iRIC Developer's manual Documentation

リリース *3.0.0*

iRIC organization

2022 年 07 月 09 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 **ソルバーの開発手順** (ページ 3) 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 **定義ファイル** *(XML)* **について** (ページ 65)、*iRIClib* **について** (ページ 113)、 **その他の情報** (ページ 255) 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 **格子生成プログラムの開発手順** (ページ 39) 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 **定義ファイル** *(XML)* **について** (ページ 65)、*iRIClib* **について** (ページ 113)、 **その他の情報** (ページ 255) を参照してください。

第 2 章

ソルバーの開発手順

2.1 開発環境の準備

iRIC ソルバの開発環境を準備します。

- C/C++ で開発するには、Microsoft Visual C/C++ 2013 以降を使用してください。
- FORTRAN で開発するには、Intel Fortran 2015 以降を利用してください。

ソルバをビルドする際には、 cgnslib と iriclib をリンクする必要があります。

必要なヘッダファイルとライブラリは、 iRIC GUI に同梱されています。標準的な場所にインストールした場合、ファイルは以下にあります。

C:\Users\(userid)\iRIC\guis\prepost\sdk

2.2 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。

2.2.1 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。

2.2.2 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, Python, 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。

2.2.3 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。

2.2.4 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。

2.2.5 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。

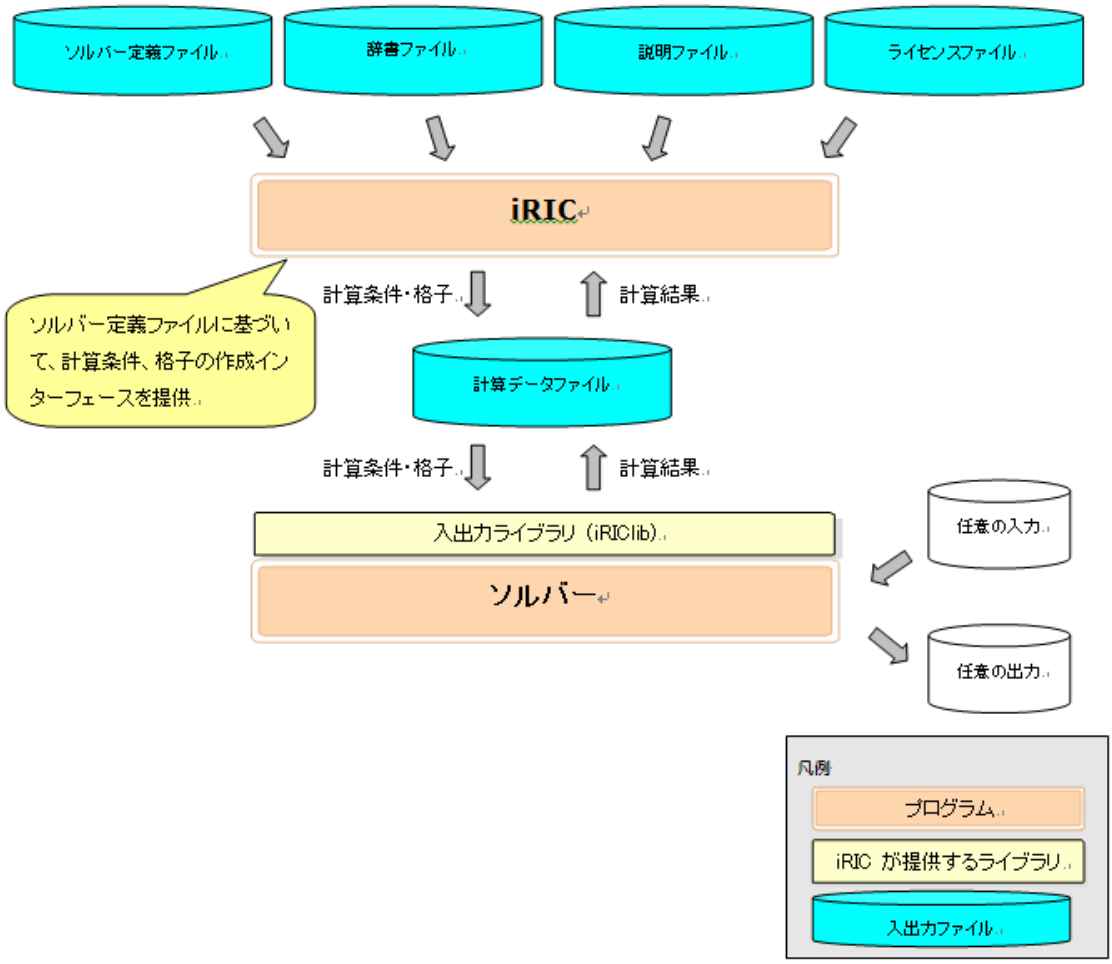

図 2.1　iRIC、ソルバー、関連ファイルの関係図

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。

2.3 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。

2.4 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 *XML* **の基礎** (ページ 110) 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。

2.4.1 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 6) 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"
>
```

(次のページに続く)

(前のページからの続き)

```
  <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

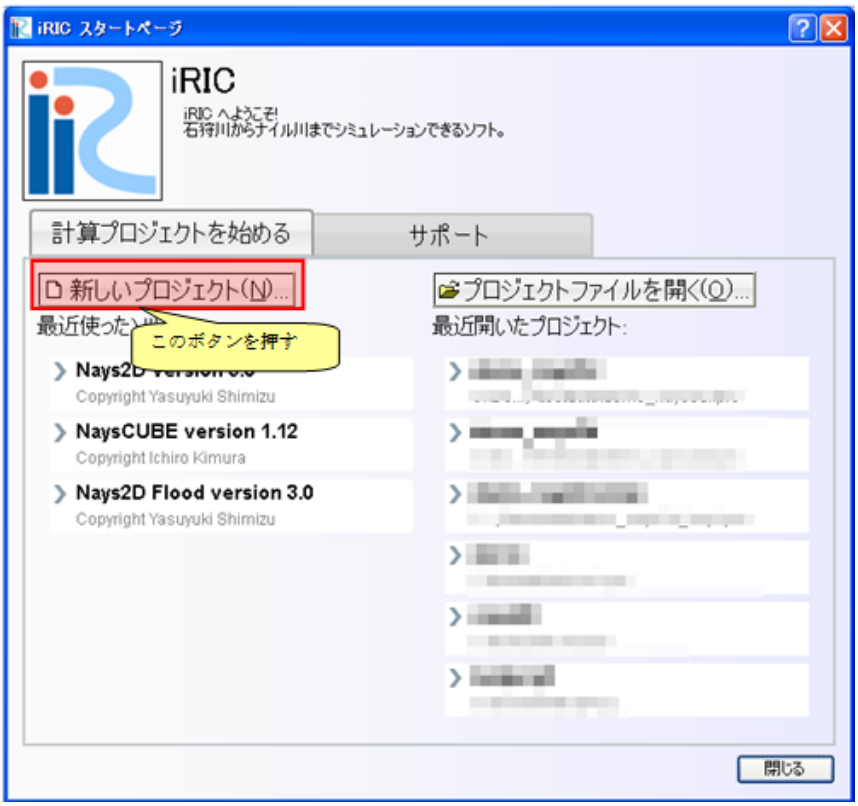

図 2.3　iRIC のスタートダイアログ

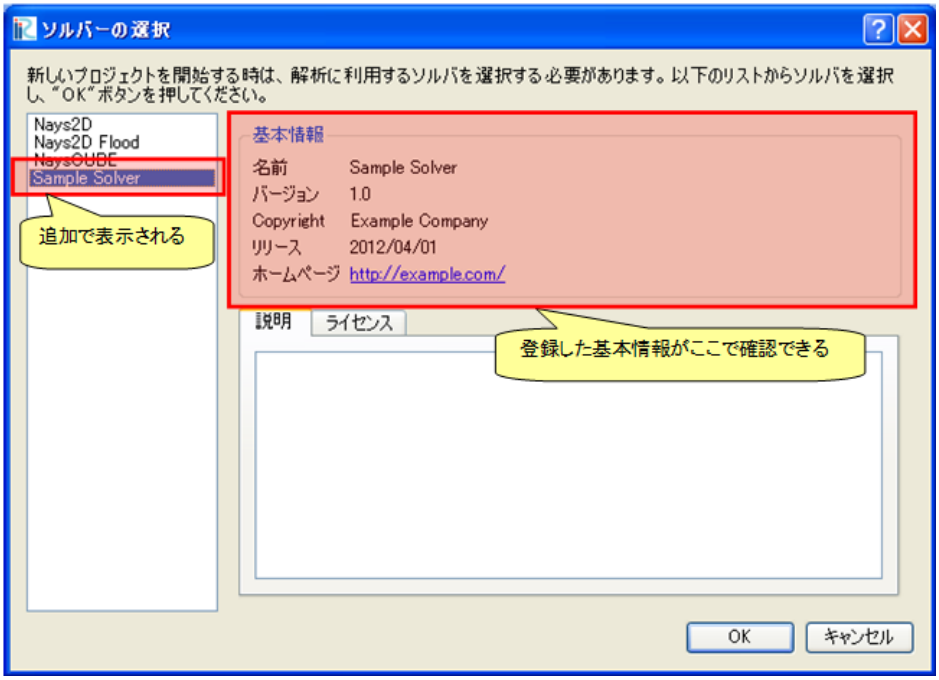

図 2.4　ソルバー選択ダイアログ

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**ソルバーのバージョンアップ時の注意点** (ページ 109) を参照してください。

2.4.2 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。

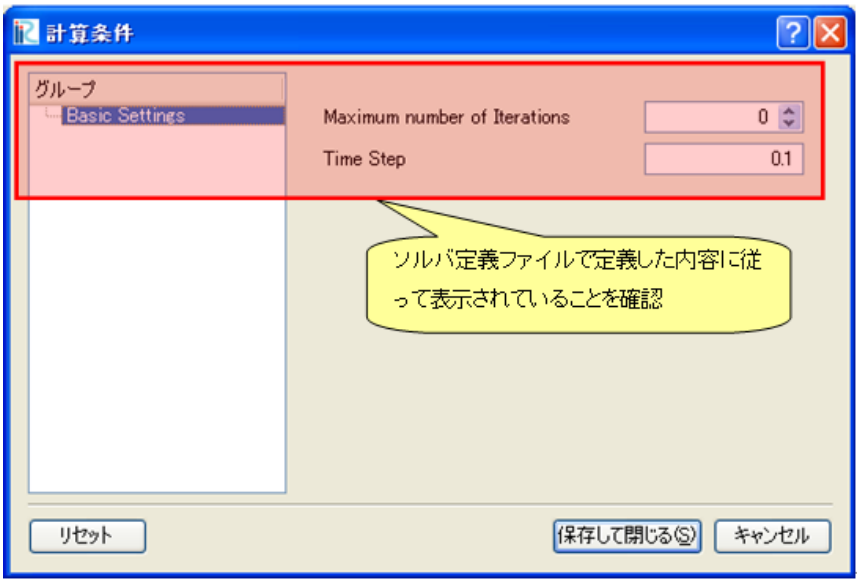

図 2.7　計算条件設定ダイアログ 表示例

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>
```

(次のページに続く)

(前のページからの続き)

```
  <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。

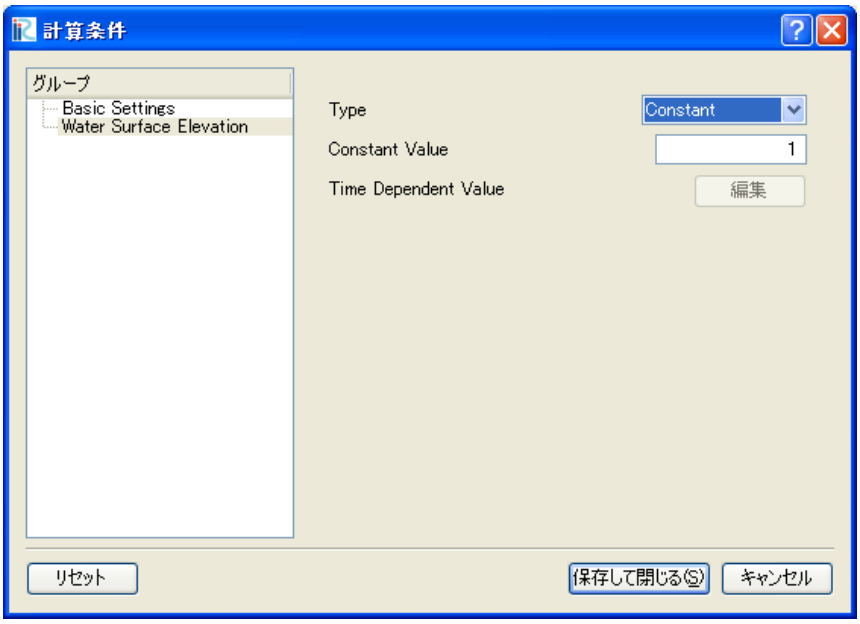

図 2.9　計算条件設定ダイアログ 表示例

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**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 70) 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、**計算条件・境界条件・格子生成条件の有効になる条件の定義例** (ページ 82) 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**計算条件・境界条件・格子生成条件のダイアログのレイアウト定義例** (ページ 82) を参照して下さい。

2.4.3 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。

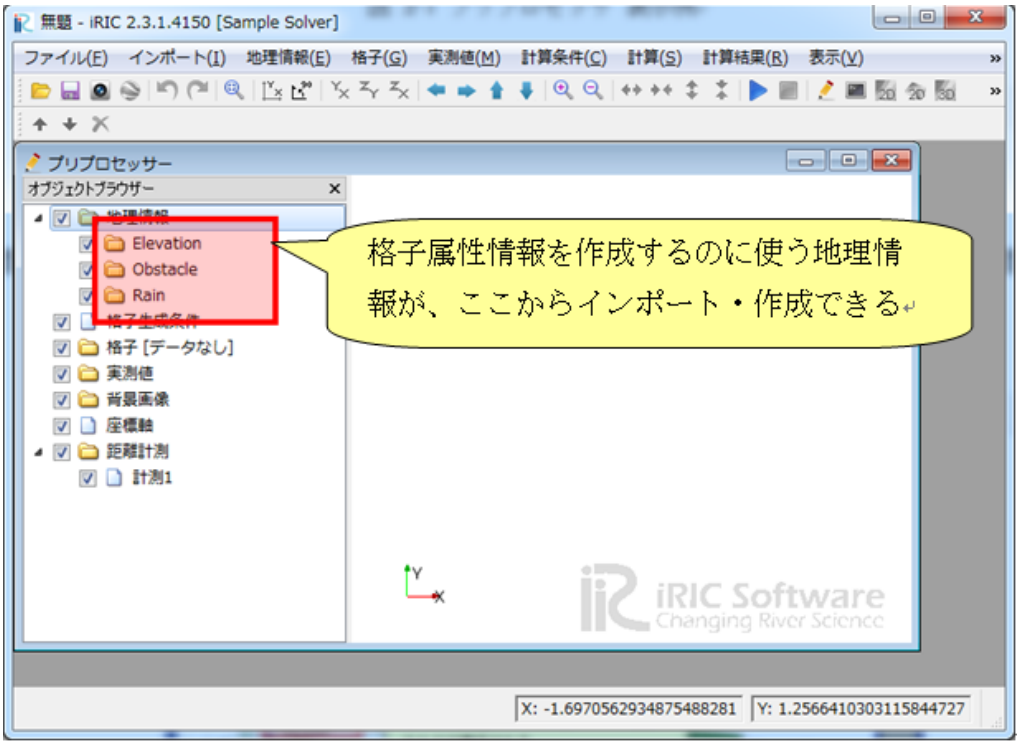

図 2.10　プリプロセッサ 表示例

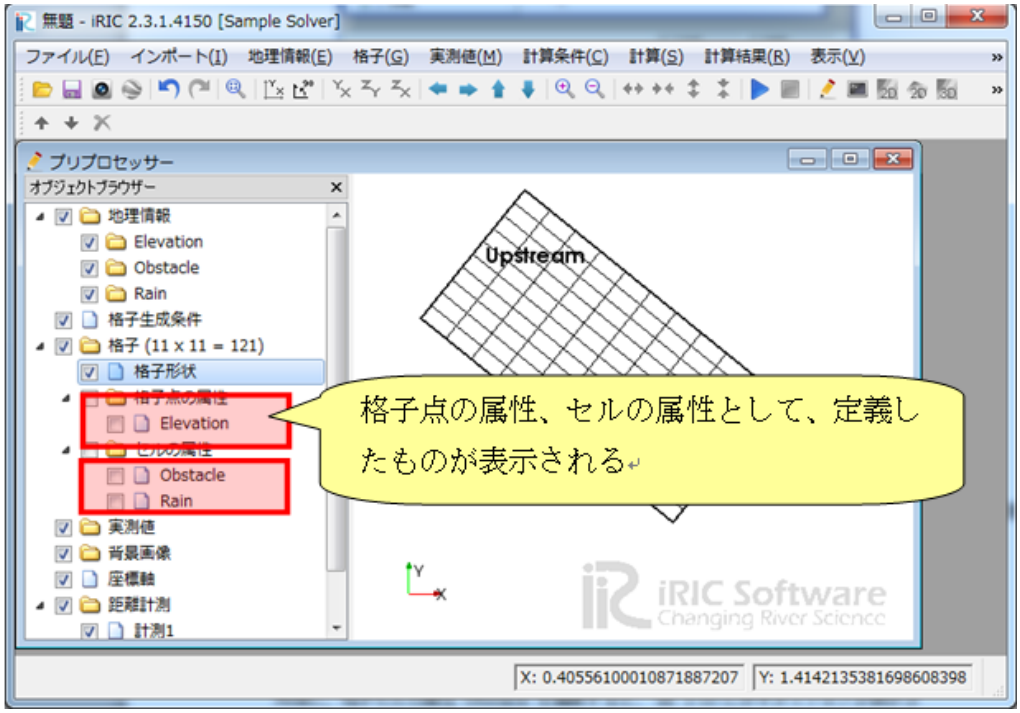

図 2.11　プリプロセッサ 表示例 (格子生成後)

