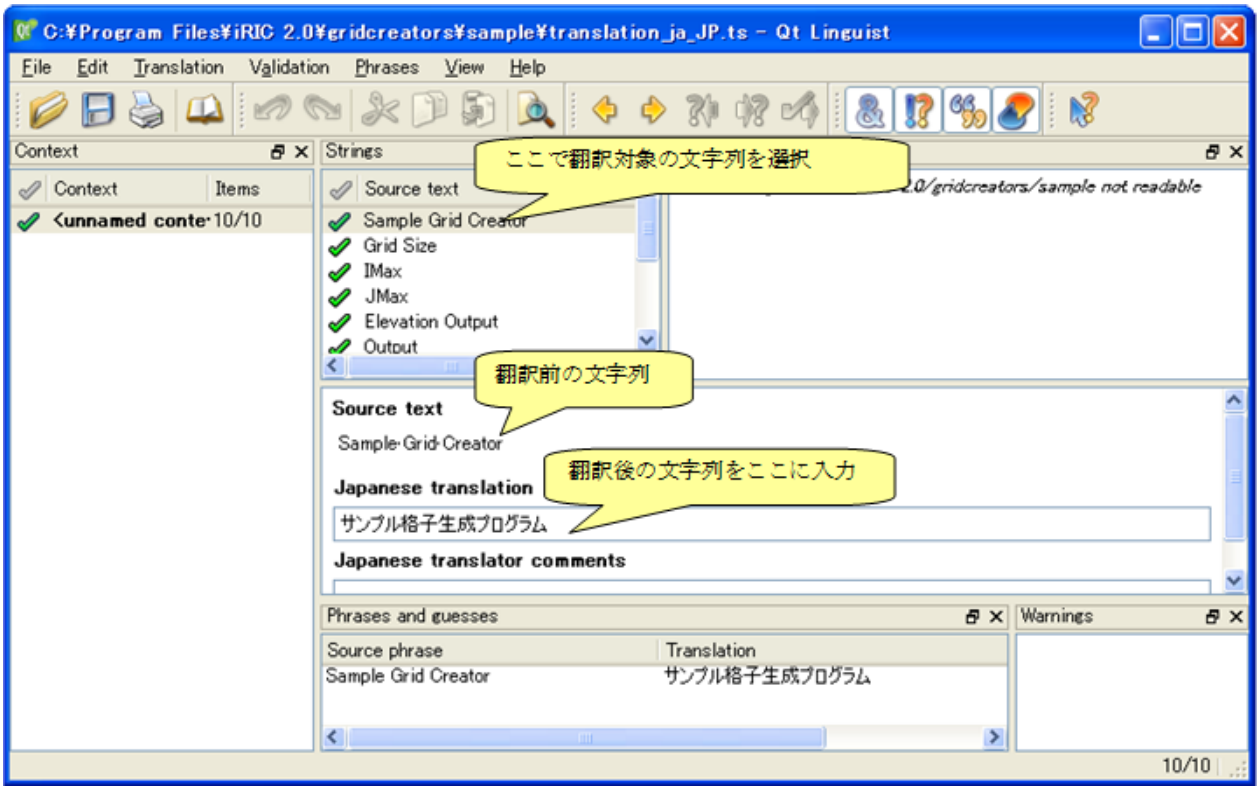

図 4.17　Qt Linguist 画面表示例

翻訳が完了したら、iRIC を確認したい言語に切り替えてから iRIC を起動し直し、正しく翻訳されて表示されるか確認します。翻訳完了後の格子生成条件設定ダイアログの表示例を図 4.18 に示します。

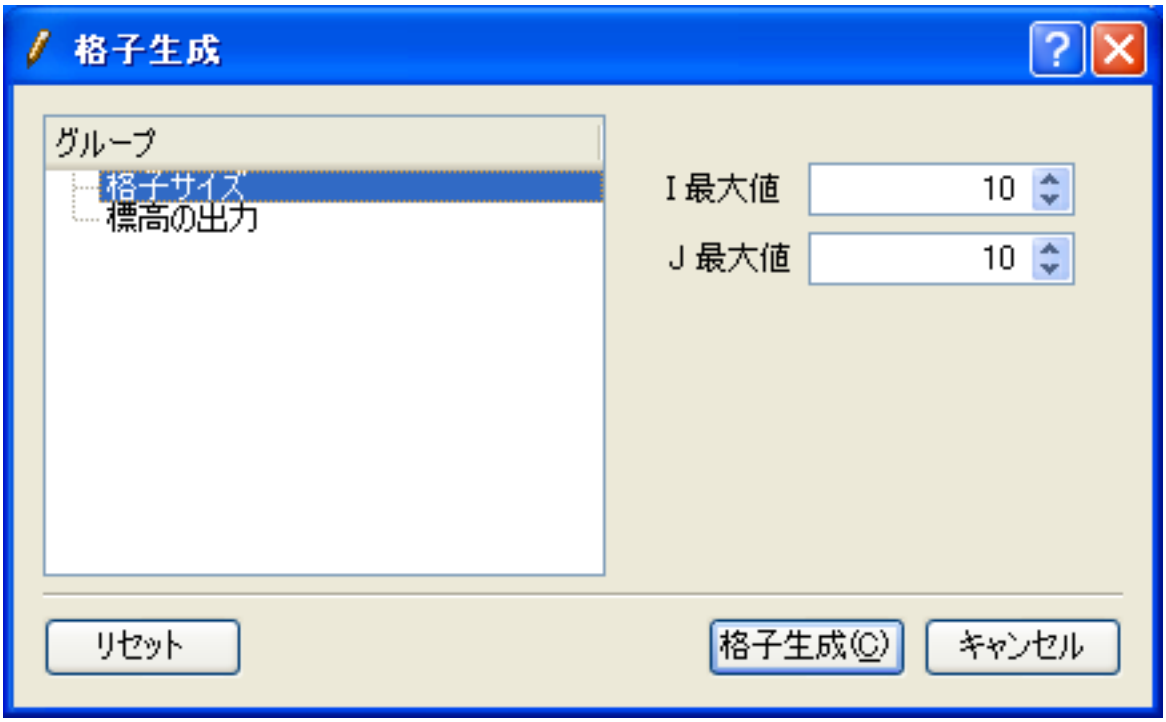

図 4.18　翻訳完了後の格子生成条件設定ダイアログ 表示例

4.6 説明ファイルの作成

格子生成プログラムの概要について説明するファイルを作成します。

README というファイル名のテキストファイルを、フォルダの作成 (ページ 45) で作成したフォルダの下に作成します。文字コードは UTF-8 にします。

なお、説明ファイルは、以下の例のようなファイル名で言語ごとに用意します。言語ごとの説明ファイルがない場合、 README が使用されます。

- 英語: README
- 日本語: README_ja_JP

「README_」 以降につく文字列は、辞書ファイルの 「translation_*****.ts」 の「*****」 の部分と同じですので、日本語以外の説明ファイルを作る際のファイル名は、辞書ファイルのファイル名を参考にして決めて下さい。

説明ファイルの内容は、格子生成アルゴリズム選択ダイアログで、説明欄に表示されます。ファイルを作成したら、iRIC 上で正しく表示されるか確認して下さい。ダイアログの表示例を、図 4.19 に示します。

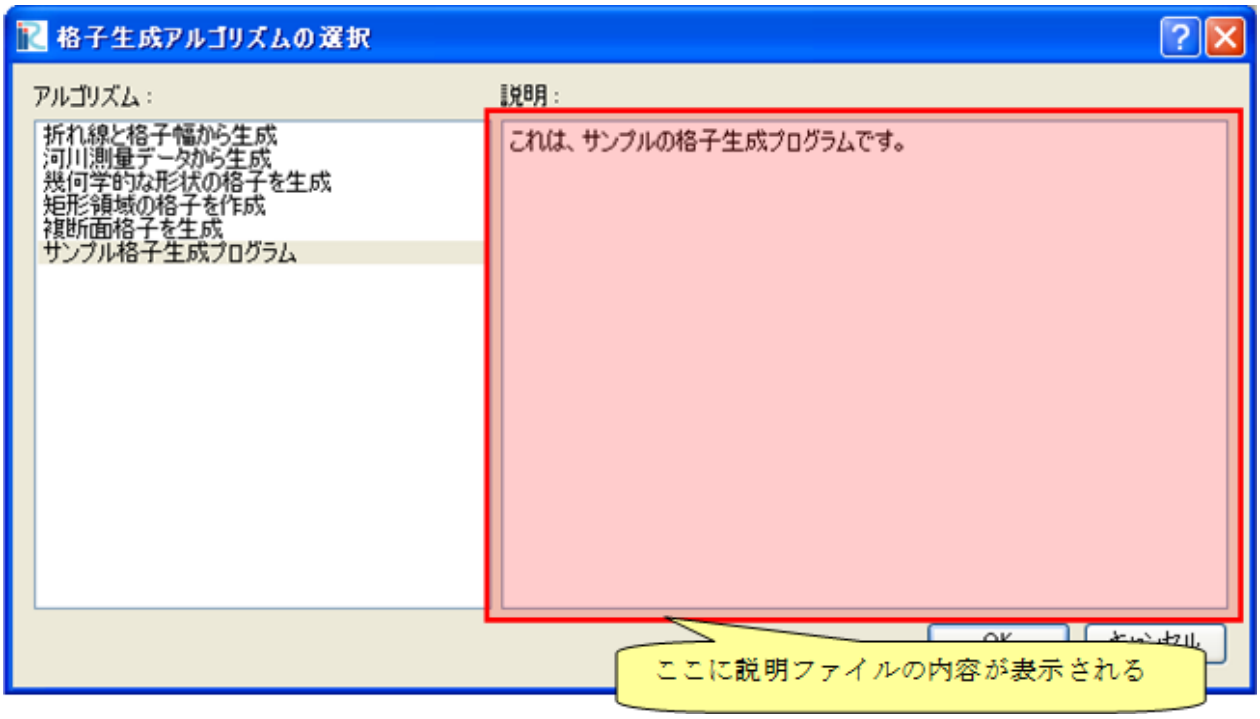

図 4.19　格子生成アルゴリズム選択ダイアログ 表示例

第 5 章

定義ファイル (XML) について

5.1 概要

iRIC は、ソルバー定義ファイル、格子生成プログラム定義ファイルを読み込むことで、そのソルバー、格子生成プログラムが必要な入力情報を作成するためのインターフェースを提供します。

5.2 構造

ソルバー定義ファイル、格子生成プログラム定義ファイルの構造を示します。

5.2.1 ソルバー定義ファイル

計算格子を 1 つ利用するソルバーでのソルバー定義ファイルの構造を図 5.1 に、複数利用するソルバーでのソルバー定義ファイルの構造を図 5.2 にそれぞれ示します。

要素	説明	必須
SolverDefinition	基本情報	○
CalculationCondition	計算条件	○
Tab	計算条件のグループ	
Item	計算条件の要素	
Definition	計算条件の定義	
Condition	計算条件が有効になる条件	
Item	計算条件	
Definition		
Condition		
...		
Tab		
...		
...		
GridRelatedCondition	格子属性	○
Item		
Item		
BoundaryCondition	境界条件	
Item	境界条件の要素	
Definition	境界条件の定義	
Condition	境界条件が有効になる条件	
Item		
Definition		
Condition		

図 5.1　ソルバー定義ファイルの構造

要素	説明	必須
SolverDefinition	基本情報	○
CalculationCondition	計算条件	○
Tab	計算条件のグループ	
Item	計算条件の要素	
Definition	計算条件の定義	
Condition	計算条件が有効になる条件	
Item	計算条件	
Definition		
Condition		
...		
Tab		
...		
...		
GridTypes		
GridType	格子種類の名前と構造	○
GridRelatedCondition	格子属性	○
Item		
...		
BoundaryCondition	境界条件	
Item	境界条件の要素	
Definition	境界条件の定義	
Condition	境界条件が有効になる条件	
...		
GridType	格子種類の名前と構造	○
GridRelatedCondition	格子属性	○
...		
BoundaryCondition	境界条件	
...		

図 5.2　複数の格子を利用するソルバーのソルバー定義ファイルの構造

複数の格子を利用するソルバーの場合、ソルバー定義ファイルでは GridType 要素を使って、それぞれの格子の構造、格子属性、境界条件を定義します。

複数の格子を利用するソルバーのソルバー定義ファイルの例を、リスト 5.1 に示します。この例では、境界条件は省略されています。以下の点が、1 つの格子を利用する場合と異なっていることに注意して下さい。

- 格子の構造 (gridtype 属性) は、SolverDefinition 要素でなく、 GridType 要素で定義されている。

リスト 5.1 に示したソルバー定義ファイルのソルバーを選択して iRIC で新しいプロジェクトを開始した場合、 図 5.3 に示すようなプリプロセッサが表示されます。