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。

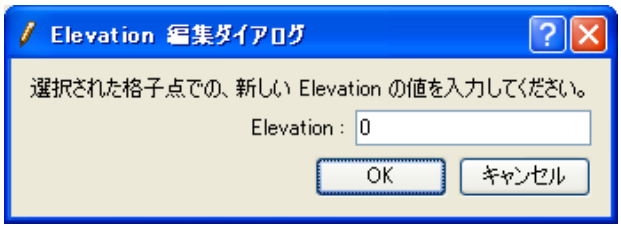

図 2.12　格子点の属性 "Elevation" の編集ダイアログ

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。

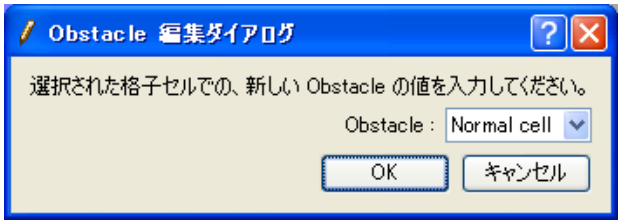

図 2.13　格子セルの属性 "Obstacle" の編集ダイアログ

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**特別な格子属性、計算結果の名前について** (ページ 257) を参照してください。

2.4.4 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 13) 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。

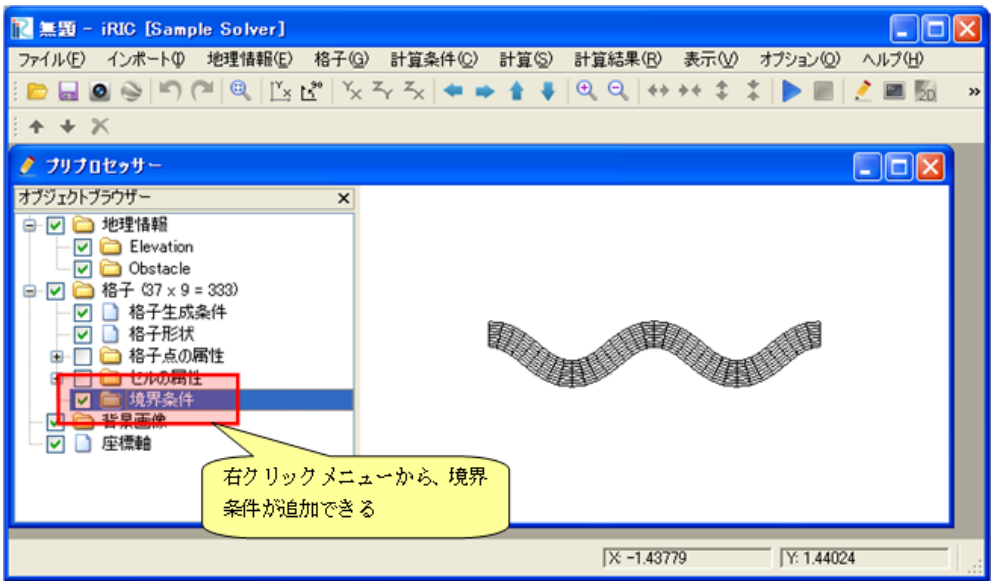

図 2.14　プリプロセッサ 表示例 (格子作成後)

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。

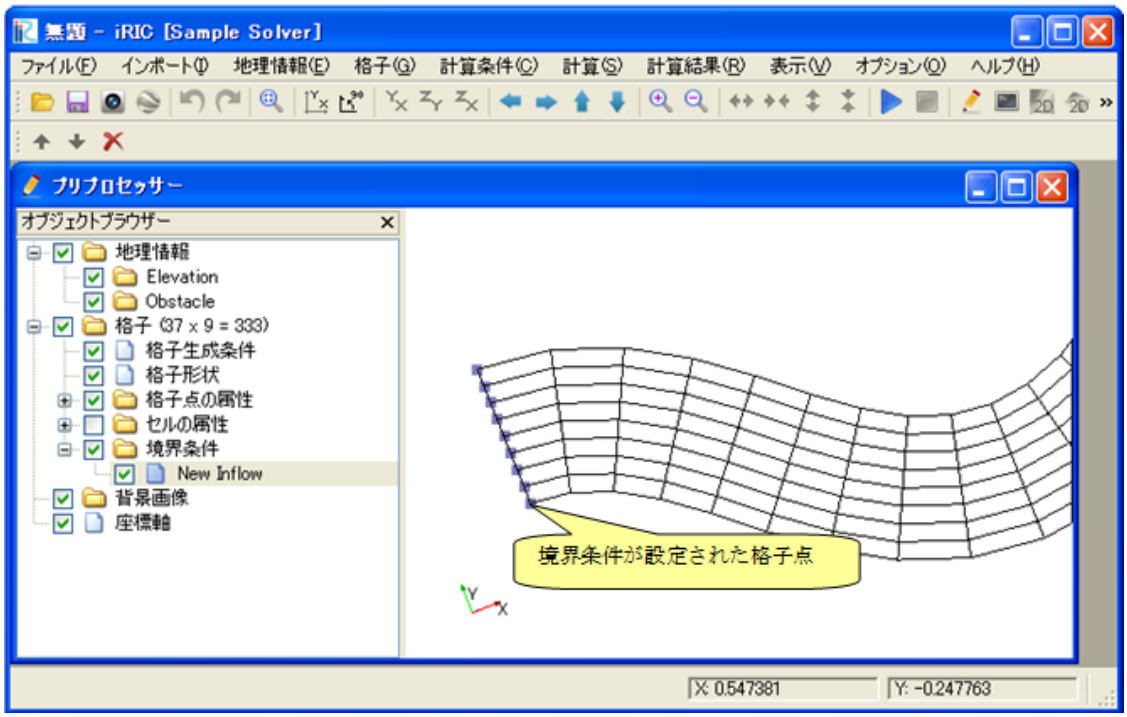

図 2.16　境界条件を設定した格子の表示例

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。

2.5 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 6) 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

2.5.1 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。Intel Fortran Compiler, gfortran でのコンパイル方法を *Fortran* **言語で** *iriclib, cgnslib* **とリンクしてビルドする方法** (ページ 256) 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

```
1  program SampleProgram
2    implicit none
3    include 'cgnslib_f.h'
4    include 'iriclib_f.h'
5
6    write(*,*) "Sample Program"
7    stop
8  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 6) 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。

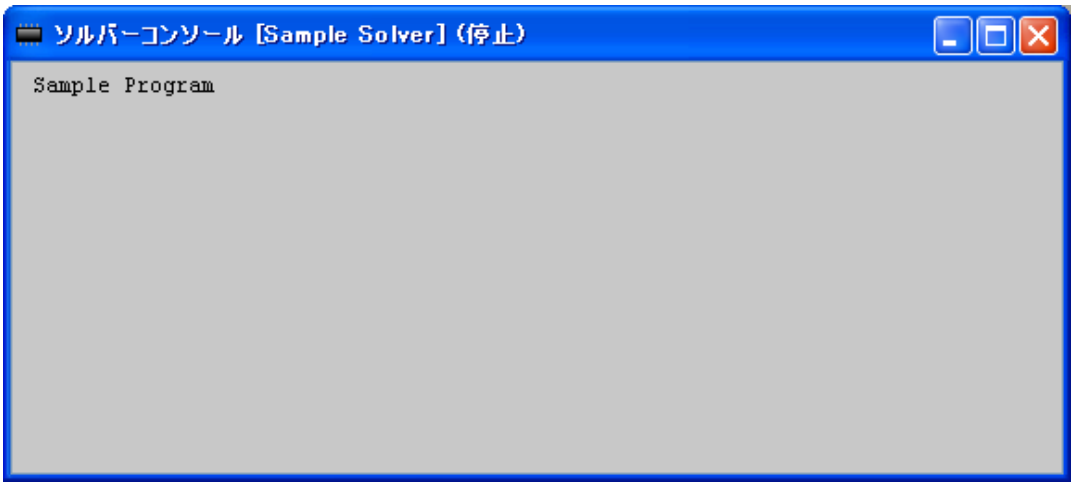

図 2.17　ソルバーコンソール表示例

2.5.2 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。Intel Fortran Compiler, gfortran での引数の取得方法を *Fortran* **プログラムでの引数の読み込み処理** (ページ 255) 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    include 'iriclib_f.h'
5    integer:: fin, ier
6    integer:: icount, istatus
```

(次のページに続く)

(前のページからの続き)

```
  character(200)::condFile

  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 20) 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 20) 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、*CGNS* **ファイルを開く** (ページ 115), **内部変数の初期化** (ページ 116), *CGNS* **ファイルを閉じる** (ページ 151) を参照してください。

2.5.3 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 6) 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'
  include 'iriclib_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)
```

(次のページに続く)

(前のページからの続き)

```
  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)

  call cg_iric_read_functionalsize_f("variableSurface", variable_surface_size, ier)
  allocate(variable_surface_time(variable_surface_size))
  allocate(variable_surface_elevation(variable_surface_size))
  call cg_iric_read_functional_f("variableSurface", variable_surface_time, variable_
↪surface_elevation, ier)

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)

  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)

  ! 格子点で定義された属性 のメモリを確保
  allocate(elevation(isize, jsize))
  allocate(obstacle(isize - 1, jsize - 1))

  ! 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)

  ! 流入口の数に従って、境界条件を保持するメモリを確保。
  allocate(inflow_element_count(inflow_count))
  allocate(discharge_type(inflow_count), discharge_constant(inflow_count))
  allocate(discharge_variable_size(inflow_count))

  ! 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
  inflow_element_max = 0
  do inflowid = 1, inflow_count
    ! 流入口に指定された格子点の数
    call cg_iric_read_bc_indicessize_f('inflow', inflowid, inflow_element_
↪count(inflowid))
    if (inflow_element_max < inflow_element_count(inflowid)) then
      inflow_element_max = inflow_element_count(inflowid)
    end if
    ! 流入口の時間依存の流入量のデータの数
    call cg_iric_read_bc_functionalsize_f('inflow', inflowid, 'FunctionalDischarge',␣
↪discharge_variable_size(inflowid), ier);
    if (discharge_variable_sizemax < discharge_variable_size(inflowid)) then
      discharge_variable_sizemax = discharge_variable_size(inflowid)
    end if
  end do

```

(次のページに続く)

(前のページからの続き)

```
  ! 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。
  allocate(inflow_element(inflow_count, 2, inflow_element_max))
  allocate(discharge_variable_time(inflow_count, discharge_variable_sizemax))
  allocate(discharge_variable_value(inflow_count, discharge_variable_sizemax))

  ! 境界条件の読み込み
  do inflowid = 1, inflow_count
    ! 流入口に指定された格子点
    call cg_iric_read_bc_indices_f('inflow', inflowid, inflow_
↪element(inflowid:inflowid,:,:), ier)
    ! 流入量の種類 (0 = 一定、1 = 時間依存)
    call cg_iric_read_bc_integer_f('inflow', inflowid, 'Type', discharge_
↪type(inflowid:inflowid), ier)
    ! 流入量 (一定)
    call cg_iric_read_bc_real_f('inflow', inflowid, 'ConstantDischarge', discharge_
↪constant(inflowid:inflowid), ier)
    ! 流入量 (時間依存)
    call cg_iric_read_bc_functional_f('inflow', inflowid, 'FunctionalDischarge',␣
↪discharge_variable_time(inflowid:inflowid,:), discharge_variable_
↪value(inflowid:inflowid,:), ier)
  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
```

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 13) 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、 **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 70) を参照してください。

また、計算条件、計算格子、境界条件の読み込みに使う関数の詳細については、**計算条件 (もしくは格子生成条件) の読み込み** (ページ 116), **計算格子の読み込み** (ページ 118), **境界条件の読み込み** (ページ 121) を参照してください。

2.5.4 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)
```

(次のページに続く)

(前のページからの続き)

```
    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、**時刻 (もしくはループ回数) の出力** (ページ 131), **計算結果の出力** (ページ 134) 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、**計算格子の出力 (計算開始後の格子)** (ページ 132) 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 **計算結果** (ページ 258) を参照してください。

2.6 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)

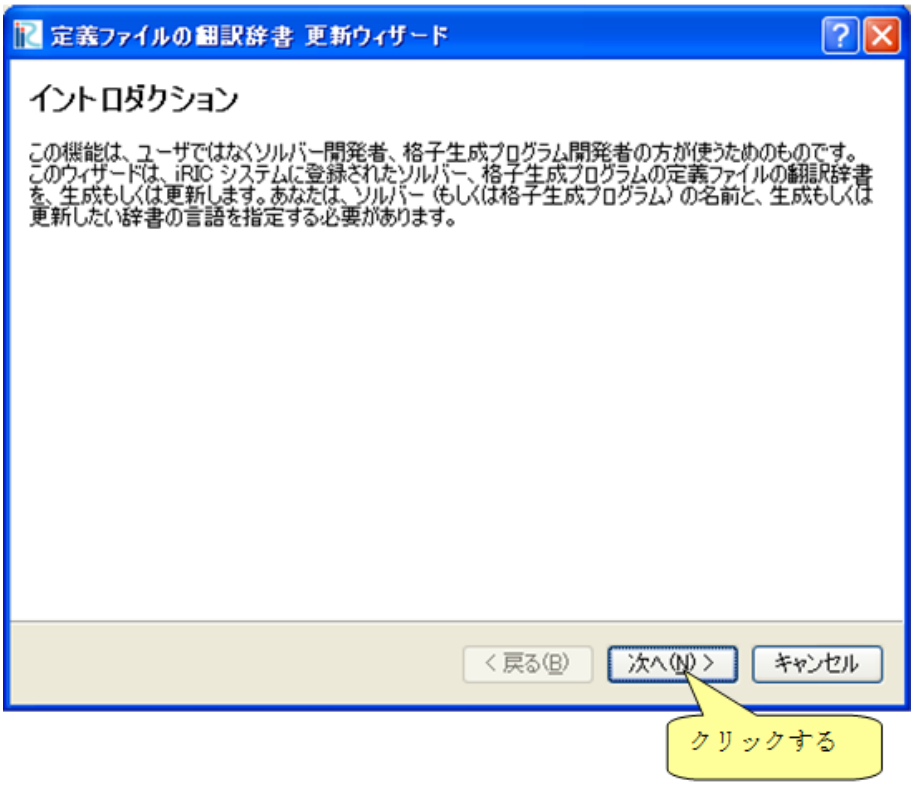

図 2.18　ソルバー定義ファイルの辞書更新ウィザード 表示例 (1 ページ目)

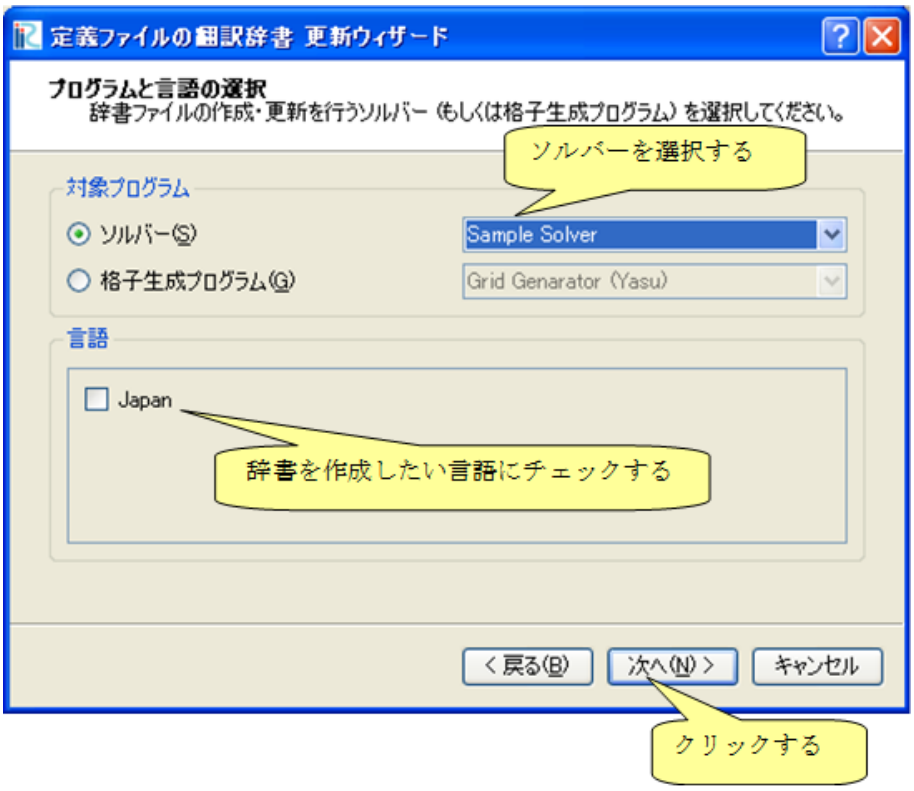

図 2.19　ソルバー定義ファイルの辞書更新ウィザード 表示例 (2 ページ目)