リスト 5.1　複数の格子を使用するソルバーのソルバー定義ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<SolverDefinition
  name="multigridsolver"
  caption="Multi Grid Solver"
  version="1.0"
  copyright="Example Company"
  release="2012.04.01"
  homepage="http://example.com/"
  executable="solver.exe"
  iterationtype="time"
>
  <CalculationCondition>
    <!-- ここで、計算条件を定義。-->
  </CalculationCondition>
  <GridTypes>
    <GridType name="river" caption="River">
      <GridRelatedCondition>
        <Item name="Elevation" caption="Elevation">
          <Definition valueType="real" position="node" />
        </Item>
        <Item name="Roughness" caption="Roughness">
          <Definition valueType="real" position="node"/>
        </Item>
        <Item name="Obstacle" caption=" Obstacle">
          <Definition valueType="integer" position="cell"/>
        </Item>
      </GridRelatedCondition>
    </GridType>
    <GridType name="floodbed" caption="Flood Bed">
      <GridRelatedCondition>
        <Item name="Elevation" caption="Elevation">
          <Definition valueType="real" position="node" />
        </Item>
      </GridRelatedCondition>
    </GridType>
  </GridTypes>
</SolverDefinition>
```

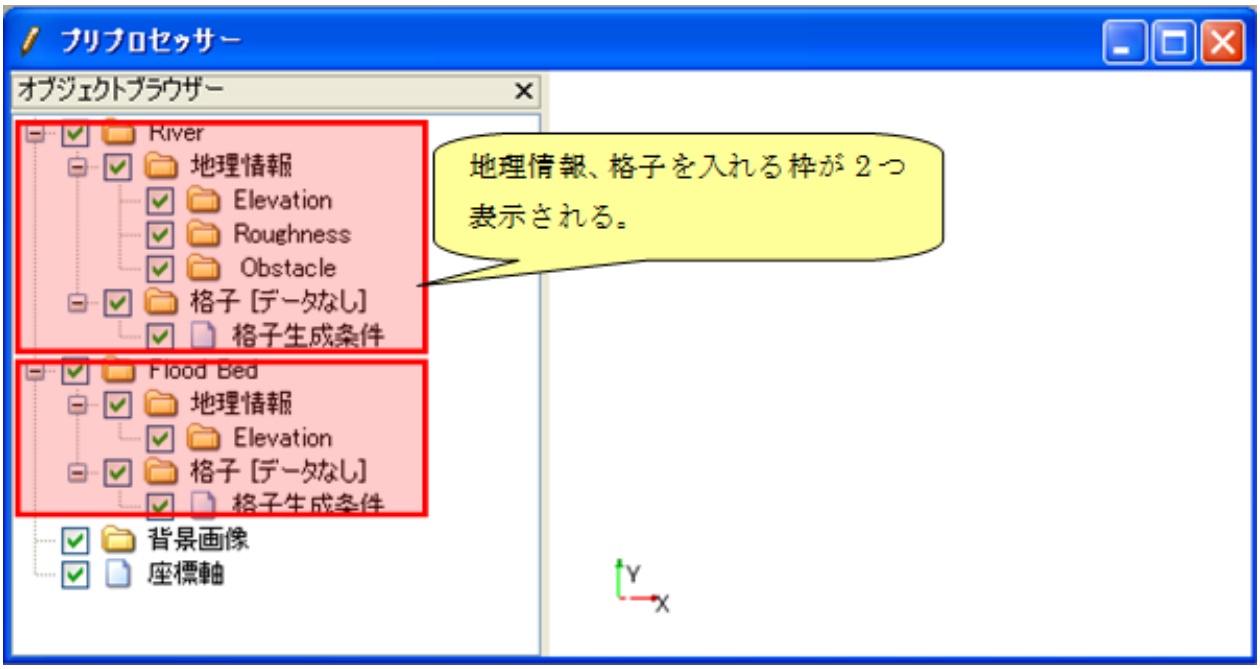

図 5.3　複数の格子を定義したソルバ定義ファイルを読み込んだ場合のプリプロセッサ 表示例

5.2.2 格子生成プログラム定義ファイル

格子生成プログラム定義ファイルの構造を、 図 5.4 に示します。

要素	説明	必須
GridGeneratorDefinition	基本情報	○
GridGeneratingCondition	格子生成条件	○
Tab	格子生成条件のグループ	
Item	格子生成条件の要素	
Definition	格子生成条件の定義	
Condition	格子生成条件が有効になる条件	
Item	格子生成条件	
Definition		
Condition		
...		
Tab		
...		
...		

図 5.4　格子生成プログラム定義ファイルの構造

5.3 定義例

5.3.1 計算条件・境界条件・格子生成条件の項目の定義と読み込み処理の例

ソルバー定義ファイルでの計算条件、格子生成プログラムでの格子生成条件の項目の定義例を示します。定義する位置は表 5.1 に示すように異なりますが、同じ文法で定義できます。各対象ファイルの構造は:ref: *def_structure* を参照してください。

表 5.1　要素の定義位置

項目	対象ファイル	定義する位置
計算条件	ソルバー定義ファイル	CalculationCondition 要素の下
格子生成条件	格子生成プログラム定義ファイル	GridGeneratingCondition 要素の下

定義できる項目の種類を、 表 5.2 に示します。この節では、以下を示します。

- 定義例
- iRIC の計算条件編集ダイアログ上での表示例
- ソルバー (もしくは格子生成プログラム) で値を読み込むための処理の記述例

ソルバー (もしくは格子生成プログラム) で値を読み込むための処理の記述例では、iRIClib の関数を使用しています。iRIClib の詳細は、6 章を参照して下さい。

記述例は読み込みに関連する部分のみですので、プログラム全体の例は 2.3.4, 4.4 を参照してください。

表 5.2　計算条件、格子生成条件の項目の種類

種類	説明	定義方法
文字列	文字列の値を入力。	valueType に 「string」 を指定
ファイル名 (読み込み用)	読み込み用のファイル名を入力。既に存在するファイルしか選択できない。	valueType に 「filename」 を指定
ファイル名 (書き込み用)	書き込み用のファイル名を入力。存在しないファイルの名前も指定できる。	valueType に 「filename_all」 を指定
フォルダ名	フォルダ名を入力。	valueType に 「foldername」 を指定
整数	任意の整数値を入力。	valueType に 「integer」 を指定
整数 (選択式)	あらかじめ用意した選択肢の中から整数値を選択。	valueType に 「integer」 を指定し、Enumeration 要素で選択肢を定義
実数	任意の実数値を入力。	valueType に 「real」 を指定
関数型	(X, Y) の組を複数入力。	valueType に 「functional」 を指定し、Parameter 要素、 Value 要素で変数と値を定義
関数型 (複数の値)	(X, Y1, Y2) の組を複数入力。	valueType に 「functional」 を指定し、Parameter 要素を 1 つと Value 要素を 2 つ定義

文字列

リスト 5.2　文字列の条件の定義例

```
<Item name="sampleitem" caption="Sample Item">
  <Definition valueType="string" />
</Item>
```

図 5.5　文字列の条件の表示例

リスト 5.3　文字列の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier
character(200):: sampleitem

call cg_iric_read_string_f("sampleitem", sampleitem, ier)
```

リスト 5.4　文字列の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier
character(200):: sampleitem

call cg_iric_read_bc_string_f("inflow", 1, "sampleitem", sampleitem, ier)
```

ファイル名 (読み込み用)

リスト 5.5　ファイル名 (読み込み用) の条件の定義例

```
<Item name="flowdatafile" caption="Flow data file">
  <Definition valueType="filename" default="flow.dat" />
</Item>
```

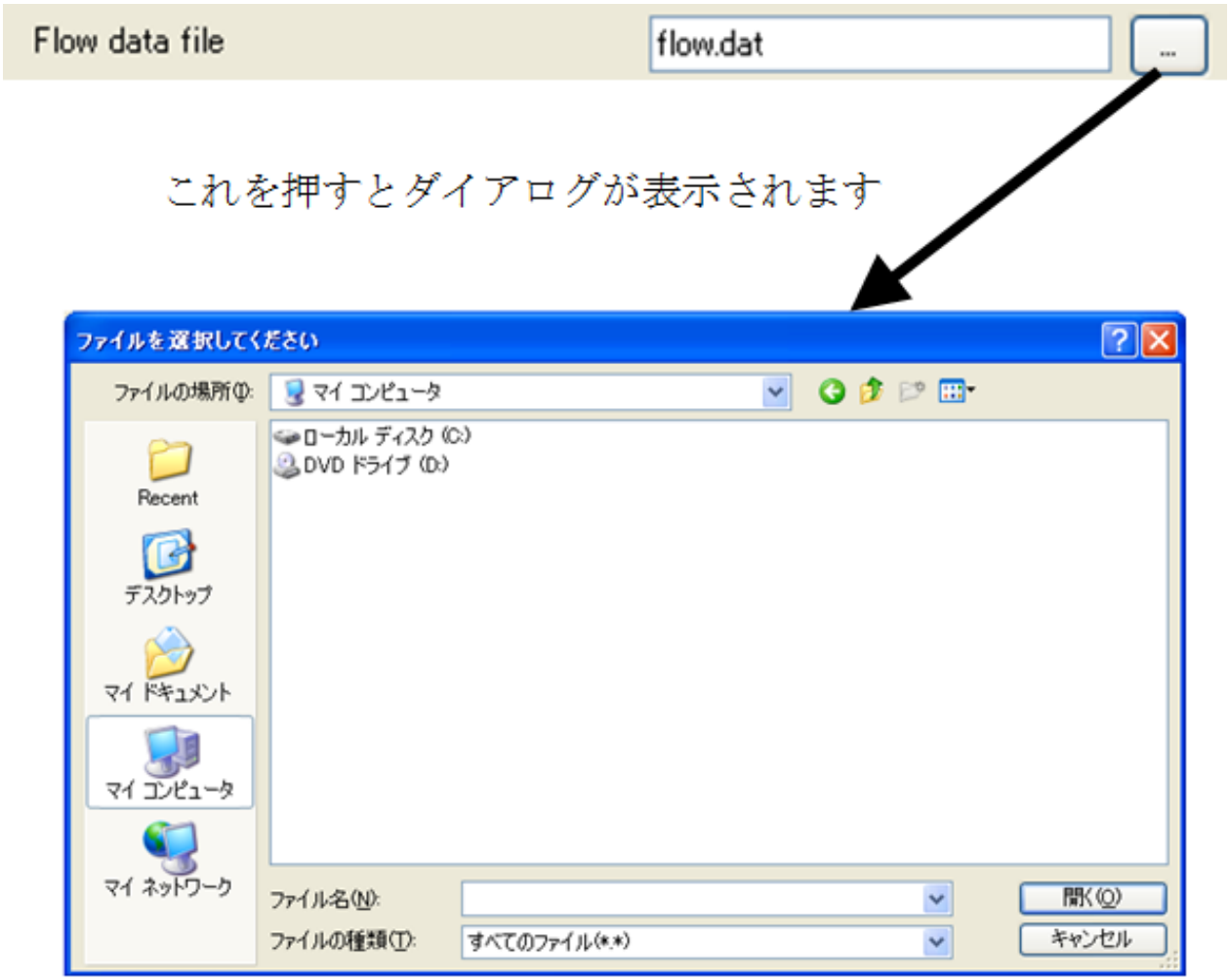

図 5.6　ファイル名 (読み込み用) の条件の表示例

リスト 5.6　ファイル名 (読み込み用) の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier
character(200):: flowdatafile

call cg_iric_read_string_f("flowdatafile", flowdatafile, ier)
```