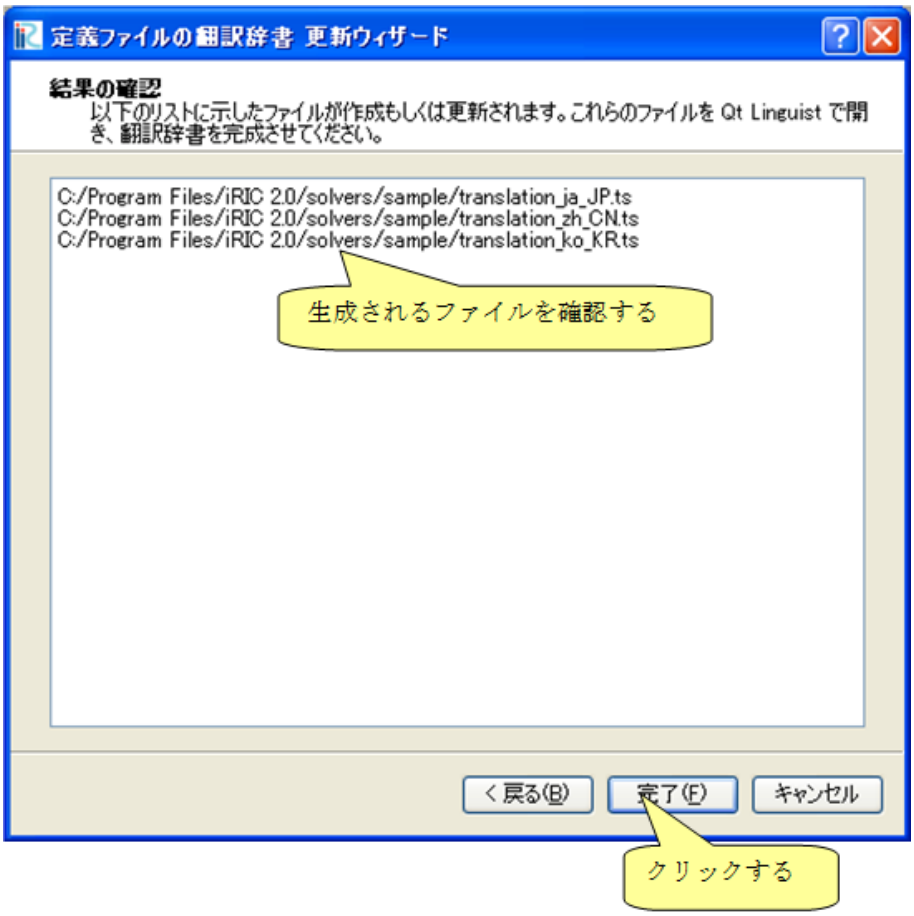

図 2.20　ソルバー定義ファイルの辞書更新ウィザード 表示例 (3 ページ目)

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/

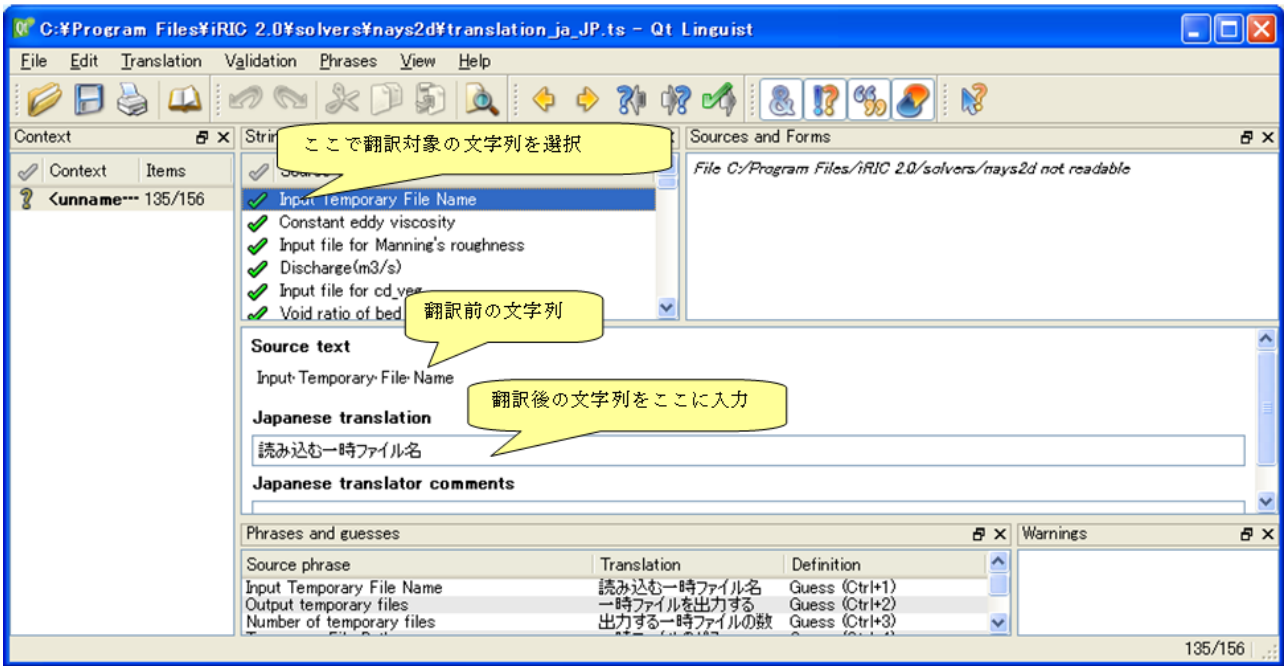

図 2.21　Qt Linguist 画面表示例

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。

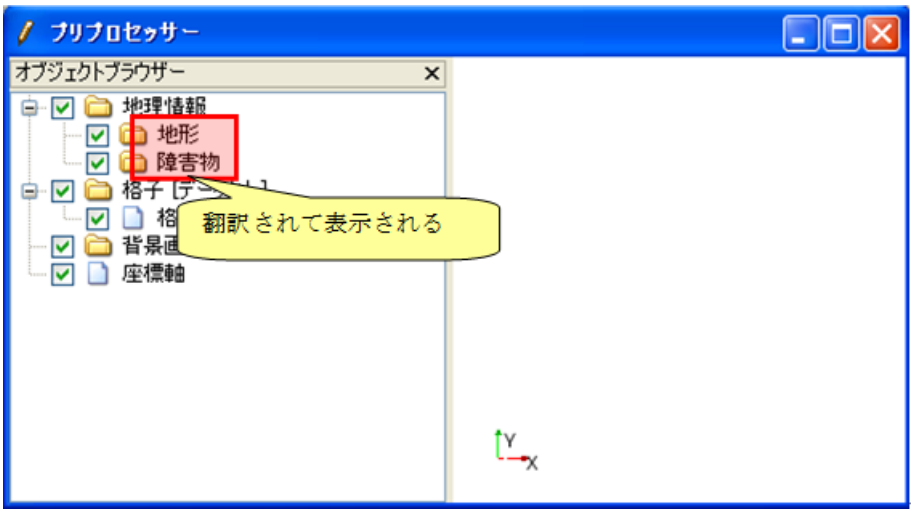

図 2.22　翻訳完了後のプリプロセッサ 表示例

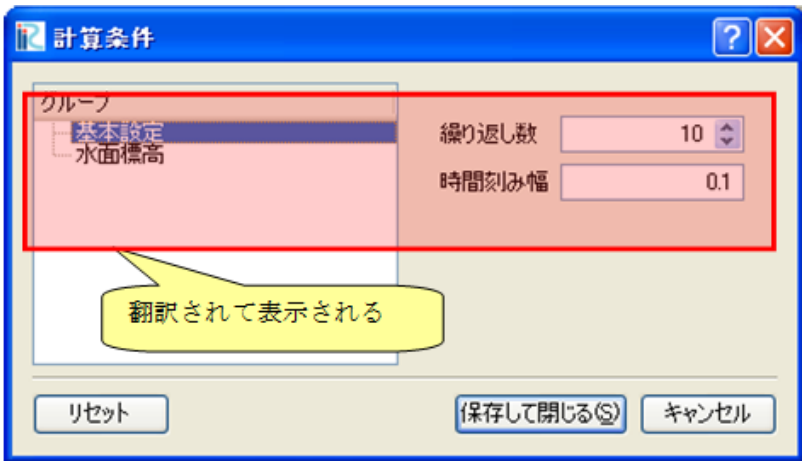

図 2.23　翻訳完了後の計算条件設定ダイアログ 表示例

2.7 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 6) 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。

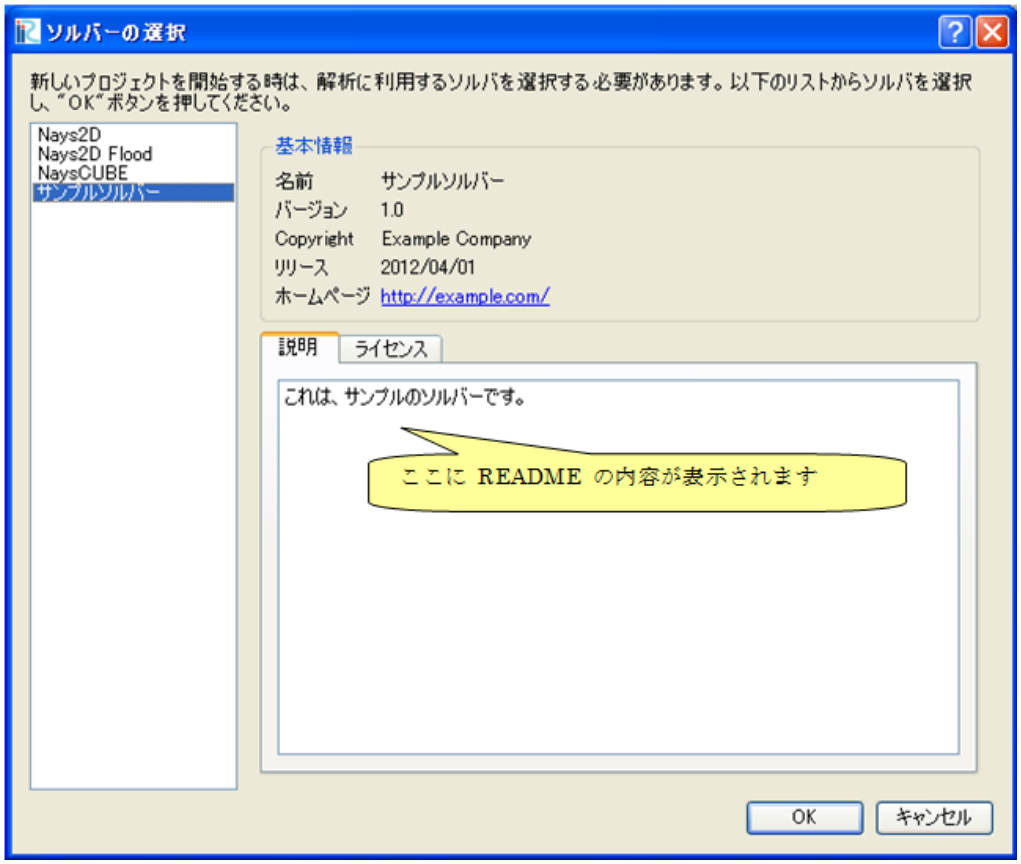

図 2.24　ソルバー選択ダイアログ 表示例

2.8 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 6) 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。

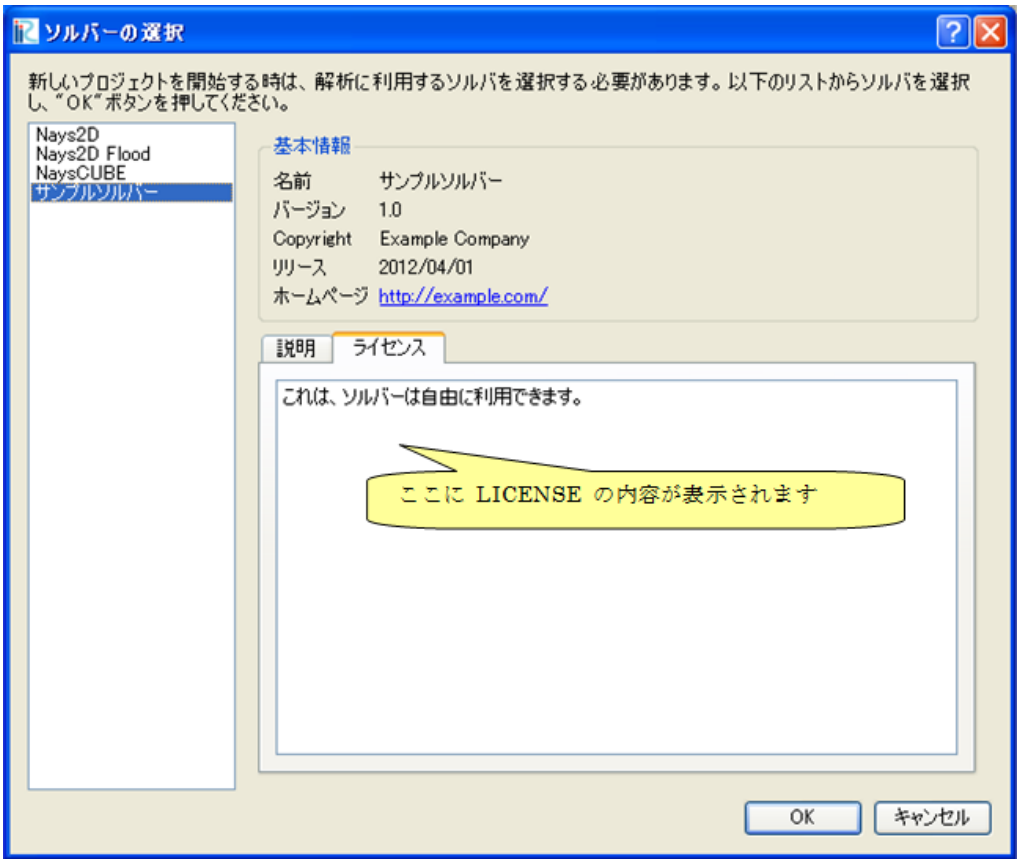

図 2.25　ソルバー選択ダイアログ 表示例

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 (**サブルーチン一覧** (ページ 152) 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)
```

(次のページに続く)

(前のページからの続き)

```

! （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
```

(次のページに続く)

(前のページからの続き)

```
  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
```

(次のページに続く)

(前のページからの続き)

```
  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 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 "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、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, Python, 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。

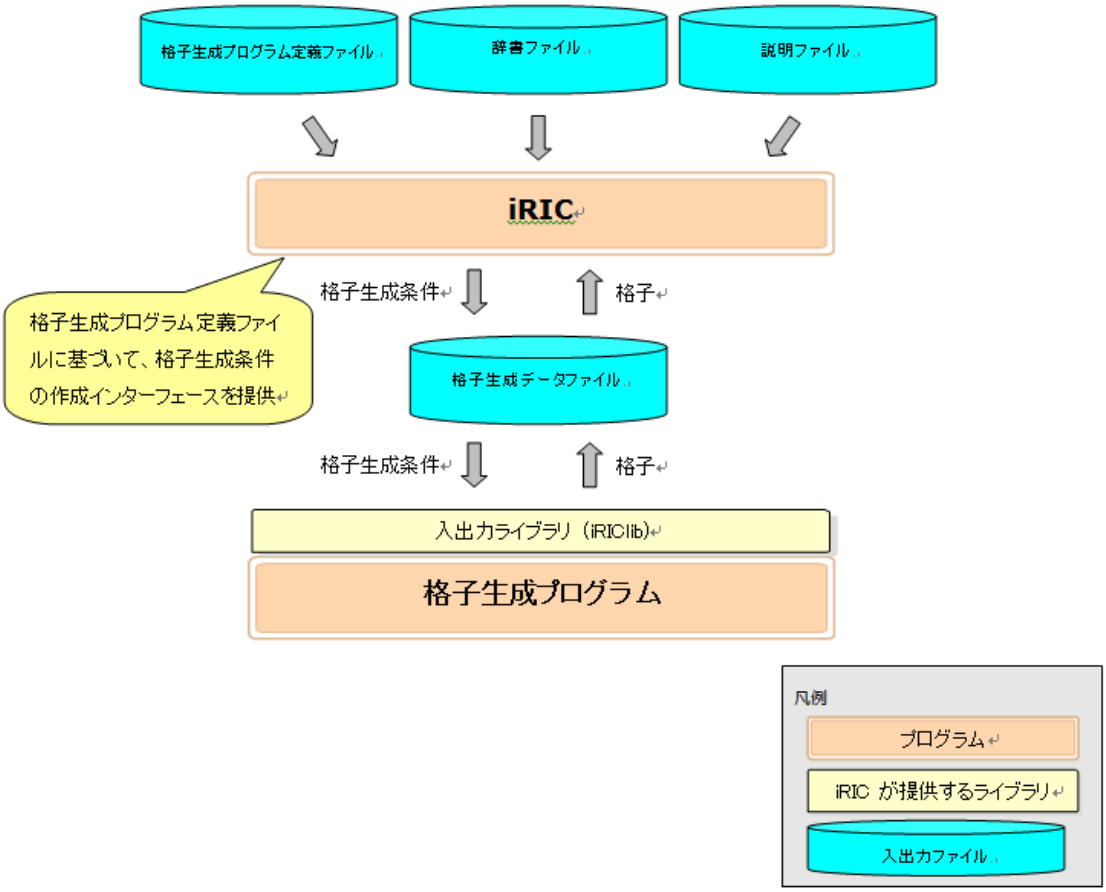

図 4.1　iRIC、格子生成プログラム、関連ファイルの関係図

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。

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 *XML* **の基礎** (ページ 110) 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 41) 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。

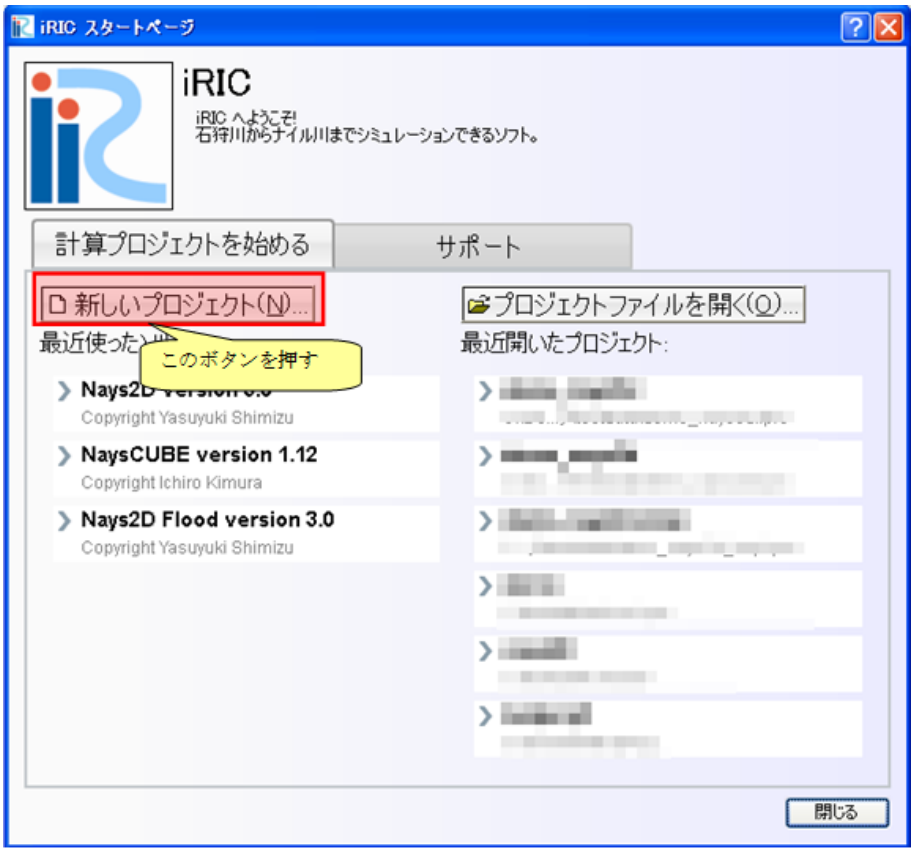

図 4.3　iRIC のスタートダイアログ

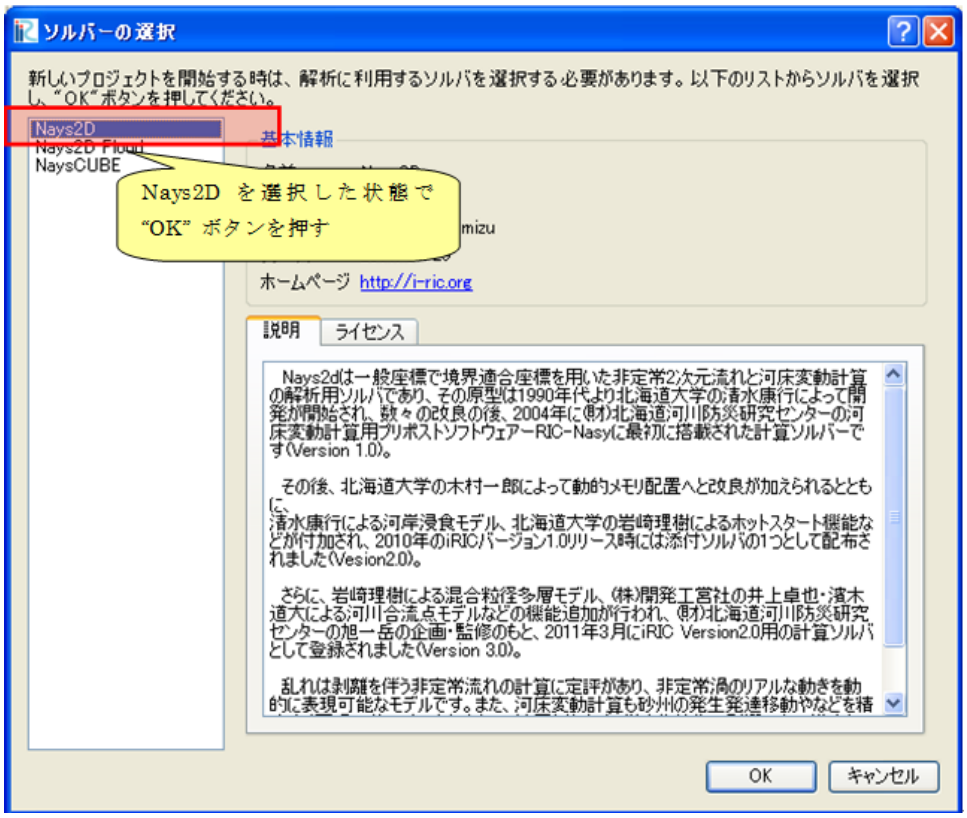

図 4.4　ソルバー選択ダイアログ

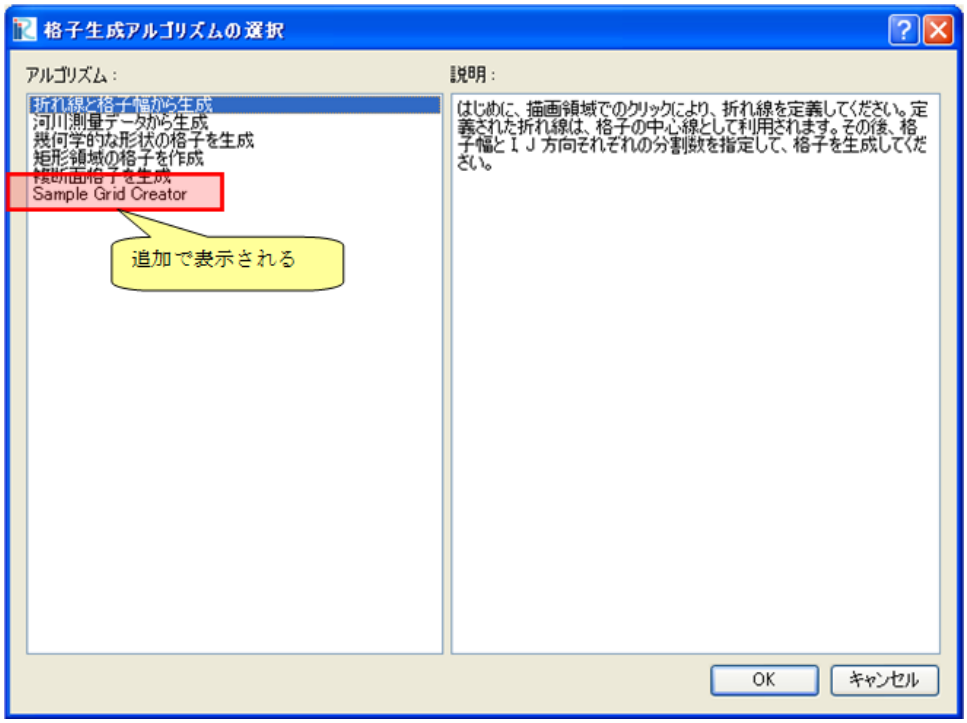

図 4.5　格子生成アルゴリズム選択ダイアログ

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 42) 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 42) 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。

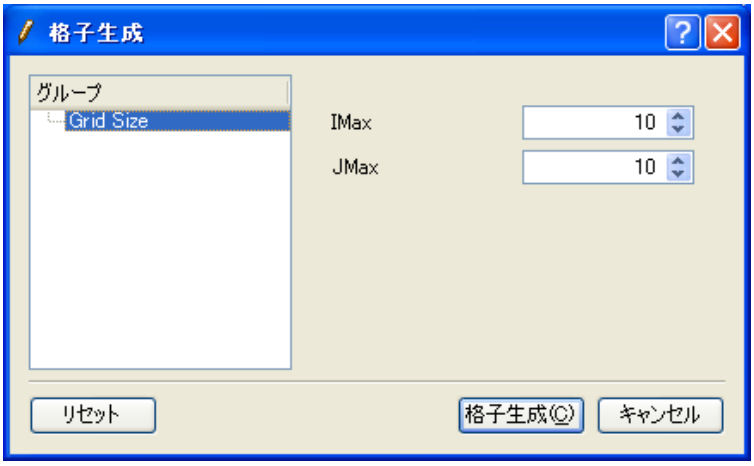

図 4.7　格子生成ダイアログ 表示例

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。

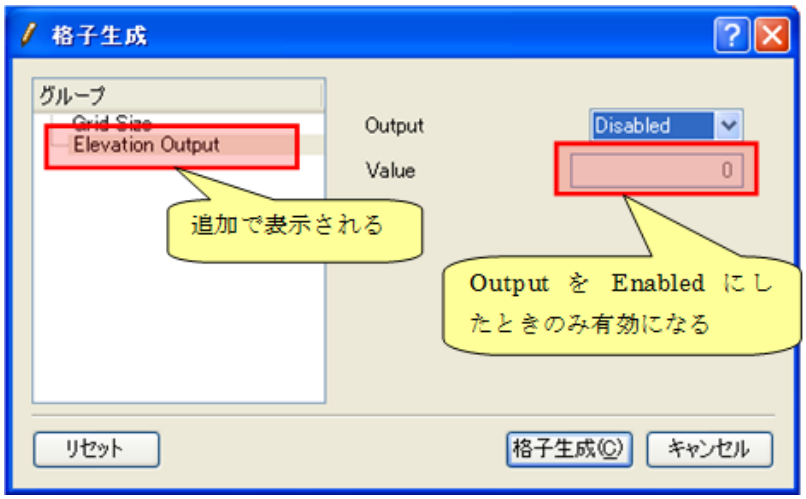

図 4.9　格子生成ダイアログ 表示例

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**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 70) 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、**計算条件・境界条件・格子生成条件の有効になる条件の定義例** (ページ 82) 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 **計算条件・境界条件・格子生成条件のダイアログのレイアウト定義例** (ページ 82) 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 45) 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>
```

(次のページに続く)

(前のページからの続き)

```
  <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 56) 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 41) 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 *Intel Fortran Compiler (Windows)* (ページ 256) 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。Intel Fortran compiler, gfortran での引数の取得方法を *Fortran* **プログラムでの引数の読み込み処理** (ページ 255) 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 50) 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、*CGNS* **ファイルを開く** (ページ 115), **内部変数の初期化** (ページ 116), *CGNS* **ファイルを閉じる** (ページ 151) 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 41) 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 42) 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 42) 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。

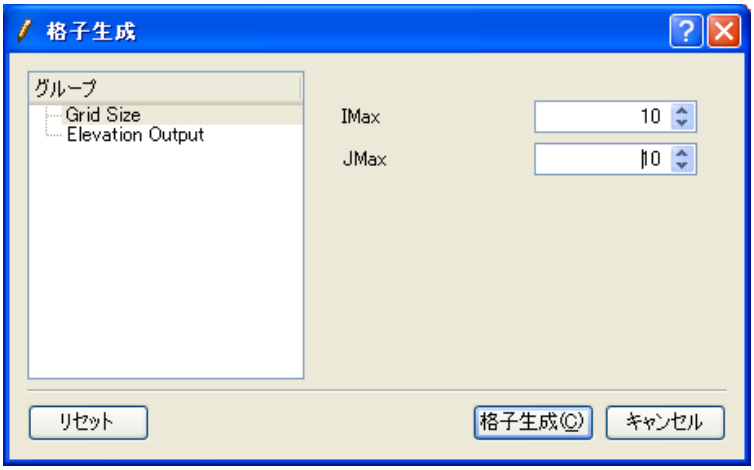

図 4.11　格子生成条件設定ダイアログ 表示例

"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。

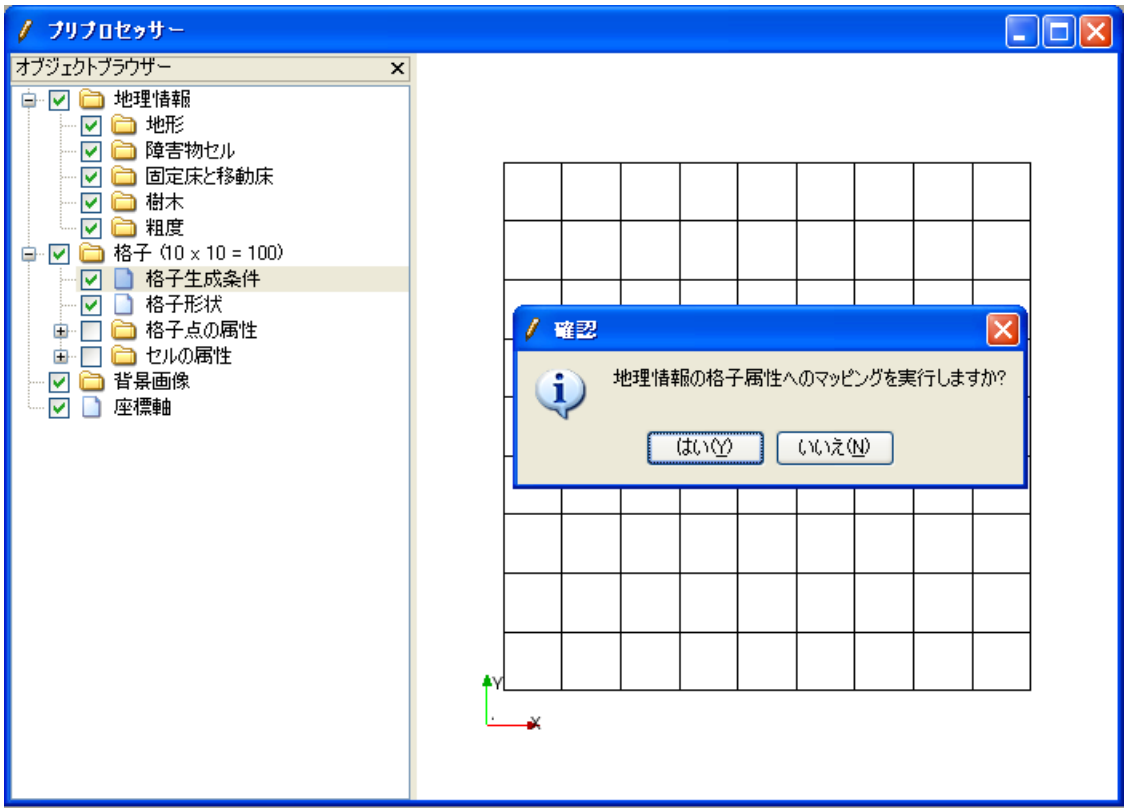

図 4.12　プリプロセッサ表示例

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、**計算格子の出力** (ページ 129) を参照してください。ただし、**計算格子の出力** (ページ 129) 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 41) 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 45) 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 51) 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、**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 70) 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、**計算条件 (もしくは格子生成条件) の読み込み** (ページ 116) 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
```

(次のページに続く)

(前のページからの続き)

```
      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 51) 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 **エラーコードの出力** (ページ 151) 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
```

(次のページに続く)

(前のページからの続き)