リスト 5.7　ファイル名 (読み込み用) の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier
character(200):: flowdatafile

call cg_iric_read_bc_string_f("inflow", 1, "flowdatafile", flowdatafile, ier)
```

ファイル名 (書き込み用)

リスト 5.8　ファイル名 (書き込み用) の条件の定義例

```
<Item name="flowdatafile" caption="Flow data file">
  <Definition valueType="filename_all" default="flow.dat" />
</Item>
```

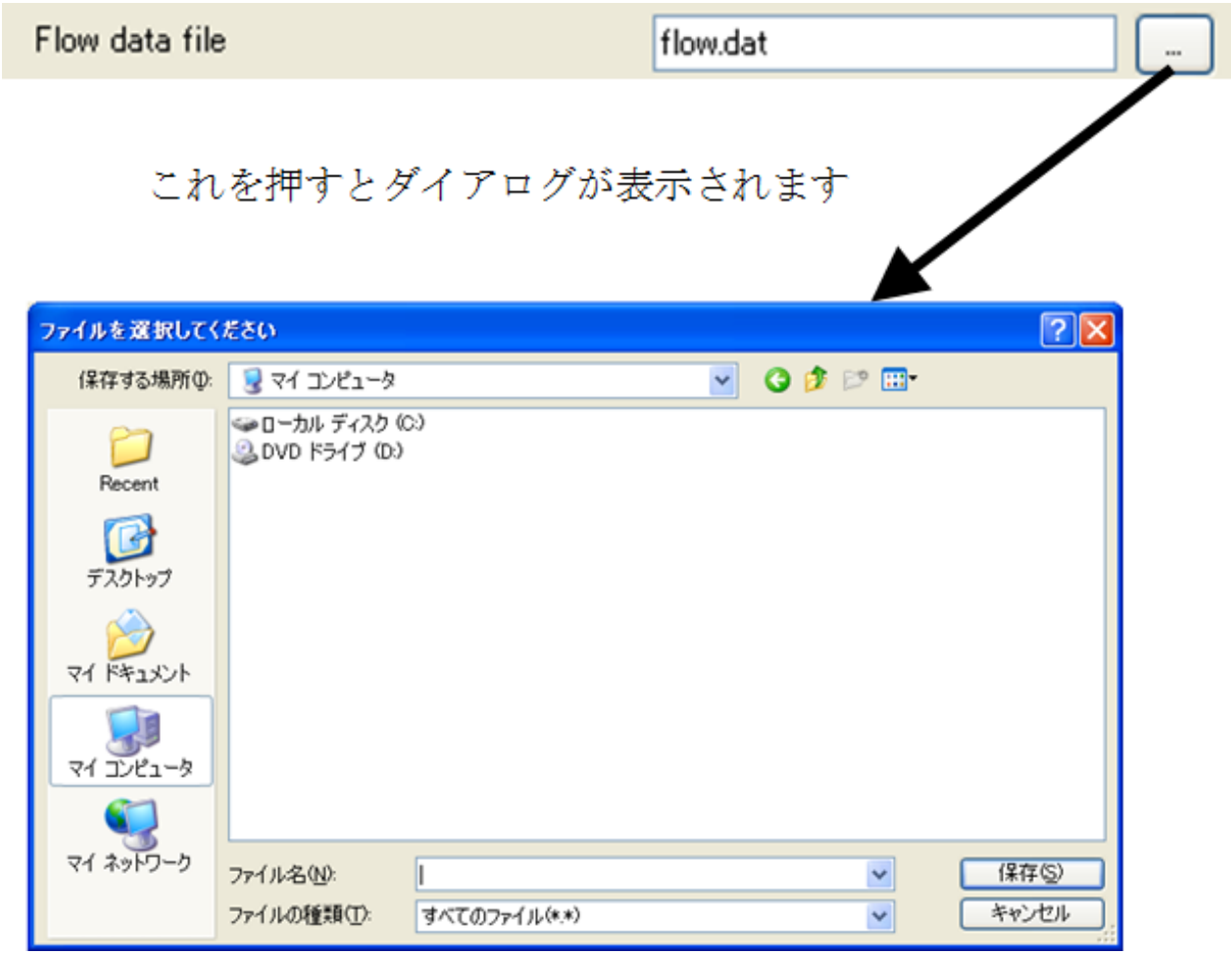

図 5.7　ファイル名 (書き込み用) の条件の表示例

リスト 5.9　ファイル名 (書き込み用) の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier
character(200):: flowdatafile

call cg_iric_read_string_f("flowdatafile", flowdatafile, ier)
```

リスト 5.10　ファイル名 (書き込み用) の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier
character(200):: flowdatafile

call cg_iric_read_bc_string_f("inflow", 1, "flowdatafile", flowdatafile, ier)
```

フォルダ名

リスト 5.11　ファイル名 (書き込み用) の条件の定義例

```
<Item name="flowdatafolder" caption="Flow data folder">
  <Definition valueType="foldername" />
</Item>
```

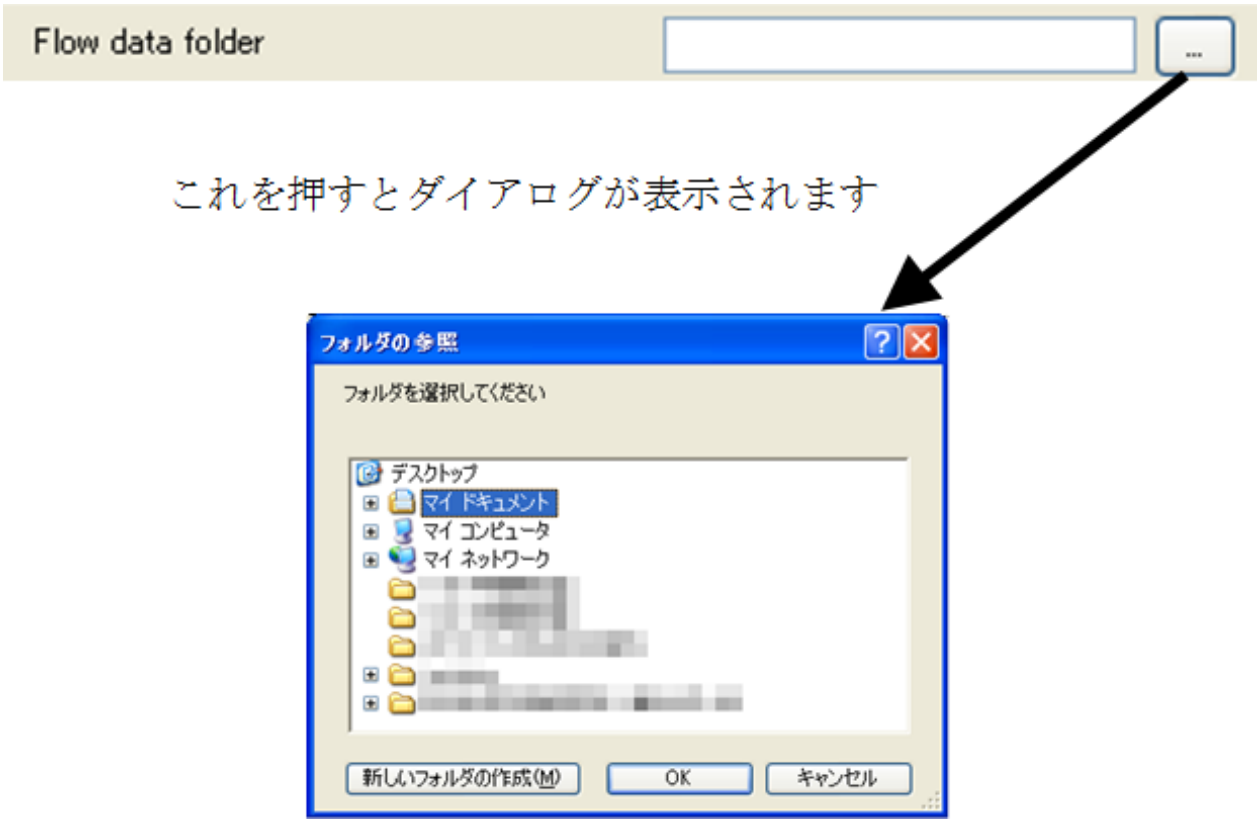

図 5.8　フォルダ名の条件の表示例

リスト 5.12　フォルダ名の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier
character(200):: flowdatafolder

call cg_iric_read_string_f("flowdatafolder", flowdatafolder, ier)
```