```
17
18    ! 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)
```

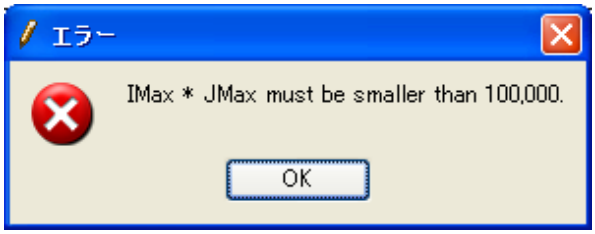

図 4.13　格子生成エラーダイアログ 表示例

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)

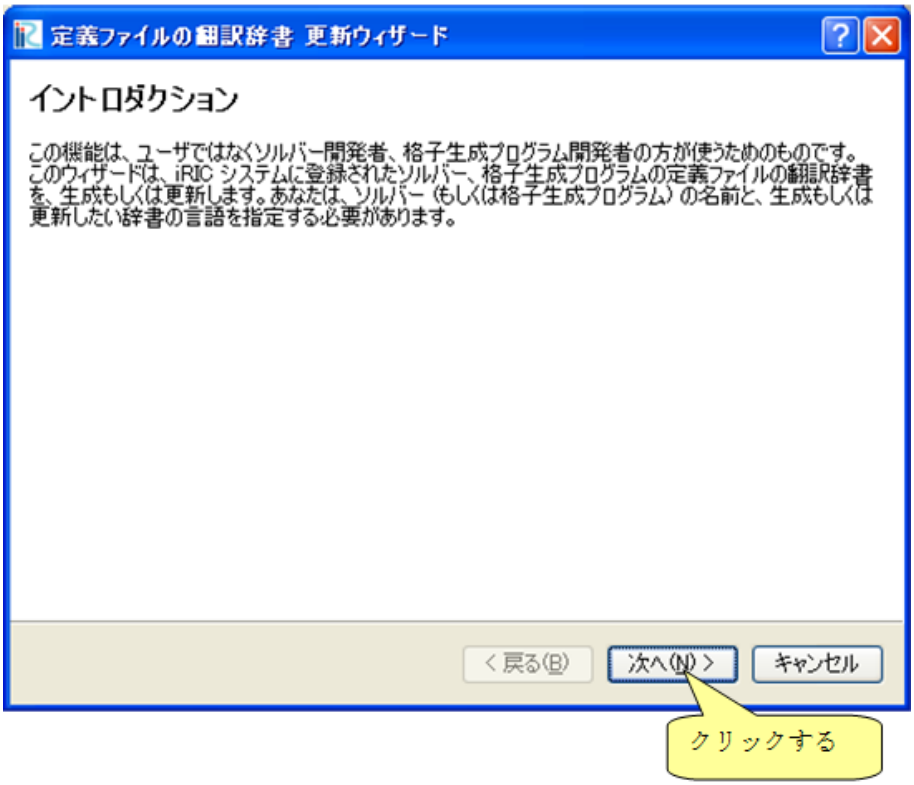

図 4.14　定義ファイルの翻訳辞書 更新ウィザード 表示例 (1 ページ目)

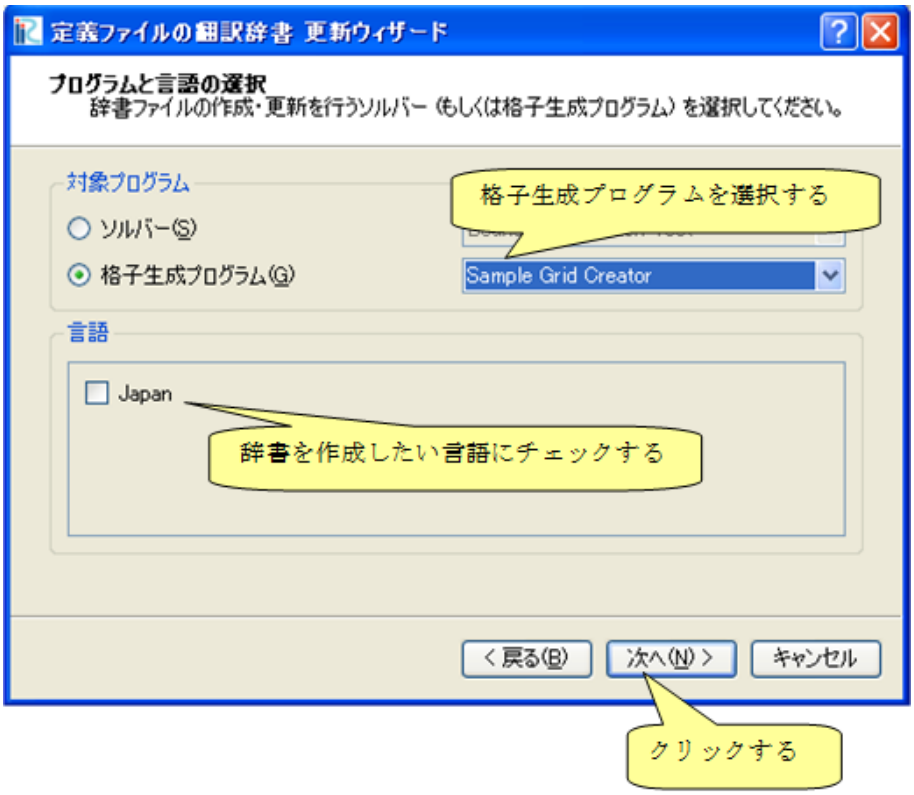

図 4.15　定義ファイルの翻訳辞書 更新ウィザード 表示例 (2 ページ目)

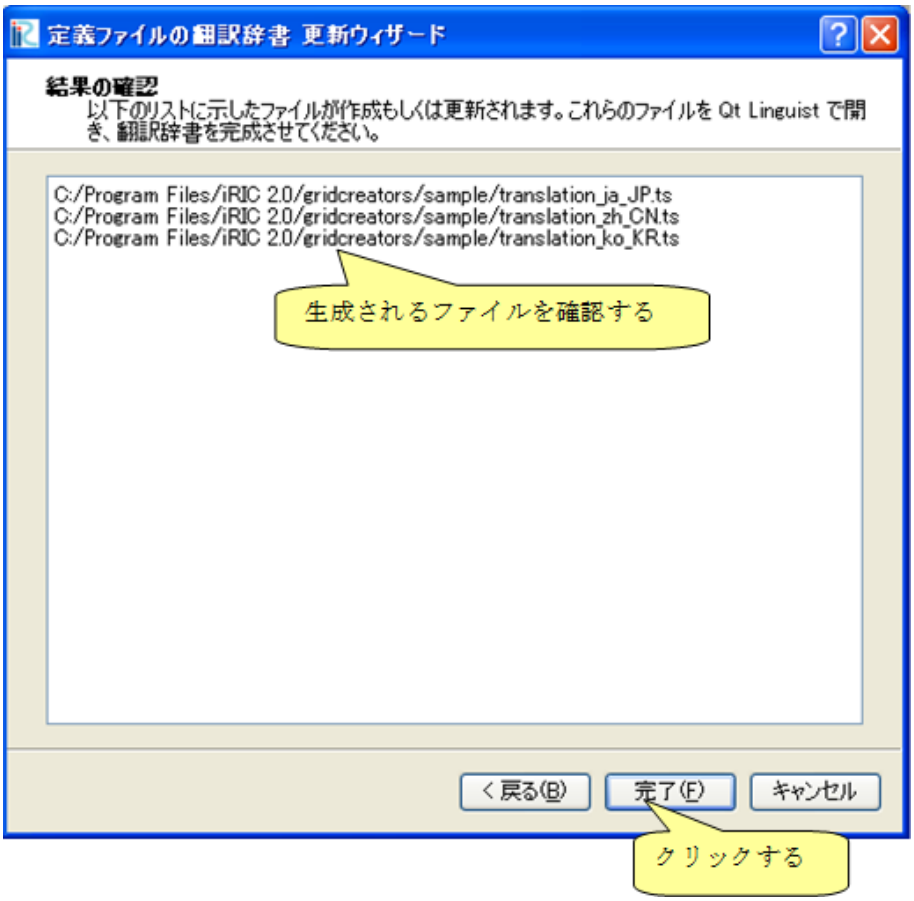

図 4.16　定義ファイルの翻訳辞書 更新ウィザード 表示例 (3 ページ目)

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>
```

(次のページに続く)

(前のページからの続き)

```
  <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/

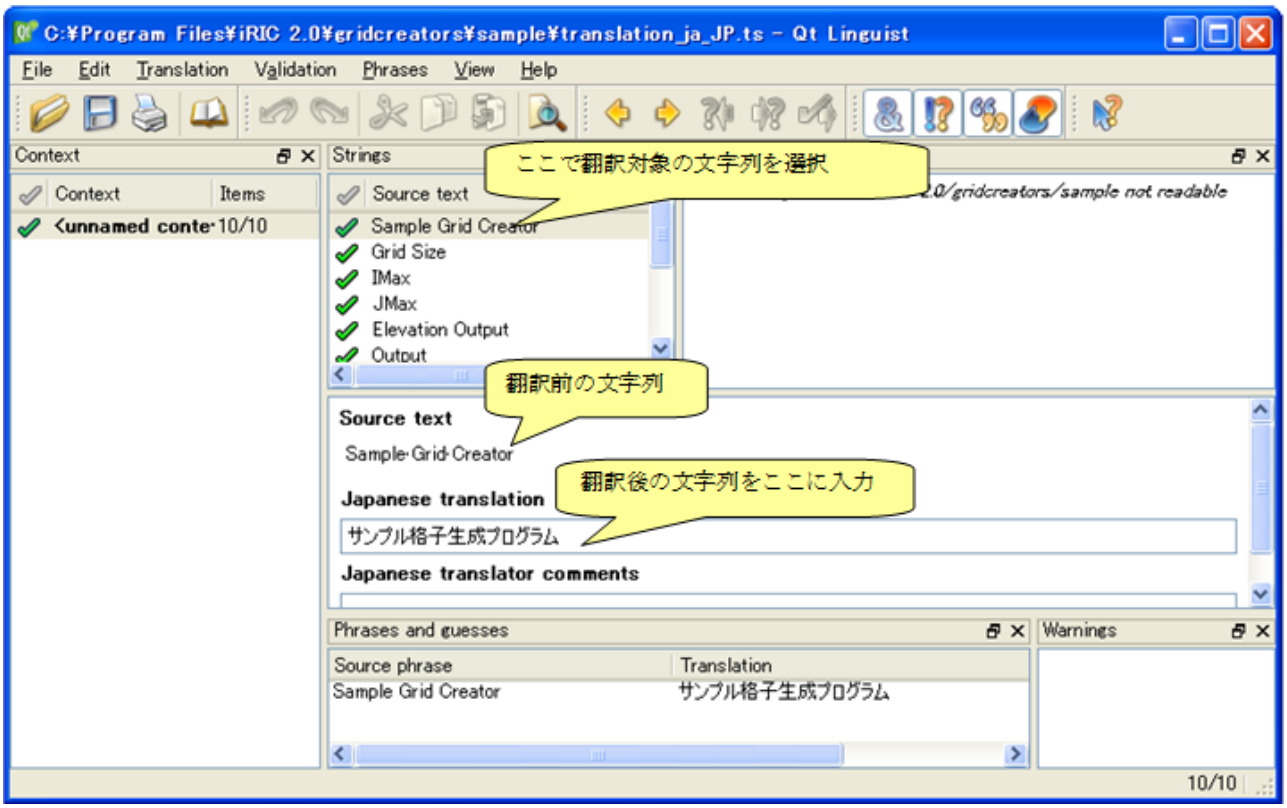

図 4.17　Qt Linguist 画面表示例

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。

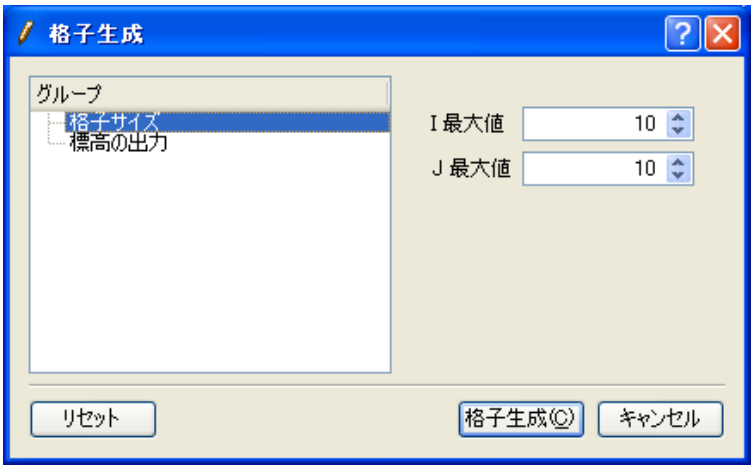

図 4.18　翻訳完了後の格子生成条件設定ダイアログ 表示例

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 41) 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。

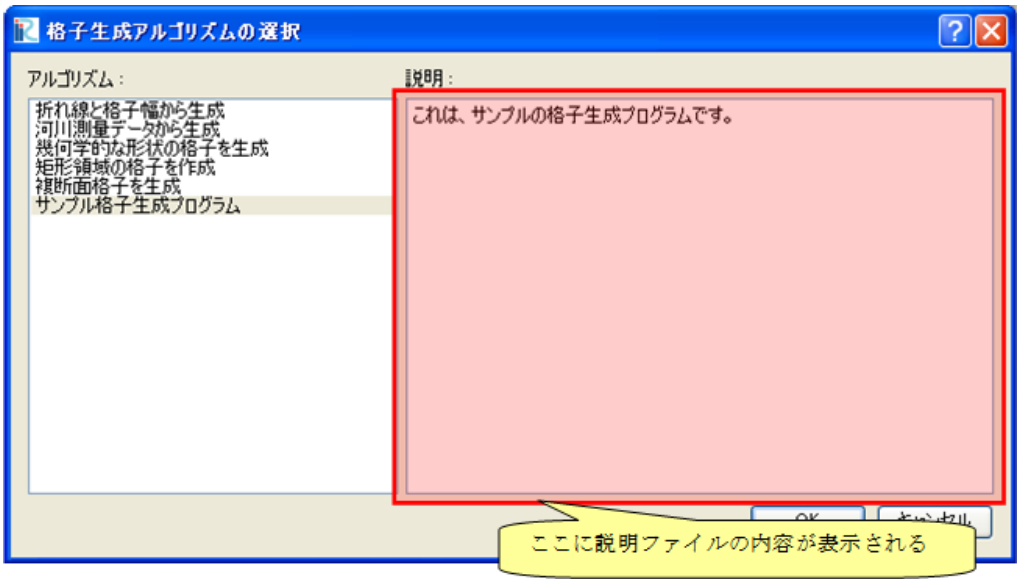

図 4.19　格子生成アルゴリズム選択ダイアログ 表示例

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>
```

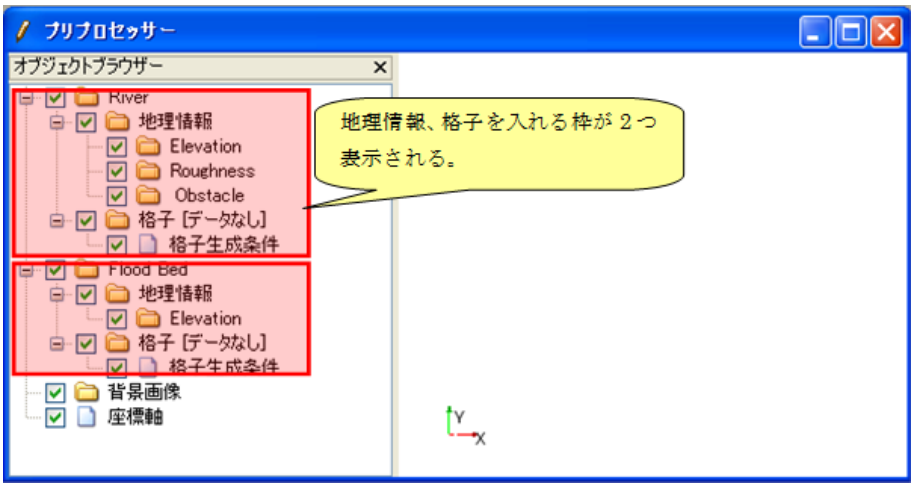

図 5.3　複数の格子を定義したソルバ定義ファイルを読み込んだ場合のプリプロセッサ 表示例

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**構造** (ページ 65) 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、 *iRIClib* **について** (ページ 113) 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**ソルバーの作成** (ページ 19), **格子生成プログラムの作成** (ページ 49) 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
CGNS ファイル名	読み込み用の CGNS 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 "cgns_filename" を指定
CGNS ファイル内の計算結果	計算結果の名前を選択	valueType に "result_gridNodeReal" などを指定

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>
```

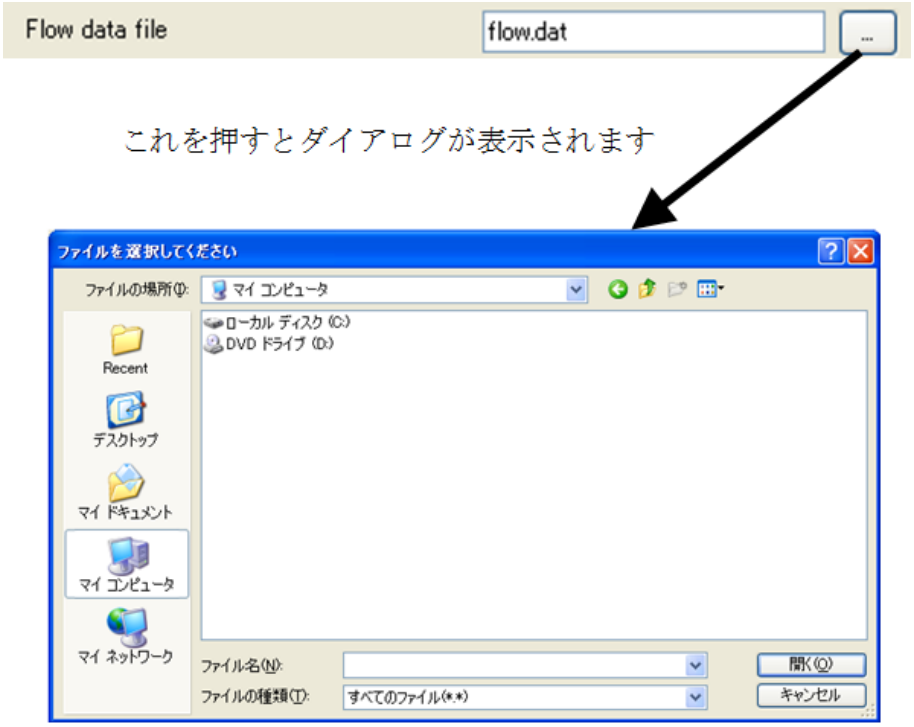

図 5.6　ファイル名 (読み込み用) の条件の表示例

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>
```

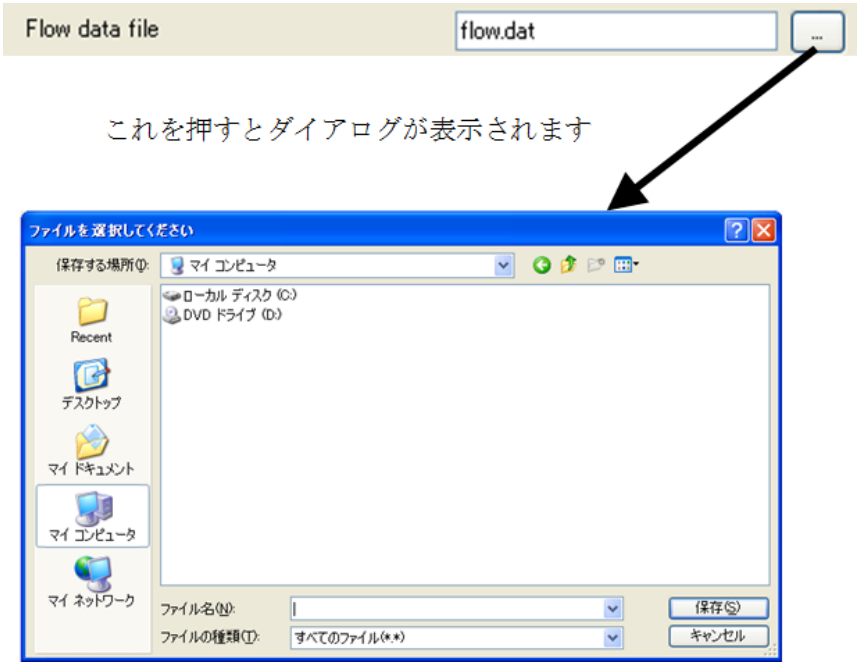

図 5.7　ファイル名 (書き込み用) の条件の表示例

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>
```

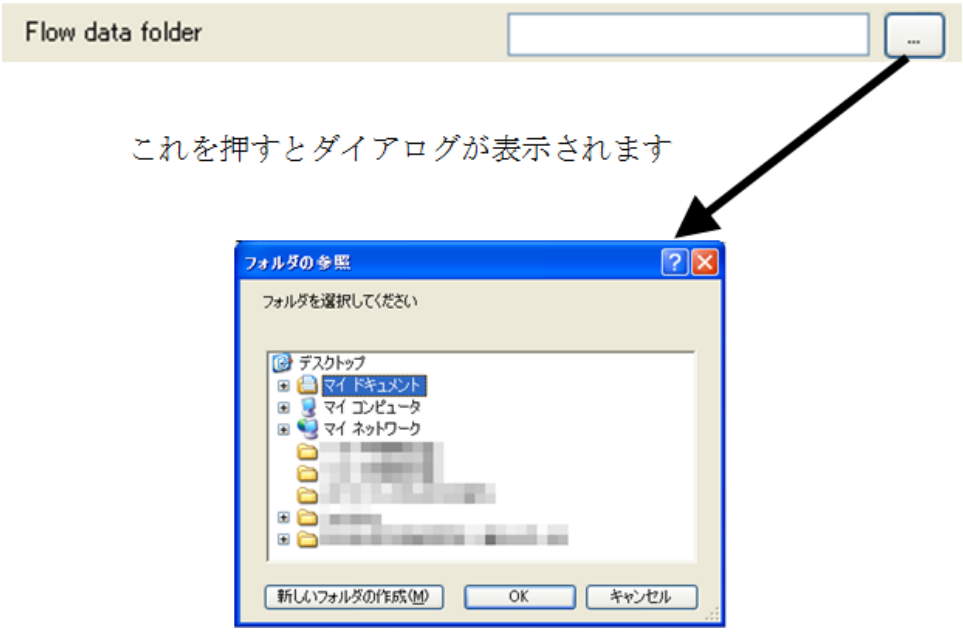

図 5.8　フォルダ名の条件の表示例

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>
```

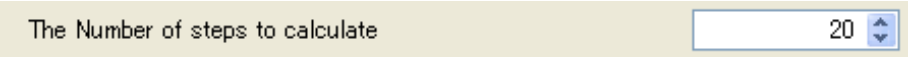

図 5.9　整数の条件の表示例

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>
```

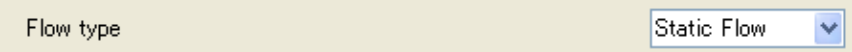

図 5.10　整数 (選択式) の条件の表示例

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>
```

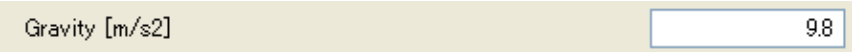

図 5.11　実数の条件の表示例

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>
```

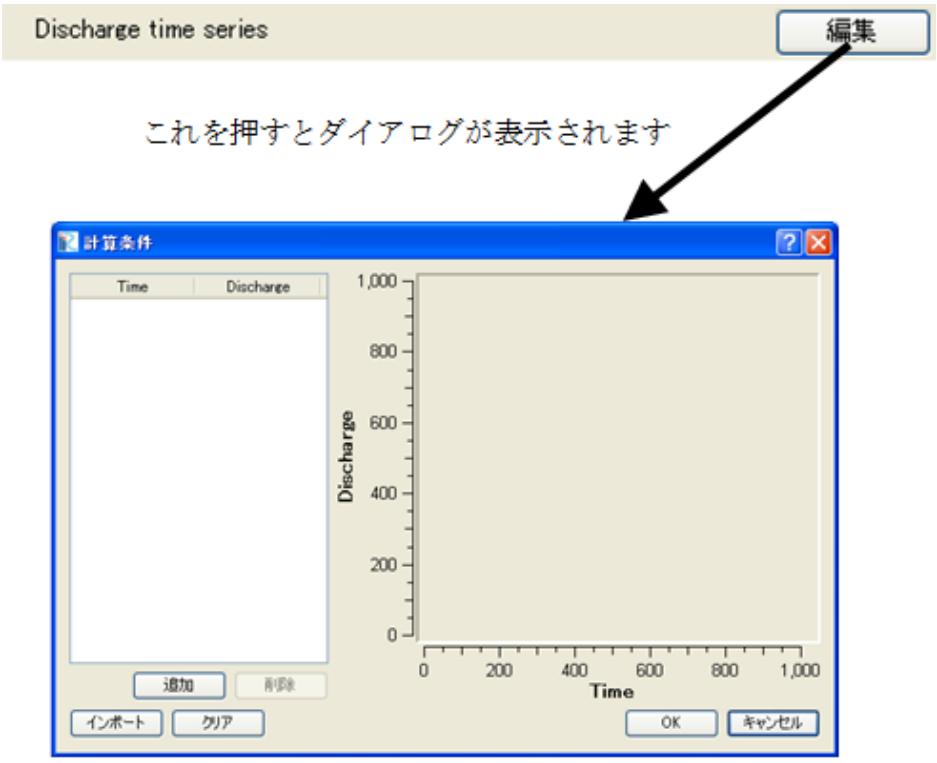

図 5.12　関数型の条件の表示例

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>
```

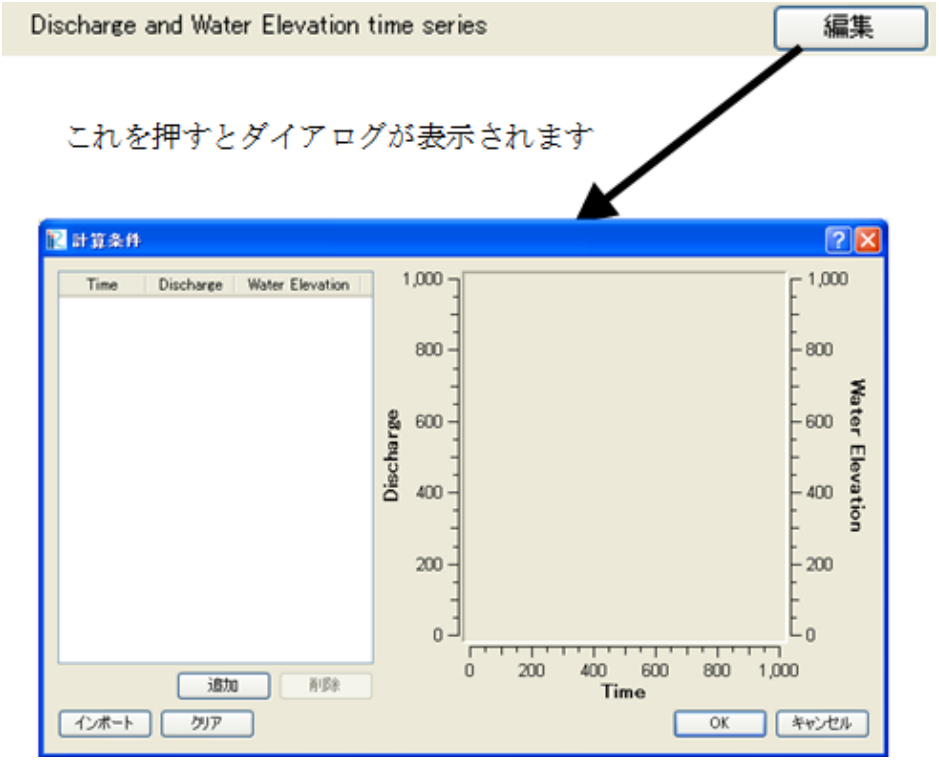

図 5.13　関数型 (複数の値) の条件の表示例

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
```

(次のページに続く)

(前のページからの続き)

```
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)
```

CGNS ファイル名 など

CGNS ファイル名と、CGNS ファイル内の計算結果は組み合わせて使用します。

CGNS ファイル名の入力欄は、 valueType に cgns_filename を指定することで作成できます。

CGNS ファイル内の計算結果は、 valueType に result_gridNodeReal などを指定し、 cgnsFile に CGNS ファイル名の入力欄に指定した name を指定することで作成できます。

リスト 5.29　CGNS ファイル名と CGNS ファイル内の計算結果の条件の定義例

```
<Item name="input_file" caption="CGNS file for input">
  <Definition valueType="cgns_filename" />
</Item>
<Item name="result_to_read" caption="Calculation result to read">
  <Definition valueType="result_gridNodeReal" cgnsFile="input_file" />
</Item>
```

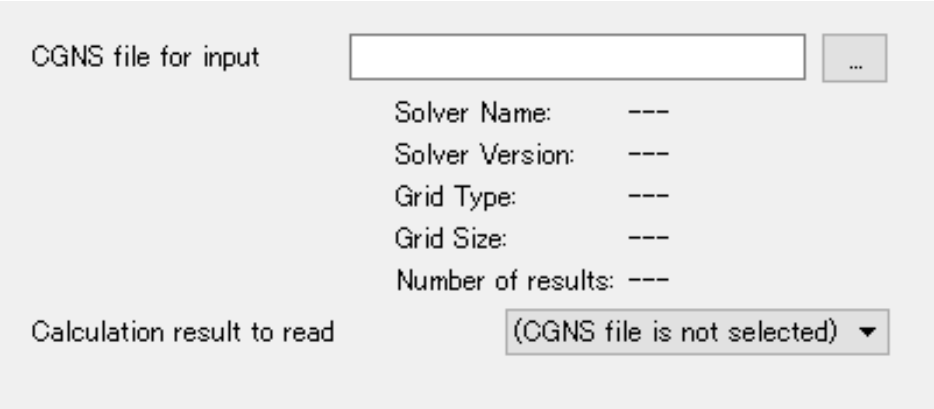

図 5.14　CGNS ファイル名と CGNS ファイル内の計算結果の条件の表示例

リスト 5.30　CGNS ファイル名と計算結果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):: cgnsName, resultName

call cg_iric_read_string_f("input_file", cgnsName, ier)
call cg_iric_read_string_f("result_to_read", resultName, ier)
```

リスト 5.31　CGNS ファイル名と計算結果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):: cgnsName, resultName

call cg_iric_read_bc_string_f("inflow", 1, "input_file", cgnsName, ier)
call cg_iric_read_bc_string_f("inflow", 1, "result_to_read", resultName, ier)
```

5.3.2 計算条件・境界条件・格子生成条件の有効になる条件の定義例

計算条件、格子生成条件、境界条件に関する有効になる条件の定義例を示します。ここで示すように、type が "and", "or" の条件を利用することによって複雑な条件を指定することができます。

var1 = 1

```
<Condition type="isEqual" target="var1" value="1" />
```

var1 = 1 and var2 > 3

```
<Condition type="or">
  <Condition type="isEqual" target="var1" value="1" />
  <Condition type="isGreaterThan" target="var2" value="3" />
</Condition>
```

(var1 = 1 or var2 < 5) and var3 = 100

```
<Condition type="and">
  <Condition type="or">
    <Condition type="isEqual" target="var1" value="1" />
    <Condition type="isLessThan" target="var2" value="5" />
  </Condition>
  <Condition type="isEqual" target="var3" value="100" />
</Condition>
```

5.3.3 計算条件・境界条件・格子生成条件のダイアログのレイアウト定義例

単純なレイアウト

Item 要素 のみを使って定義した単純なレイアウトの例をリスト 5.32 に、ダイアログでの表示例を 図 5.15 にそれぞれ示します。

リスト 5.32　単純なレイアウトの定義例

```
<Tab name="simple" caption="Simple">
  <Item name="jrep" caption="Periodic boundary condition">
    <Definition valueType="integer" default="0">
      <Enumeration value="0" caption="Disabled"/>
      <Enumeration value="1" caption="Enabled"/>
    </Definition>
  </Item>
  <Item name="j_wl" caption="Water surface at downstream">
    <Definition valueType="integer" default="1">
      <Enumeration value="0" caption="Constant value"/>
      <Enumeration value="1" caption="Uniform flow"/>
      <Enumeration value="2" caption="Read from file"/>
    </Definition>
  </Item>
  <Item name="h_down" caption="   Constant value (m)">
    <Definition valueType="real" default="0" />
  </Item>
  <Item name="j_slope" caption="   Slope for uniform flow">
    <Definition valueType="integer" default="0">
      <Enumeration value="0" caption="Calculated from geographic data"/>
      <Enumeration value="1" caption="Constant value"/>
    </Definition>
  </Item>
  <Item name="bh_slope" caption="   Slope value at downstream">
    <Definition valueType="real" default="0.001">
    </Definition>
  </Item>
  <Item name="j_upv" caption="Velocity at upstream">
    <Definition valueType="integer" default="1">
      <Enumeration value="1" caption="Uniform flow"/>
      <Enumeration value="2" caption="Calculated from upstream depth"/>
    </Definition>
  </Item>
  <Item name="j_upv_slope" caption="   Slope for uniform flow">
    <Definition valueType="integer" default="0">
      <Enumeration value="0" caption="Calculated from geographic data"/>
      <Enumeration value="1" caption="Constant value"/>
    </Definition>
  </Item>
  <Item name="upv_slope" caption="   Slope value at upstream">
    <Definition valueType="real" default="0.001">
    </Definition>
```

(次のページに続く)

(前のページからの続き)

```
  </Item>
</Tab>
```

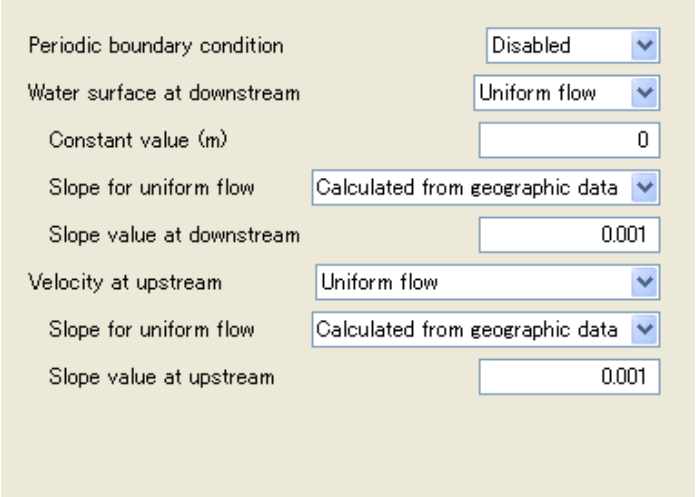

図 5.15　単純なレイアウトのダイアログの表示例

グループボックスを利用したレイアウト

グループボックスを利用したレイアウトの例をリスト 5.33 に、ダイアログでの表示例を図 5.16 にそれぞれ示します。

GroupBox 要素に Item 要素を入れることで、グループに分けて項目を表示できます。

リスト 5.33　グループボックスを利用したレイアウトの定義例

```
<Tab name="grouping" caption="Group">
  <Item name="g_jrep" caption="Periodic boundary condition">
    <Definition valueType="integer" default="0">
      <Enumeration value="0" caption="Disabled"/>
      <Enumeration value="1" caption="Enabled"/>
    </Definition>
  </Item>
  <GroupBox caption="Water surface at downstream">
    <Item name="g_j_wl" caption="Basic Setting">
      <Definition valueType="integer" default="1">
        <Enumeration value="0" caption="Constant value"/>
        <Enumeration value="1" caption="Uniform flow"/>
        <Enumeration value="2" caption="Read from file"/>
      </Definition>
    </Item>
    <Item name="g_h_down" caption="Constant value (m)">
      <Definition valueType="real" default="0" />
    </Item>
```

(次のページに続く)

(前のページからの続き)