リスト 5.13　フォルダ名の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier
character(200):: flowdatafolder

call cg_iric_read_bc_string_f("inflow", 1, "flowdatafolder", flowdatafolder, ier)
```

整数

リスト 5.14　整数の条件の定義例

```
<Item name="numsteps" caption="The Number of steps to calculate">
  <Definition valueType="integer" default="20" min="1" max="200" />
</Item>
```

図 5.9　整数の条件の表示例

リスト 5.15　整数の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier, numsteps

call cg_iric_read_integer_f("numsteps", numsteps, ier)
```

リスト 5.16　整数の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier, numsteps

call cg_iric_read_bc_integer_f("inflow", 1, "numsteps", numsteps, ier)
```

整数 (選択式)

リスト 5.17　整数 (選択式) の条件の定義例

```
<Item name="flowtype" caption="Flow type">
  <Definition valueType="integer" default="0">
    <Enumeration value="0" caption="Static Flow"/>
    <Enumeration value="1" caption="Dynamic Flow"/>
  </Definition>
</Item>
```

図 5.10　整数 (選択式) の条件の表示例

リスト 5.18　整数 (選択式) の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier, flowtype

call cg_iric_read_integer_f("flowtype", flowtype, ier)
```

リスト 5.19　整数 (選択式) の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier, flowtype

call cg_iric_read_bc_integer_f("inflow", 1, "flowtype", flowtype, ier)
```

実数

リスト 5.20　実数の条件の定義例

```
<Item name="g" caption="Gravity [m/s2]">
  <Definition valueType="real" default="9.8" />
</Item>
```

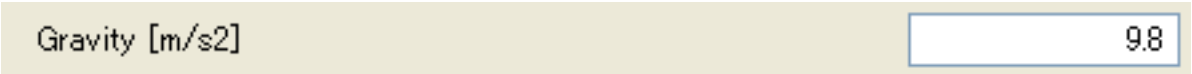

図 5.11　実数の条件の表示例

リスト 5.21　実数の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier
double precision:: g

call cg_iric_read_real_f("g", g, ier)
```

リスト 5.22　実数の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier
double precision:: g

call cg_iric_read_bc_real_f("inflow", 1, "g", g, ier)
```

関数型

リスト 5.23　関数型の条件の定義例

```
<Item name="discharge" caption="Discharge time series">
  <Definition valueType="functional" >
    <Parameter valueType="real" caption="Time" />
    <Value valueType="real" caption="Discharge" />
  </Definition>
</Item>
```

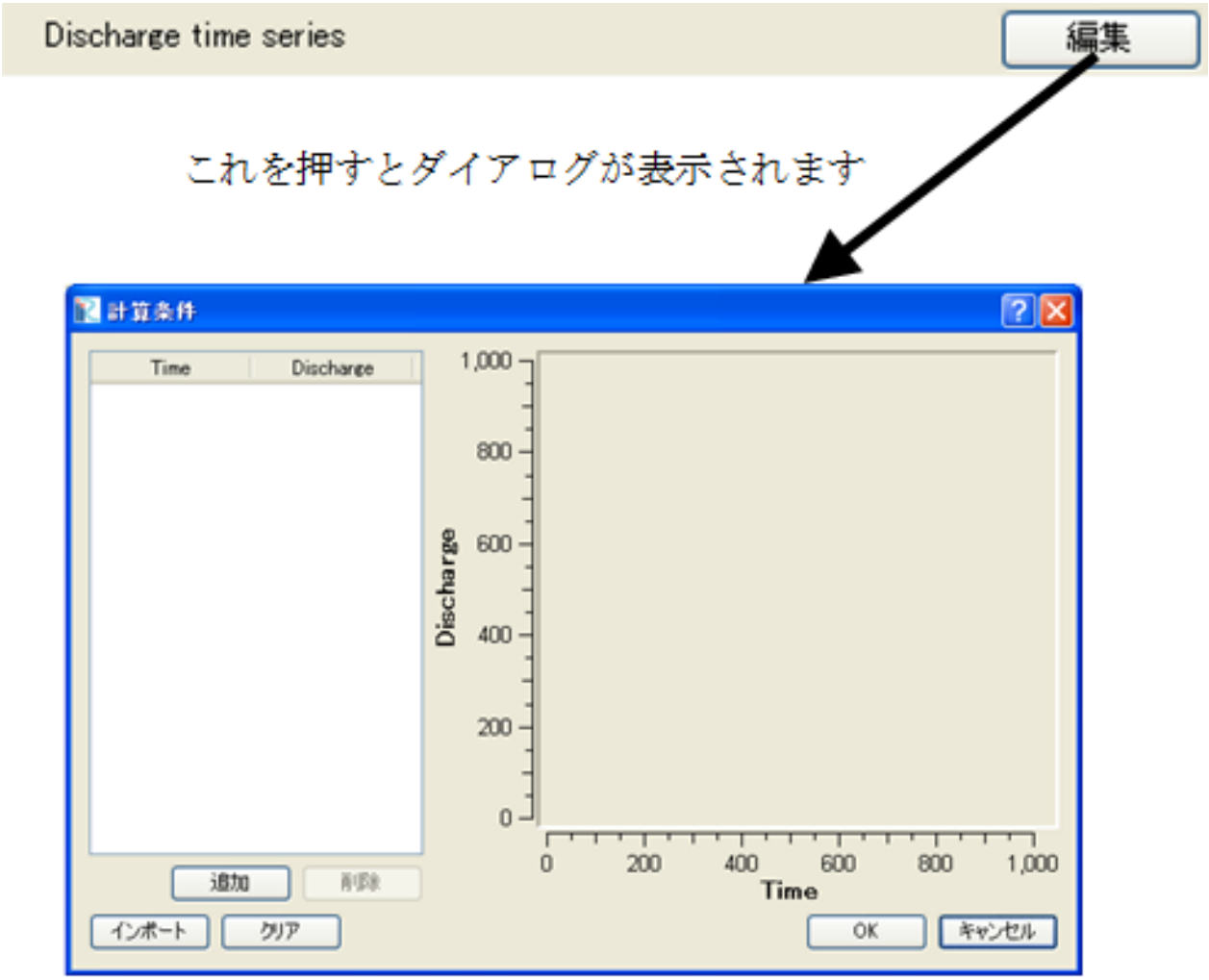

図 5.12　関数型の条件の表示例

リスト 5.24　関数型の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier, discharge_size
double precision, dimension(:), allocatable:: discharge_time, discharge_value

! サイズを調べる
call cg_iric_read_functionalsize_f("discharge", discharge_size, ier)
! メモリを確保
allocate(discharge_time(discharge_size))
allocate(discharge_value(discharge_size))
! 確保したメモリに値を読み込む
call cg_iric_read_functional_f("discharge", discharge_time, discharge_value, ier)
```

リスト 5.25　関数型の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier, discharge_size
double precision, dimension(:), allocatable:: discharge_time, discharge_value