```
    <Item name="g_j_slope" caption="Slope for uniform flow">
      <Definition valueType="integer" default="0">
        <Enumeration value="0" caption="Calculated from geographic data"/>
        <Enumeration value="1" caption="Constant value"/>
      </Definition>
    </Item>
    <Item name="g_bh_slope" caption="Slope value at downstream">
      <Definition valueType="real" default="0.001">
      </Definition>
    </Item>
  </GroupBox>
  <GroupBox caption="Velocity at upstream">
    <Item name="g_j_upv" caption="Basic Setting">
      <Definition valueType="integer" default="1">
        <Enumeration value="1" caption="Uniform flow"/>
        <Enumeration value="2" caption="Calculated from upstream depth"/>
      </Definition>
    </Item>
    <Item name="g_j_upv_slope" caption="Slope for uniform flow">
      <Definition valueType="integer" default="0">
        <Enumeration value="0" caption="Calculated from geographic data"/>
        <Enumeration value="1" caption="Constant value"/>
      </Definition>
    </Item>
    <Item name="g_upv_slope" caption="Slope value at upstream">
      <Definition valueType="real" default="0.001">
      </Definition>
    </Item>
  </GroupBox>
</Tab>
```

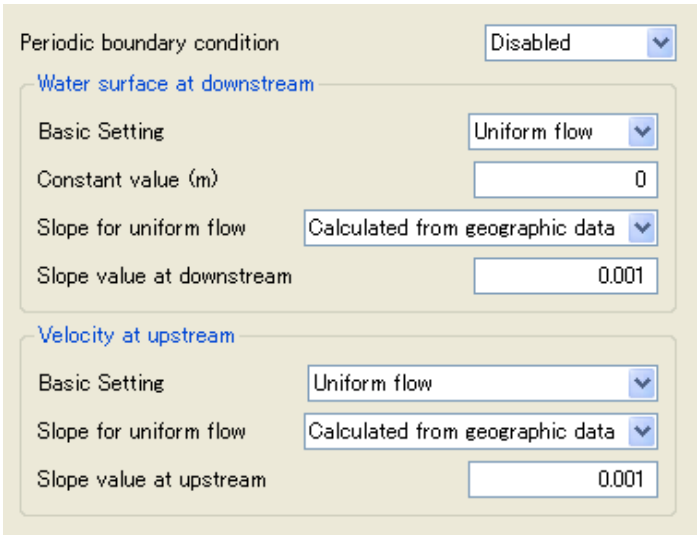

図 5.16　グループボックスを利用したレイアウトのダイアログの表示例

自由なレイアウト

GridLayout 要素を利用することで、自由なレイアウトを実現した例をリスト 5.34 に、ダイアログでの表示例を図 5.17 にそれぞれ示します。

GridLayout (表形式のレイアウト), HBoxLayout (水平に並べるレイアウト), VBoxLayout(垂直に並べるレイアウト) を使うことで、自由に要素を配置できます。また、これらのレイアウトの中では Item では caption 属性は指定せず、Label 要素でラベルを表示します。

GridLayout, HBoxLayout, VBoxLayout は入れ子にできます。また、その中で GroupBox を利用することもできます。

リスト 5.34　自由なレイアウトの定義例

```
<Tab name="roughness" caption="Roughness">
  <Item name="diam" caption="Diameter of uniform bed material (mm)">
    <Definition valueType="real" default="0.55" />
  </Item>
  <Item name="j_drg" caption="Bed roughness">
    <Definition valueType="integer" default="0">
      <Enumeration value="0" caption="Calculated from bed material"/>
      <Enumeration value="1" caption="Constant value"/>
      <Enumeration value="2" caption="Read from file"/>
    </Definition>
  </Item>
  <GroupBox caption="Manning's roughness parameter">
    <GridLayout>
      <Label row="0" col="0" caption="Low water channel" />
      <Item row="1" col="0" name="sn_l">
        <Definition valueType="real" default="0.01" />
      </Item>
      <Label row="0" col="1" caption="Flood channel" />
      <Item row="1" col="1" name="sn_h">
        <Definition valueType="real" default="0.01" />
      </Item>
      <Label row="0" col="2" caption="Fixed bed" />
      <Item row="1" col="2" name="sn_f">
        <Definition valueType="real" default="0.01" />
      </Item>
    </GridLayout>
  </GroupBox>
  <Item name="snfile" caption="Input file for Manning's roughness">
    <Definition valueType="filename" default="Select File" />
  </Item>
</Tab>
```

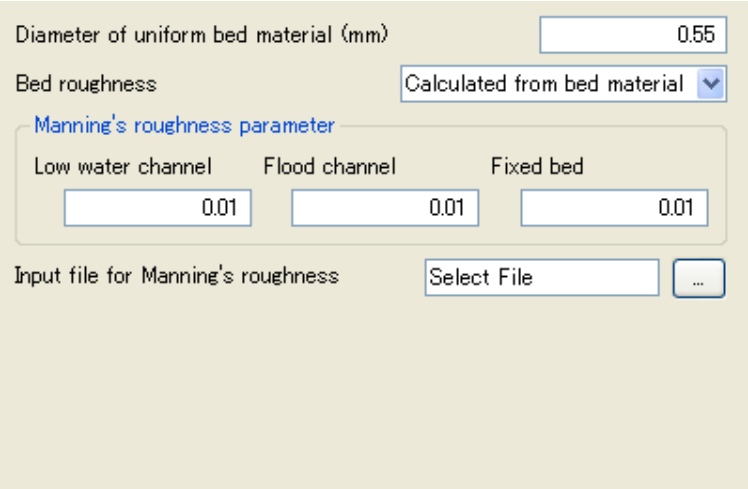

図 5.17　自由なレイアウトのを利用したレイアウトのダイアログの表示例

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 5.35　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 5.36　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 5.37　Condition の定義例 1

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

リスト 5.38　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
condition-Type	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
Condi-tion		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 5.39　Definition の定義例 1

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

リスト 5.40　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 70) 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
noSort	true もしくは false		テーブルを Param の値でソートしたくない場合は true を指定
cgnsFile	文字列		参照先の CGNS ファイルの名前

表 5.11　valueType の値

値	意味
integer	整数
real	実数
string	文字列
functional	関数型
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
cgns_filename	CGNS ファイル名
foldername	フォルダ名
result_gridNodeInteger	格子点で定義された整数の計算結果の選択
result_gridNodeReal	格子点で定義された実数の計算結果の選択
result_gridCellInteger	格子セルで定義された整数の計算結果の選択
result_gridCellReal	格子セルで定義された実数の計算結果の選択
result_gridEdgeIInteger	I 方向のエッジ定義された整数の計算結果の選択
result_gridEdgeIReal	I 方向のエッジで定義された実数の計算結果の選択
result_gridEdgeJInteger	J 方向のエッジ定義された整数の計算結果の選択
result_gridEdgeJReal	J 方向のエッジで定義された実数の計算結果の選択
result_baseIterativeInteger	グローバルな整数の計算結果の選択
result_baseIterativeReal	グローバルな実数の計算結果の選択

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。
Param		関数の横軸にする値。valueType が functional の場合のみ指定できる。
Value		関数の縦軸にする値。valueType が functional の場合のみ指定できる。

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 5.41　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 5.42　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 5.43　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 75) 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 5.44　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 5.45　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 5.46　GridGeneratingCondition の定義例

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

    （略）

  </Tab>
```

(次のページに続く)

(前のページからの続き)

```
  <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 5.47　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 **自由なレイアウト** (ページ 86) 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 5.48　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 5.49　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 layout_groupbox_example 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 5.50　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 5.51　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 70) 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 5.52　Label の定義例

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

定義例は **自由なレイアウト** (ページ 86) 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 5.53　Param の定義例

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

定義例は **関数型** (ページ 77) 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 5.54　SolverDefinition の定義例

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

(次のページに続く)

(前のページからの続き)

```
  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、**ソルバーのバージョンアップ時の注意点** (ページ 109) 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 5.55　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 5.56　Value の定義例

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

定義例は **関数型** (ページ 77) 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
span	true or false		値を区間ごとに定義するなら 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 5.57　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

5.5 ソルバーのバージョンアップ時の注意点

ソルバーをバージョンアップする際は、ソルバーそのものを改変するとともに、ソルバー定義ファイルを更新する必要があります。ソルバー定義ファイルを更新する際の注意点について、以下に示します。

- SolverDefinition 要素の name 属性は編集しないでください。 name 属性が異なると、iRIC は別のソルバーとみなし、過去のソルバー用に作成したプロジェクトファイルは読み込めなくなります。
- SolverDefinition 要素の caption 属性を変更してください。caption 属性は、ソルバーの名前とバージョンの情報を保持する任意の文字列ですので、例えば"Sample Solver 1.0"、"Sample Solver 3.2 Beta", "Sample Solver 3.0 RC1" など任意の形式でバージョンを記述できます。下で示す version 属性とは独立に設定できます。
- SolverDefinition 要素の version 属性を、 表 5.49 のポリシーに従って変更して下さい。

なお、バージョン番号は、例えば "3.1.4" などのように、 "." で区切られた番号から構成されます。先頭から順に "メジャー番号"、 "マイナー番号"、 "修正番号" と呼びます。修正番号は省略可能です。

表 5.49　更新するバージョン番号の構成要素

更新する要素	更新する条件	例
メジャー番号	入力する格子・計算条件が過去のものと全く互換性のなくなる、大きな変更を行った場合	2.1 –> 3.0
マイナー番号	入力する格子・計算条件に項目を追加したが、入力されなかった場合はデフォルト値で実行すれば問題ない変更を行った場合	2.1 –> 2.2
修正番号	入力する格子・計算条件には全く変更がなく、内部のアルゴリズムの変更やバグの修正のみを行った場合	2.1 –> 2.1.1

なお、バージョン番号が異なるプロジェクトファイルの互換性については、iRIC では以下のように扱われます。

- メジャー番号が異なるソルバーのプロジェクトファイルは、互換性がない。
- マイナー番号が異なるだけのソルバーのプロジェクトファイルは、より古いものであれば互換性がある。
- 修正番号が異なるだけのソルバーのプロジェクトファイルは、常に互換性がある。

ソルバーとプロジェクトファイルのバージョン番号と、互換性の例を図 5.18 に示します。

		ソルバーのバージョン			
		1.0	2.0	2.1	2.1.1
プロジェクトファイルのバージョン	1.0	○	×	×	×
	2.0	×	○	○	○
	2.1	×	×	○	○
	2.1.1	×	×	○	○

図 5.18　ソルバー・プロジェクトファイルのバージョン番号と互換性の例

バージョン番号の増やし方の基本方針は 表 5.49 で示したとおりですが、最終的には、プロジェクトファイルの互換性を考慮してソルバー開発者の方がご判断ください。

また、同一のソルバーの異なるバージョンを、同じ環境に混在させたい場合は、solvers フォルダの下に、異なる名前でフォルダを作成し、その中にそれぞれ異なるバージョンのソルバーを配置することができます。フォルダの名前は、ソルバーの名前とは独立につけることができます。

5.6 XML の基礎

この節では、iRIC で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に用いられている XML という言語の基礎について説明します。

5.6.1 要素の書き方

要素の開始は、要素名を "<" と ">" で囲って記述します。

要素の終了は、要素名を "</" と ">" で囲って記述します。

Item 要素の記述例を リスト 5.58 に示します。

リスト 5.58　要素の記述例

```
<Item>

</Item>
```

要素は、以下を持つことができます。

- 子要素
- 属性

要素は、同じ名前の子要素を複数持つことができます。一方、属性は同じ名前の属性は 1 つしか持てません。子要素 SubItem と、属性 name を持つ Item 要素の記述例をリスト 5.59 に示します。

リスト 5.59　要素の記述例

```
<Item name="sample">
  <SubItem>
  </SubItem>

  <SubItem>
  </SubItem>
</Item>
```

また、子要素を持たない要素は "<要素名 />" という形式で記述できます。例えば、リスト 5.60, リスト 5.61 の要素は、読み込まれると同じデータとして処理されます。

リスト 5.60　子要素を持たない要素の記述例

```
<Item name="sample">

</Item>
```

リスト 5.61　子要素を持たない要素の記述例

```
<Item name="sample" />
```

5.6.2 タブ、スペース、改行について

XML では、タブ、スペース、改行は無視されますので、XML を読みやすくするために自由に追加できます。ただし、属性の値の中のスペースなどは無視されません。

リスト 5.62, リスト 5.63, リスト 5.64 の要素は、読み込まれるとすべて同じデータとして処理されます。

リスト 5.62　要素の記述例

```
<Item name="sample">
  <SubItem>
  </SubItem>
</Item>
```

リスト 5.63　要素の記述例

```
<Item
  name="sample"
>
  <SubItem></SubItem>
</Item>
```

リスト 5.64　要素の記述例

```
<Item name="sample"><SubItem></SubItem></Item>
```

5.6.3 コメントの書き方

XML では、"<!–" と "–>" で囲まれた間がコメントになります。リスト 5.65 にコメントの記述例を示します。

リスト 5.65　コメントの記述例

```
<!-- この部分はコメントになります。-->
<Item name="sample">
  <SubItem>
  </SubItem>
</Item>
```

第 6 章

iRIClib について

6.1 iRIClib とは

iRIClib は、ソルバー、格子生成プログラムを iRIC と連携させるためのライブラリです。

iRIC は、ソルバー、格子生成プログラムとの情報の入出力に使う計算データファイル、格子生成データファイルに CGNS ファイルを利用しています。 CGNS ファイルの入出力関数群は cgnslib というライブラリとしてオープンソースで公開されています (*CGNS* ファイル、 *CGNS* ライブラリに関する情報 (ページ 258) 参照)。しかし、cgnslib を直接利用して必要な入出力を記述すると、煩雑な処理を記述する必要があります。

そこで、iRIC プロジェクトでは iRIC に対応するソルバーでよく利用する入出力処理を簡便に記述するためのラッパー関数を提供するライブラリとして、iRIClib を用意しています。単一の構造格子を用いて計算を行うソルバーと、格子生成プログラムの入出力処理は、iRIClib で用意された関数を利用することで簡単に記述できます。

なお、複数の格子や非構造格子を使うソルバーなどで必要な関数は iRIClib では提供されません。そのようなソルバーでは、cgnslib で用意された関数を直接利用する必要があります。

この文書では、iRIClib を構成する関数群と利用例及びコンパイル方法について説明します。

6.2 利用可能な言語

iRIClib は、以下の言語から利用することができます。

- FORTRAN
- C/C++
- Python

マニュアルでは、FORTRAN から利用する場合の例を主に記載しています。

ここでは、FORTRAN, C/C++, Python から iRIClib を利用する方法の概要について説明します。

なお、関数を呼び出す際の引数などは、言語によって異なります。言語ごとの関数の呼び出し方法については、**リファレンス** (ページ 151) の各関数の解説を参照してください。

6.2.1 FORTRAN

リスト 6.1 に示すように、ヘッダファイルを読み込んだ上で、iRIClib の関数を呼び出します。

リスト 6.1　FORTRAN から iRIClib を利用するための記述例

```
include 'iriclib_f.h'

call cg_iric_init_f(fid, ier)
```

6.2.2 C/C++

リスト 6.2 に示すように、ヘッダファイルを読み込んだ上で、iRIClib の関数を呼び出します。

リスト 6.2　C/C++ から iRIClib を利用するための記述例

```
#include "iriclib.h"

// (中略)
ier = cg_iric_init(fid);
```

6.2.3 Python

リスト 6.3 に示すように、iric モジュールから関数を読み込んだ上で、iRIClib の関数を呼び出します。

リスト 6.3　Python から iRIClib を利用するための記述例

```
from iric import *

cg_iric_init(fid)
```

6.3 この章の読み方

概要 (ページ 115) で、iRIC がソルバー、格子生成プログラムについて想定している入出力処理と、その処理のために用意している関数についてを説明します。

まずは、**概要** (ページ 115) を読んで iRIClib の概要についてご理解ください。概要を理解したら、関数の引数のリストなどの詳細な情報は **リファレンス** (ページ 151) を参照してください。

6.4 概要

この節では、iRIClib の概要について説明します。

6.4.1 プログラムの処理と iRIClib の関数

ソルバー、格子生成プログラムで必要な入出力処理を 表 6.1 、表 6.2 に示します。

表 6.1　ソルバーの入出力処理

処理内容
CGNS ファイルを開く
内部変数の初期化
オプションの設定
計算条件の読み込み
計算格子の読み込み
境界条件の読み込み
地形データの読み込み (必要な場合のみ)
計算格子の出力 (格子の生成、再分割を行う場合のみ)
時刻 (もしくはループ回数) の出力
計算格子の出力 (移動格子の場合のみ)
計算結果の出力
CGNS ファイルを閉じる

表 6.2　格子生成プログラムの入出力処理

処理内容
CGNS 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
エラーコードの出力
格子の出力
CGNS ファイルを閉じる

6.4.2 CGNS ファイルを開く

CGNS ファイルを開き、読み込み、書き込みができる状態にします。この関数は cgnslib で定義された関数です。

表 6.3　利用する関数

関数	備考
cg_open_f	CGNS ファイルを開く。

6.4.3 内部変数の初期化

開いた CGNS ファイルを、 iRIClib から利用するための準備をします。CGNS ファイルを開いた後、いずれかを必ず実行します。

書き込みを行う場合には CG_MODE_MODIFY モードで CGNS ファイルを開き、cg_iric_init_f により初期化します。

読み込み専用の場合には、CG_MODE_READ モードで CGNS ファイルを開き、cg_iric_initread_f により初期化します。

表 6.4　利用する関数

関数	備考
cg_iric_init_f	指定したファイルを読み込み・書き込み用に iRIClib から利用するため、内部変数を初期化し、ファイルを初期化する
cg_iric_initread_f	指定したファイルを読み込み専用で iRIClib から利用するため、内部変数を初期化する

6.4.4 オプションの設定

ソルバーのオプションの設定を行います。現在指定できるオプションを、以下に示します。

- IRIC_OPTION_CANCEL: iric_check_cancel_f を利用してキャンセルを検知します。

表 6.5　利用する関数

関数	備考
iric_initoption_f	ソルバーのオプションを設定する

6.4.5 計算条件 (もしくは格子生成条件) の読み込み

CGNS ファイルから、計算条件 (もしくは格子生成条件) を読み込みます。

表 6.6　利用する関数

関数	備考
cg_iric_read_integer_f	整数の条件を読み込む
cg_iric_read_real_f	倍精度実数の条件を読み込む
cg_iric_read_realsingle_f	単精度実数の条件を読み込む
cg_iric_read_string_f	文字列の条件を読み込む
cg_iric_read_functionalsize_f	関数型の条件のサイズを調べる
cg_iric_read_functional_f	倍精度実数の関数型の条件を読み込む
cg_iric_read_functional_realsingle_f	単精度実数の関数型の条件を読み込む
cg_iric_read_functionalwithname_f	値を複数持つ倍精度実数の関数型の条件を読み込む