! サイズを調べる
call cg_iric_read_bc_functionalsize_f("inflow", 1, "discharge", discharge_size, ier)
! メモリを確保
allocate(discharge_time(discharge_size))
allocate(discharge_value(discharge_size))
! 確保したメモリに値を読み込む
call cg_iric_read_bc_functional_f("inflow", 1, "discharge", discharge_time, discharge_
↪value, ier)
```

関数型 (複数の値)

リスト 5.26　関数型 (複数の値) の条件の定義例

```
<Item name="discharge_and_elev" caption="Discharge and Water Elevation time series">
  <Definition valueType="functional" >
    <Parameter name="time" valueType="real" caption="Time" />
    <Value name="discharge" valueType="real" caption="Discharge" />
    <Value name="elevation" valueType="real" caption="Water Elevation" />
  </Definition>
</Item>
```

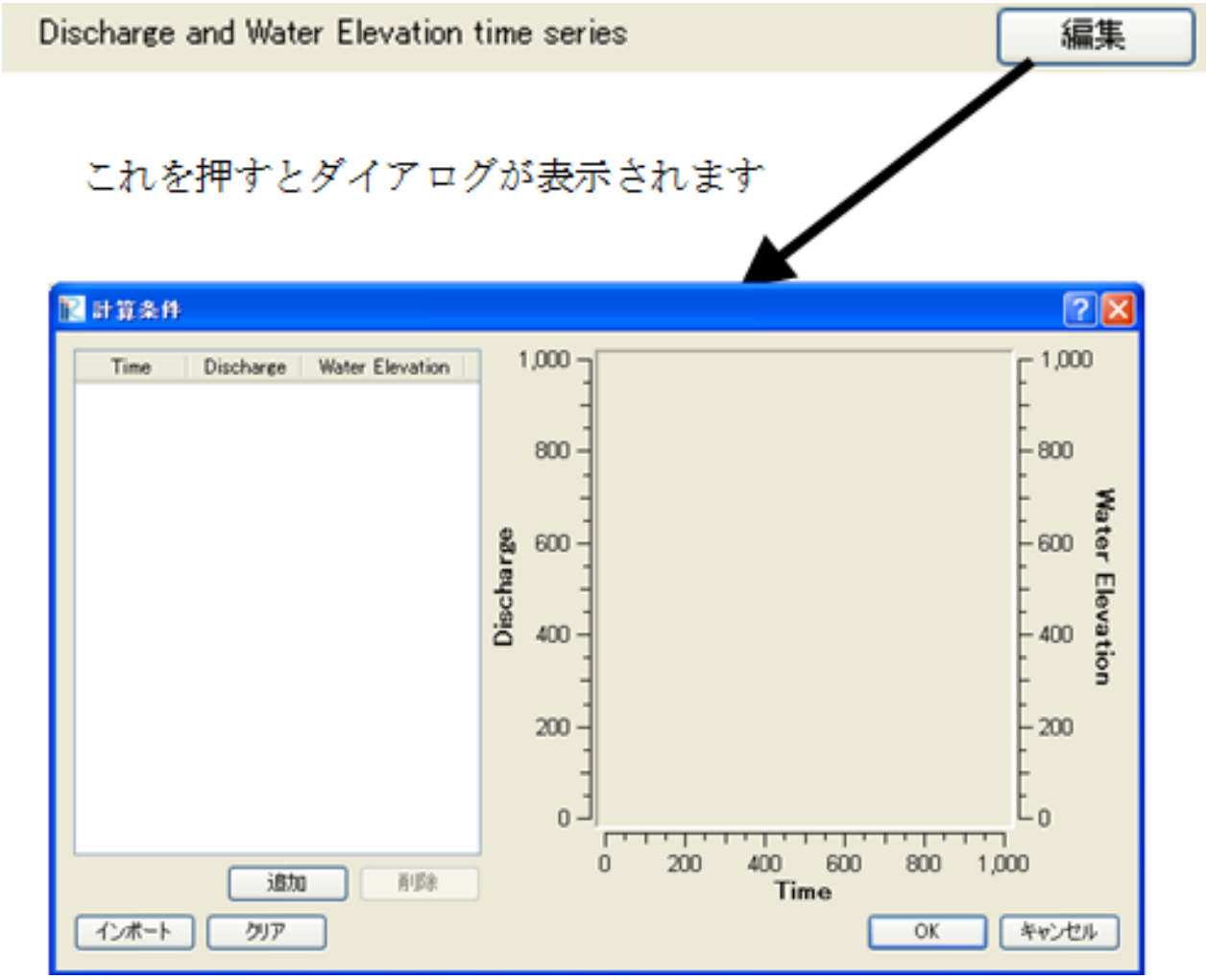

図 5.13　関数型 (複数の値) の条件の表示例

リスト 5.27　関数型 (複数の値) の条件を読み込むための処理の記述例 (計算条件・格子生成条件)

```
integer:: ier, discharge_size
double precision, dimension(:), allocatable:: time_value
double precision, dimension(:), allocatable:: discharge_value, elevation_value

! サイズを調べる
call cg_iric_read_functionalsize_f("discharge", discharge_size, ier)
! メモリを確保
allocate(time_value(discharge_size))
allocate(discharge_value(discharge_size), elevation_value(discharge_size))
! 確保したメモリに値を読み込む
call cg_iric_read_functionalwithname_f("discharge", "time", time_value)
call cg_iric_read_functionalwithname_f("discharge", "discharge", discharge_value)
call cg_iric_read_functionalwithname_f("discharge", "elevation", elevation_value)
```

リスト 5.28　関数型 (複数の値) の条件を読み込むための処理の記述例 (境界条件)

```
integer:: ier, discharge_size
double precision, dimension(:), allocatable:: time_value
double precision, dimension(:), allocatable:: discharge_value, elevation_value

! サイズを調べる
call cg_iric_read_bc_functionalsize_f("discharge", discharge_size, ier)
! メモリを確保
allocate(time_value(discharge_size))
allocate(discharge_value(discharge_size), elevation_value(discharge_size))
! 確保したメモリに値を読み込む
call cg_iric_read_bc_functionalwithname_f("discharge", "time", time_value)
call cg_iric_read_bc_functionalwithname_f("discharge", "discharge", discharge_value)
call cg_iric_read_bc_functionalwithname_f("discharge", "elevation", elevation_value)
```

5.4 要素のリファレンス

5.4.1 BoundaryCondition

境界条件の情報を保持します。

例

リスト 5.29　BoundaryCondition の定義例

```
<BoundaryCondition name="inflow" caption="In flow" position="node">
  <Item name="discharge" caption="Discharge">
    <Definition valueType="real" default="0" />
  </Item>
</BoundaryCondition>
```

属性

表 5.3　BoundaryCondition の属性

名前	値	必須	説明
name	文字列	○	要素名
caption	文字列	○	名前 (ダイアログ上に表示される)
position	下の表を参照	○	定義位置

表 5.4　position の値

値	意味
node	格子点
cell	セル
edge	格子の辺

子要素

表 5.5　BoundaryCondition の子要素

名前	必須	説明
Item		要素の定義

5.4.2 CalculationCondition

計算条件の情報を保持します。

例

リスト 5.30　CalculationCondition の定義例

```
<CalculationCondition>
  <Tab name="basic" caption="Basic Setting">

    （略）

  </Tab>
  <Tab name="time" caption="Calculation Time Setting">

    （略）

  </Tab>
</CalculationCondition>
```

属性

定義できる属性はありません。

子要素

表 5.6　CalculationCondition の子要素

名前	必須	説明
Tab		計算条件ダイアログの各ページの情報を持つ要素。

5.4.3 Condition

計算条件 (もしくは格子生成条件) での入力項目が有効になる場合の条件の情報を保持します。

例

リスト 5.31　Condition の定義例 1

```
<Condition conditionType="isEqual" target="type" value="1" />
```

リスト 5.32　Condition の定義例 2

```
<Condition conditionType="and">
  <Condition conditionType="isEqual" target="type" value="1" />
  <Condition conditionType="isEqual" target="inflow" value="0" />
</Condition>
```

例は ????? も参照して下さい。

属性

表 5.7　Condition の属性

名前	値	必須	説明
conditionType	下の表を参照	○	条件の種類
target	文字列	△	比較対象の計算条件の名前。 conditionType が and, or, not の場合に指定する。
value	文字列	△	比較対象の値。 conditionType が and, or, not の場合に指定する。

表 5.8　conditionType の値

値	意味
isEqual	等しい
isGreaterEqual	等しいか大きい
isGreaterThan	大きい
isLessEqual	等しいか小さい
isLessThan	小さい
and	and
or	or
not	not

子要素

表 5.9　Condition の子要素

名前	必須	説明
Condition		and, or, not の演算子を適用する対象条件。conditionType 属性 が and, or, not のいずれかの場合のみ指定する。

5.4.4 Definition (計算条件・境界条件・格子生成条件の定義)

計算条件・境界条件・格子生成条件の定義情報を保持します。

例

リスト 5.33　Definition の定義例 1

```
<Definition valueType="integer" default="1" />
```

リスト 5.34　Definition の定義例 2

```
<Definition valueType="integer" default="0" >
  <Enumeration value="0" caption="Standard" />
  <Enumeration value="1" caption="Advanced" />
</Definition>
```

例は 計算条件・境界条件・格子生成条件の項目の定義と読み込み処理の例 (ページ 74) も参照して下さい。

属性

表 5.10　Definition の属性

名前	値	必須	説明
valueType	下の表を参照	○	値の種類
default	文字列		デフォルト値

表 5.11　valueType の値

値	意味
integer	整数
real	実数
string	文字列
filename	ファイル名
filename_all	ファイル名。存在しないファイルも選択可能
foldername	フォルダ名
functional	関数型

子要素

表 5.12　Definition の子要素

名前	必須	説明
Enumeration		値を選択肢からのみ選べるようにしたい場合に指定する。valueType が integer, real の場合のみ指定できる。
Condition		この条件が、有効になる時の条件を指定する。

5.4.5 Definition (格子属性の定義)

計算格子の属性の定義情報を保持します。

例

リスト 5.35　Definition の定義例

```
<Definition valueType="integer" position="node" default="min" />
```

属性

表 5.13　Definition の属性

名前	値	必須	説明
valueType	下の表を参照	○	値の種類
position	下の表を参照	○	属性の定義位置
default	文字列		デフォルト値。「min」,「max」を指定すると、地理情報が存在しない領域では最小値、最大値が利用される。

表 5.14　valueType の値

値	意味
integer	整数
real	実数
complex	複合型

表 5.15　position の値

値	意味
node	格子点
cell	セル

子要素

表 5.16　Definition の子要素

名前	必須	説明
Dimension		次元 (例: 時刻) を追加したい場合に指定する。
Enumeration		値を選択肢からのみ選べるようにしたい場合に指定する。
Item		valueType="complex" の場合のみ指定する。ここで定義する Item 要素以下の構造は、BoundaryCondition 要素の下の Item 要素と同じ。

5.4.6 Dimension

計算格子属性の次元の定義情報を保持します。

例

リスト 5.36　Dimension の定義例

```
<Dimension name="Time" caption="Time" valueType="integer" />
```

属性

表 5.17　Dimension の属性

名前	値	必須	説明
name	文字列	○	要素名
caption	文字列	○	名前 (プリプロセッサ上に表示される)
valueType	下の表を参照	○	値の種類

表 5.18　valueType の値

値	意味
integer	整数
real	実数

子要素

定義できる子要素はありません。

5.4.7 Enumeration

計算条件 (もしくは格子生成条件) の項目の選択肢の定義情報を保持します。

例

リスト 5.37　Enumeration の定義例

```
<Enumeration value="0" caption="Standard" />
```

Enumeration を使って計算条件を定義した例は整数 (選択式) (ページ 80) を参照して下さい。

属性

表 5.19　Enumeration の属性

名前	値	必須	説明
value	文字列	○	caption に対応する値
caption	文字列	○	表示する文字列

子要素

定義できる子要素はありません。

5.4.8 ErrorCode

エラーコードの定義情報を保持します。

例

リスト 5.38　ErrorCode の定義例

```
<ErrorCode value="1" caption="Grid is data do not exist" />
```

属性

表 5.20　ErrorCode の属性

名前	値	必須	説明
value	整数	○	エラーコード
caption	文字列	○	表示する文字列

子要素

定義できる子要素はありません。

5.4.9 ErrorCodes

エラーコードのリストを保持します。

例

リスト 5.39　ErrorCodes の定義例

```
<ErrorCodes>
  <ErrorCode value="1" caption="Grid is data do not exist" />
  <ErrorCode value="2" caption="Grid size is too big" />
</ErrorCodes>
```

属性

定義できる属性はありません。

子要素

表 5.21　ErrorCodes の子要素

名前	必須	説明
ErrorCode		エラーコード

5.4.10 GridGeneratingCondition

格子生成条件の情報を保持します。

例

リスト 5.40　GridGeneratingCondition の定義例

```
<GridGeneratingCondition>
  <Tab name="basic" caption="Basic Setting">

    (略)

  </Tab>
```

```
  <Tab name="time" caption="Calculation Time Setting">

    （略）

  </Tab>
</GridGeneratingCondition>
```

属性

定義できる属性はありません。

子要素

表 5.22　GridGeneratingCondition の子要素

名前	必須	説明
Tab		格子生成条件ダイアログの各ページの情報を持つ要素。

5.4.11 GridGeneratorDefinition

格子生成プログラムの定義情報を保持します。

例

リスト 5.41　GridGeneratorDefinition の定義例

```
<GridGeneratorDefinition
  name="samplecreator"
  caption="Sample Grid Creator"
  version="1.0"
  copyright="Example Company"
  executable="generator.exe"
  gridtype="structured2d"
>
  <GridGeneratingCondition>
    <Tab name="basic" caption="Basic Setting">

      （略）

    </Tab>
  </GridGeneratingCondition>
</GridGeneratorDefinition>
```

属性

表 5.23　GridGeneratorDefinition の属性

名前	値	必須	説明
name	文字列	○	格子生成プログラムの識別名 (英数字のみ)
caption	文字列	○	格子生成プログラムの名前 (任意の文字を利用可能)
version	文字列	○	バージョン番号。"1.0", "1.3.2" などの型式で
copyright	文字列	○	著作権者の名前。基本的に英語で記述
release	文字列	○	リリース日。"2010.01.01" などの型式で
homepage	文字列	○	格子生成プログラム情報を示す Web ページの URL
executable	文字列	○	実行プログラムのファイル名 (例: GridGen.exe)
gridtype	下の表を参照	○	生成する格子の種類

表 5.24　gridType の値

値	意味
structured2d	2 次元構造格子
unstructured2d	2 次元非構造格子

子要素

表 5.25　GridGeneratingCondition の子要素

名前	必須	説明
GridGeneratingCondition	○	格子生成条件
ErrorCodes		エラーコードのリスト

5.4.12 GridLayout

計算条件 (もしくは格子生成条件) の入力ダイアログに表示する格子状のレイアウトの定義情報を保持します。

例

GridLayout の定義例は ???? を参照して下さい。

属性

定義できる属性はありません。

子要素

表 5.26　GridLayout の子要素

名前	必須	説明
Item, VBoxLayout など		レイアウト内に表示する要素や子レイアウトの定義

5.4.13 GridRelatedCondition

格子属性の定義情報のリストを保持します。

例

リスト 5.42　GridRelatedCondition の定義例

```
<GridRelatedCondition>
  <Item name="Elevation" caption="Elevation(m)">
    <Definition valueType="real" position="node" default="max" />
  </Item>
</GridRelatedCondition>
```

属性

定義できる属性はありません。

子要素

表 5.27　GridRelatedCondition の子要素

名前	必須	説明
Item	○	格子属性

5.4.14 GridType

入力格子の定義情報を保持します。

例

リスト 5.1 を参照して下さい。

属性

表 5.28　GridType の属性

名前	値	必須	説明
gridtype	下の表を参照	○	格子の種類
multiple	true or false		複数の格子を生成できるなら true

表 5.29　gridtype の値

値	意味
1d	1 次元格子
1.5d	1.5 次元格子
1.5d_withcrosssection	横断データを持つ 1.5 次元格子
structured2d	2 次元構造格子
unstructured2d	2 次元非構造格子

子要素

表 5.30　GridType の子要素

名前	必須	説明
GridRelatedCondition	○	格子属性

5.4.15 GridTypes

入力格子の定義情報のリストを保持します。

例

リスト 5.1 を参照して下さい。

属性

定義できる属性はありません。

子要素

表 5.31　GridTypes の子要素

名前	必須	説明
GridType	○	格子の種類

5.4.16 GroupBox

計算条件 (もしくは格子生成条件) の入力ダイアログに表示するグループボックスの定義情報を保持します。

例

リスト 5.43　GroupBox の定義例

```
<GroupBox caption="Time">
  <Item name="stime" caption="Start Time">
    <Definition valueType="real" />
  </Item>
  <Item name="etime" caption="End Time">
    <Definition valueType="real" />
  </Item>
</GroupBox>
```

GroupBox の定義例は ???? も参照して下さい。

属性

表 5.32　GroupBox の属性

名前	値	必須	説明
caption	文字列	○	表示する文字列

子要素

表 5.33　GroupBox の子要素

名前	必須	説明
Item, VBoxLayout など		グループボックス内に表示する要素やレイアウトの定義

5.4.17 HBoxLayout

計算条件 (もしくは格子生成条件) の入力ダイアログに表示する水平に並べるレイアウトの定義情報を保持します。

例

リスト 5.44　HBoxLayout の定義例

```
<HBoxLayout>
  <Item name="stime" caption="Start Time">

    （略）

  </Item>
  <Item name="etime" caption="End Time">

    （略）

  </Item>
</HBoxLayout>
```

属性

定義できる属性はありません。

子要素

表 5.34　HBoxLayout の子要素

名前	必須	説明
Item, VBoxLayout など		レイアウト内に表示する要素や子レイアウトの定義

5.4.18 Item

計算条件 (もしくは格子生成条件) の入力項目、計算格子の属性、境界条件の定義情報を保持します。

例

リスト 5.45　Item の定義例

```
<Item name="stime" caption="Start Time">
  <Definition valueType="real" default="0" />
</Item>
```

定義例は 計算条件・境界条件・格子生成条件の項目の定義と読み込み処理の例 (ページ 74) も参照して下さい。

属性

表 5.35　Item の属性

名前	値	必須	説明
name	文字列	○	要素名
caption	文字列		名前 (ダイアログ上に表示される)

子要素

表 5.36　Item の子要素

名前	必須	説明
Definition	○	要素の定義

5.4.19 Label

計算条件 (もしくは格子生成条件) の入力ダイアログに表示するラベルの定義情報を保持します。

例

リスト 5.46　Label の定義例

```
<Label caption="Start Time" />
```

定義例は ?????? も参照して下さい。

属性

表 5.37　Label の属性

名前	値	必須	説明
caption	文字列		名前 (ダイアログ上に表示される)

子要素

定義できる子要素はありません。

5.4.20 Param

計算条件 (もしくは格子生成条件) の入力ダイアログに表示するラベルの定義情報を保持します。

例

リスト 5.47　Param の定義例

```
<Param caption="Time" valueType="real" />
```

定義例は 関数型 (ページ 82) も参照して下さい。

属性

表 5.38　Param の属性

名前	値	必須	説明
caption	文字列	○	表示する文字列
valueType	下の表を参照	○	データ型
axislog	true or false		横軸を対数軸にするなら true

子要素

定義できる子要素はありません。

5.4.21 SolverDefinition

ソルバーの定義情報を保持します。

例

リスト 5.48　SolverDefinition の定義例

```
<SolverDefinition
  name="samplesolver"
  caption="Sample Solver 1.0"
  version="1.0"
  copyright="Example Company"
```

```
  release="2012.04.01"
  homepage="http://example.com/"
  executable="solver.exe"
  iterationtype="time"
  gridtype="structured2d"
>
  <CalculationCondition>

    （略）

  </CalculationCondition>
  <GridRelatedCondition>

    （略）

  </GridRelatedCondition>
</SolverDefinition>
```

属性

表 5.39　SolverDefinition の属性

名前	値	必須	説明
name	文字列	○	ソルバーの識別名 (英数字のみ)
caption	文字列	○	ソルバーの名前 (任意の文字を利用可能)
version	文字列	○	バージョン番号。"1.0", "1.3.2" などの型式で
copyright	文字列	○	著作権者の名前。基本的に英語で記述
release	文字列	○	リリース日。"2010.01.01" などの型式で
homepage	文字列	○	ソルバー情報を示す Web ページの URL
executable	文字列	○	実行プログラムのファイル名 (例: GridGen.exe)
iterationtype	下の表を参照	○	計算結果出力の単位
gridtype	下の表を参照	○	生成する格子の種類
multiple	true or false		複数の格子を生成できるなら true

表 5.40　iterationtype の値

値	意味
time	時間ごとの結果を出力
iteration	イテレーションごとの結果を出力。

表 5.41　gridtype の値

値	意味
1d	1 次元格子
1.5d	1.5 次元格子
1.5d_withcrosssection	横断データを持つ 1.5 次元格子
structured2d	2 次元構造格子
unstructured2d	2 次元非構造格子

ソルバーのバージョンアップを行う時は、 version 属性を変更します。ソルバーのバージョンアップ時の注意点については、???? を参照して下さい。

子要素

表 5.42　SolverDefinition の子要素

名前	必須	説明
CalculationCondition	○	計算条件
GridRelatedCondition		1 種類の入力格子を利用する場合のみ定義する
GridTypes		2 種類以上の入力格子を利用する場合のみ定義する

5.4.22 Tab

計算条件 (もしくは格子生成条件) 設定ダイアログの、ページの定義情報を保持します。

例

リスト 5.49　Tab の定義例

```
<Tab caption="Basic Setting">
  <Item name="stime" caption="Start Time">

    (略)

  </Item>
  <Item name="etime" caption="End Time">

    (略)

  </Item>
</Tab>
```

属性

表 5.43　Tab の属性

名前	値	必須	説明
caption	文字列	○	表示する文字列

子要素

表 5.44　Tab の子要素

名前	必須	説明
Item, GroupBox など	○	このページに表示する計算条件 (もしくは格子生成条件) の定義

5.4.23 Value

関数型の計算条件 (もしくは格子生成条件) 、格子属性、境界条件の値の定義情報を保持します。

例

リスト 5.50　Param の定義例

```
<value caption="Discharge" valueType="real" />
```

定義例は 関数型 (ページ 82) も参照して下さい。

属性

表 5.45　Value の属性

名前	値	必須	説明
caption	文字列	○	表示する文字列
valueType	下の表を参照	○	データ型
name	文字列		識別名 (英数字のみ)。複数の値を持つ関数型条件の場合のみ指定する。
axis	下の表を参照		設定ダイアログでの Y 軸
axislog	true or false		縦軸を対数軸にするなら true
axisreverse	true or false		Y 軸を上下逆転するなら true
step	true or false		グラフを棒グラフとして表示するなら true
hide	true or false		設定ダイアログのグラフに表示しない場合は true

表 5.46　valueType の値

値	意味
integer	整数
real	実数

表 5.47　axis の値

値	意味
left	左側の Y 軸を利用
right	右側の Y 軸を利用

子要素

定義できる子要素はありません。

5.4.24 VBoxLayout

計算条件 (もしくは格子生成条件) の入力ダイアログに表示する垂直に並べるレイアウトの定義情報を保持します。

例

リスト 5.51　VBoxLayout の定義例

```
<VBoxLayout>
  <Item name="stime" caption="Start Time">

    (略)

  </Item>
  <Item name="etime" caption="End Time">

    (略)

  </Item>
</VBoxLayout>
```

属性

定義できる属性はありません。

子要素

表 5.48　VBoxLayout の子要素

名前	必須	説明
Item, VBoxLayout など		レイアウト内に表示する要素や子レイアウトの定義