関数型以外の条件については、一つの関数で一つの条件を読み込むことができます。整数の計算条件を読み込む処理の例を リスト 6.4 に示します。

リスト 6.4　整数型の計算条件を読み込む処理の記述例

```
program Sample1
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, i_flow

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', 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 ***"

  call cg_iric_read_integer_f('i_flow', i_flow, ier)
  print *, i_flow;

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample1
```

一方、関数型の計算条件では、cg_iric_read_functionalsize_f, cg_iric_read_functional_f の二つの関数を利用する必要があります。関数型の計算条件を読み込む処理の例をリスト 6.5 に示します。

リスト 6.5　関数型の計算条件を読み込む処理の記述例

```
program Sample2
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, discharge_size, i
  double precision, dimension(:), allocatable:: discharge_time, discharge_value  ! discharge の時刻と値を保持する配列

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', 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 ***"

  ! まず、関数型の入力条件のサイズを調べる
  call cg_iric_read_functionalsize_f('discharge', discharge_size, ier)
  ! メモリを確保
  allocate(discharge_time(discharge_size), 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)

  ! （出力）
  if (ier ==0) then
    print *, 'discharge: discharge_size=', discharge_size
    do i = 1, min(discharge_size, 5)
      print *, ' i,time,value:', i, discharge_time(i), discharge_value(i)
    end do
  end if

  ! allocate で確保したメモリを開放
  deallocate(discharge_time, discharge_value)

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample2
```

計算条件 (もしくは 格子生成条件) の種類別の読み込み処理の記述例については、**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 70) を参照してください。

6.4.6 計算格子の読み込み

CGNS ファイルから、計算格子を読み込みます。iRIClib では、構造格子の読み込みの関数のみ提供します。

表 6.7　利用する関数

関数	備考
cg_iric_gotogridcoord2d_f	2 次元構造格子を読み込む準備をする
cg_iric_getgridcoord2d_f	2 次元構造格子を読み込む
cg_iric_gotogridcoord3d_f	3 次元構造格子を読み込む準備をする
cg_iric_getgridcoord3d_f	3 次元構造格子を読み込む
cg_iric_read_grid_integer_node_f	格子点で定義された整数の属性を読み込む
cg_iric_read_grid_real_node_f	格子点で定義された倍精度実数の属性を読み込む
cg_iric_read_grid_integer_cell_f	セルで定義された整数の属性を読み込む
cg_iric_read_grid_real_cell_f	セルで定義された倍精度実数の属性を読み込む
cg_iric_read_complex_count_f	複合型の属性のグループの数を読み込む
cg_iric_read_complex_integer_f	複合型の属性の整数の条件を読み込む
cg_iric_read_complex_real_f	複合型の属性の倍精度実数の条件を読み込む
cg_iric_read_complex_realsingle_f	複合型の属性の単精度実数の条件を読み込む
cg_iric_read_complex_string_f	複合型の属性の文字列の条件を読み込む
cg_iric_read_complex_functionalsize_f	複合型の属性の関数型の条件のサイズを調べる
cg_iric_read_complex_functional_f	複合型の属性の倍精度実数の関数型の条件を読み込む
cg_iric_read_complex_functionalwithname_f	複合型の属性の単精度実数の関数型の条件を読み込む
cg_iric_read_complex_functional_realsingle_f	複合型の属性の値を複数持つ倍精度実数の関数型の条件を読み込む
cg_iric_read_grid_complex_node_f	格子点で定義された複合型の属性を読み込む
cg_iric_read_grid_complex_cell_f	セルで定義された複合型の属性を読み込む
cg_iric_read_grid_functionaltimesize_f	次元「時刻」(Time) を持つ格子属性の、時刻の数を調べる
cg_iric_read_grid_functionaltime_f	次元「時刻」(Time) の値を読み込む
cg_iric_read_grid_functionaldimensionsize_f	次元の数を調べる
cg_iric_read_grid_functionaldimension_integer_f	整数の次元の値を読み込む
cg_iric_read_grid_functionaldimension_real_f	倍精度実数の次元の値を読み込む
cg_iric_read_grid_functional_integer_node_f	次元「時刻」を持つ、格子点で定義された整数の属性を読み込む
cg_iric_read_grid_functional_real_node_f	次元「時刻」を持つ、格子点で定義された倍精度実数の属性を読み込む
cg_iric_read_grid_functional_integer_cell_f	次元「時刻」を持つ、セルで定義された整数の属性を読み込む
cg_iric_read_grid_functional_real_cell_f	次元「時刻」を持つ、セルで定義された倍精度実数の属性を読み込む

cg_iric_read_grid_integer_node_f など属性読み込み用の関数は、2 次元構造格子、3 次元構造格子で共通で利用することができます。2 次元構造格子を読み込む処理の記述例を リスト 6.6 に示します。

リスト 6.6　2 次元格子を読み込む処理の記述例

```
program Sample3
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, discharge_size, i, j
  integer:: isize, jsize
  double precision, dimension(:,:), allocatable:: grid_x, grid_y
  double precision, dimension(:,:), allocatable:: elevation
  integer, dimension(:,:), allocatable:: obstacle
  integer:: rain_timeid
  integer:: rain_timesize
  double precision, dimension(:), allocatable:: rain_time
  double precision, dimension(:,:), allocatable:: rain

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', 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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)

  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)

  if (ier /=0) STOP "*** No grid data ***"
  ! （出力）
  print *, 'grid x,y: isize, jsize=', isize, jsize
  do i = 1, min(isize,5)
    do j = 1, min(jsize,5)
      print *, ' (',i,',',j,')=(',grid_x(i,j),',',grid_y(i,j),')'
    end do
  end do

  ! 格子点で定義された属性 elevation のメモリを確保
  allocate(elevation(isize, jsize))
  ! 属性を読み込む
  call cg_iric_read_grid_real_node_f('Elevation', elevation, ier)
  print *, 'Elevation: isize, jsize=', isize, jsize
  do i = 1, min(isize,5)
    do j = 1, min(jsize,5)
```

(次のページに続く)

(前のページからの続き)

```
      print *, ' (',i,',',j,')=(',elevation(i,j),')'
    end do
  end do

  ! セルで定義された属性 obstacle のメモリを確保。セルの属性なのでサイズは (isize-1) * (jsize-1)
  allocate(obstacle(isize-1, jsize-1))
  ! 属性を読み込む
  call cg_iric_read_grid_integer_cell_f('Obstacle', obstacle, ier)
  print *, 'Obstacle: isize -1, jsize-1=', isize-1, jsize-1
  do i = 1, min(isize-1,5)
    do j = 1, min(jsize-1,5)
      print *, ' (',i,',',j,')=(',obstacle(i,j),')'
    end do
  end do
  ! Rain の時刻の数を読み込む
  call cg_iric_read_grid_functionaltimesize_f('Rain', rain_timesize);
  ! Rain の時刻を読み込むメモリを確保。
  allocate(rain_time(rain_timesize))

  ! セルで定義された属性 rain のメモリを確保。セルの属性なのでサイズは (isize-1) * (jsize-1)
  allocate(rain(isize-1, jsize-1))
  ! Time = 1 での属性を読み込む
  rain_timeid = 1
  call cg_iric_read_grid_functional_real_cell_f('Rain', rain_timeid, rain, ier)
  print *, 'Rain: isize -1, jsize-1=', isize-1, jsize-1
  do i = 1, min(isize-1,5)
    do j = 1, min(jsize-1,5)
      print *, ' (',i,',',j,')=(',rain(i,j),')'
    end do
  end do

  ! allocate で確保したメモリを開放
  deallocate(grid_x, grid_y, elevation, obstacle, rain_time, rain)

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample3
```

3 次元の格子の場合も同様の処理になります。

6.4.7 境界条件の読み込み

CGNS ファイルから、境界条件を読み込みます。

表 6.8　利用する関数

関数	備考
cg_iric_read_bc_count_f	境界条件の数を取得する
cg_iric_read_bc_indicessize_f	境界条件の設定された要素 (格子点もしくはセル) の数を取得する
cg_iric_read_bc_indices_f	境界条件の設定された要素 (格子点もしくはセル) のインデックスの配列を取得する
cg_iric_read_bc_integer_f	整数型境界条件の値を取得する
cg_iric_read_bc_real_f	実数 (倍精度) 境界条件の値を取得する
cg_iric_read_bc_realsingle_f	実数 (単精度) 境界条件の値を取得する
cg_iric_read_bc_string_f	文字列型境界条件の値を取得する
cg_iric_read_bc_functionalsize_f	関数型境界条件のサイズを取得する
cg_iric_read_bc_functional_f	倍精度実数の関数型境界条件の値を取得する
cg_iric_read_bc_functionalwithname_f	単精度実数の関数型境界条件の値を取得する

同じ種類の境界条件を、1 つの格子に複数定義することができます。例えば、流入口を一つの格子に複数定義し、流入量をそれぞれ独立に与えることができます。

境界条件を読み込む処理の記述例を リスト 6.7 に示します。この例では、流入口 (Inflow) の数を cg_iric_read_bc_count_f で調べ、必要なメモリを確保してから境界条件の設定情報を読み込んでいます。なお、GUI で指定した境界条件の名前は cg_iric_read_bc_string_f で読み込めます。

リスト 6.7　境界条件を読み込む処理の記述例

```
1  program Sample8
2    implicit none
3    include 'cgnslib_f.h'
4
5    integer:: fin, ier, isize, jsize, ksize, i, j, k, aret
6    integer:: condid, indexid
7    integer:: condcount, indexlenmax, funcsizemax
8    integer:: tmplen
9    integer, dimension(:), allocatable:: condindexlen
10   integer, dimension(:,:,:), allocatable:: condindices
11   integer, dimension(:), allocatable:: intparam
12   double precision, dimension(:), allocatable:: realparam
13   character(len=200), dimension(:), allocatable:: stringparam
14   character(len=200):: tmpstr
15   integer, dimension(:), allocatable:: func_size
16   double precision, dimension(:,:), allocatable:: func_param;
17   double precision, dimension(:,:), allocatable:: func_value;
18
19   ! CGNS ファイルのオープン
20   call cg_open_f('bctest.cgn', CG_MODE_MODIFY, fin, ier)
21   if (ier /=0) STOP "*** Open error of CGNS file ***"
```

(次のページに続く)

(前のページからの続き)

```
22
23    ! 内部変数の初期化
24    call cg_iric_init_f(fin, ier)
25    if (ier /=0) STOP "*** Initialize error of CGNS file ***"
26
27    ! 流入口の数を取得する
28    call cg_iric_read_bc_count_f('inflow', condcount)
29    ! 流入口の数に従って、パラメータの保存用のメモリを確保する。
30    allocate(condindexlen(condcount), intparam(condcount), realparam(condcount))
31    allocate(stringparam(condcount), func_size(condcount))
32    print *, 'condcount ', condcount
33
34    ! 境界条件が設定された格子点の数と、関数型の境界条件の最大サイズを調べる
35    indexlenmax = 0
36    funcsizemax = 0
37    do condid = 1, condcount
38      call cg_iric_read_bc_indicessize_f('inflow', condid, condindexlen(condid), ier)
39      if (indexlenmax < condindexlen(condid)) then
40        indexlenmax = condindexlen(condid)
41      end if
42      call cg_iric_read_bc_functionalsize_f('inflow', condid, 'funcparam', func_
    ↪size(condid), ier);
43      if (funcsizemax < func_size(condid)) then
44        funcsizemax = func_size(condid)
45      end if
46    end do
47
48    ! 格子点のインデックス格納用の配列と、関数型境界条件の格納用変数のメモリを確保
49    allocate(condindices(condcount, 2, indexlenmax))
50    allocate(func_param(condcount, funcsizemax), func_value(condcount, funcsizemax))
51    ! インデックスと、境界条件 を読み込み
52    do condid = 1, condcount
53      call cg_iric_read_bc_indices_f('inflow', condid, condindices(condid:condid,:,:),␣
    ↪ier)
54      call cg_iric_read_bc_integer_f('inflow', condid, 'intparam',␣
    ↪intparam(condid:condid), ier)
55      call cg_iric_read_bc_real_f('inflow', condid, 'realparam',␣
    ↪realparam(condid:condid), ier)
56      call cg_iric_read_bc_string_f('inflow', condid, 'stringparam', tmpstr, ier)
57      stringparam(condid) = tmpstr
58      call cg_iric_read_bc_functional_f('inflow', condid, 'funcparam', func_
    ↪param(condid:condid,:), func_value(condid:condid,:), ier)
59    end do
60
61    ! 読み込まれた境界条件を表示
62    do condid = 1, condcount
63      do indexid = 1, condindexlen(condid)
64        print *, 'condindices ', condindices(condid:condid,:,indexid:indexid)
```

(次のページに続く)

(前のページからの続き)

```
    end do
    print *, 'intparam ', intparam(condid:condid)
    print *, 'realparam ', realparam(condid:condid)
    print *, 'stringparam ', stringparam(condid)
    print *, 'funcparam X ', func_param(condid:condid, 1:func_size(condid))
    print *, 'funcparam Y ', func_value(condid:condid, 1:func_size(condid))
  end do

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample8
```

6.4.8 地形データの読み込み

プロジェクトでインポートして格子生成に利用した地形データを読み込みます。ソルバーで、河川測量データやポリゴンを直接読み込んで解析に使用したい場合に行います。地形データを読み込む場合の手順は、以下の通りになります。

1. CGNS ファイルから、プロジェクトで使用した地形データのファイル名などを読み込みます。
2. 地形データファイルを開き、地形データを読み込みます。

表 6.9　利用する関数

関数	備考
cg_iric_read_geo_count_f	地形データの数を返す
cg_iric_read_geo_filename_f	地形データのファイル名と種類を返す
iric_geo_polygon_open_f	ポリゴンファイルを開く
iric_geo_polygon_read_integervalue_f	ポリゴンの値を整数で返す
iric_geo_polygon_read_realvalue_f	ポリゴンの値を実数で返す
iric_geo_polygon_read_pointcount_f	ポリゴンの頂点の数を返す
iric_geo_polygon_read_points_f	ポリゴンの頂点の座標を返す
iric_geo_polygon_read_holecount_f	ポリゴンに開いた穴の数を返す
iric_geo_polygon_read_holepointcount_f	ポリゴンの穴の頂点の数を返す
iric_geo_polygon_read_holepoints_f	ポリゴンの穴の頂点の座標を返す
iric_geo_polygon_close_f	ポリゴンファイルを閉じる
iric_geo_riversurvey_open_f	河川測量データを開く
iric_geo_riversurvey_read_count_f	河川横断線の数を返す
iric_geo_riversurvey_read_position_f	横断線の中心点の座標を返す
iric_geo_riversurvey_read_direction_f	横断線の向きを返す
iric_geo_riversurvey_read_name_f	横断線の名前を文字列として返す
iric_geo_riversurvey_read_realname_f	横断線の名前を実数値として返す
iric_geo_riversurvey_read_leftshift_f	横断線の標高データのシフト量を返す
iric_geo_riversurvey_read_altitudecount_f	横断線の標高データの数を返す
iric_geo_riversurvey_read_altitudes_f	横断線の標高データを返す
iric_geo_riversurvey_read_fixedpointl_f	横断線の左岸延長線のデータを返す
iric_geo_riversurvey_read_fixedpointr_f	横断線の右岸延長線のデータを返す
iric_geo_riversurvey_read_watersurfaceelevation_f	横断線での水面標高のデータを返す
iric_geo_riversurvey_close_f	河川測量データを閉じる

地形データのうち、ポリゴンを読み込む処理の記述例をリスト 6.8 に、河川測量データを読み込む処理の記述例をリスト 6.9 にそれぞれ示します。

リスト 6.8　ポリゴンを読み込む処理の記述例

```
1  program TestPolygon
2    implicit none
3    include 'cgnslib_f.h'
4    include 'iriclib_f.h'
5    integer:: fin, ier
6    integer:: icount, istatus
7
8    integer:: geoid
9    integer:: elevation_geo_count
```

(次のページに続く)

(前のページからの続き)

```
  character(len=1000):: filename
  integer:: geotype
  integer:: polygonid
  double precision:: polygon_value
  integer:: region_pointcount
  double precision, dimension(:), allocatable:: region_pointx
  double precision, dimension(:), allocatable:: region_pointy
  integer:: hole_id
  integer:: hole_count
  integer:: hole_pointcount
  double precision, dimension(:), allocatable:: hole_pointx
  double precision, dimension(:), allocatable:: hole_pointy


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

  ! iRIClib の初期化
  call cg_iric_init_f(fin, ier)

  ! 地形データの数を取得
  call cg_iric_read_geo_count_f("Elevation", elevation_geo_count, ier)

  do geoid = 1, elevation_geo_count
    call cg_iric_read_geo_filename_f('Elevation', geoid, &
      filename, geotype, ier)
    if (geotype .eq. iRIC_GEO_POLYGON) then
      call iric_geo_polygon_open_f(filename, polygonid, ier)
      call iric_geo_polygon_read_realvalue_f(polygonid, polygon_value, ier)
      print *, polygon_value
      call iric_geo_polygon_read_pointcount_f(polygonid, region_pointcount, ier)
      allocate(region_pointx(region_pointcount))
      allocate(region_pointy(region_pointcount))
      call iric_geo_polygon_read_points_f(polygonid, region_pointx, region_pointy, ier)
      print *, 'region_x: ', region_pointx
      print *, 'region_y: ', region_pointy
      deallocate(region_pointx)
      deallocate(region_pointy)
      call iric_geo_polygon_read_holecount_f(polygonid, hole_count, ier)
      print *, 'hole count: ', hole_count
      do hole_id = 1, hole_count
        print *, 'hole ', hole_id
        call iric_geo_polygon_read_holepointcount_f(polygonid, hole_id, hole_
↪pointcount, ier)
        print *, 'hole pointcount: ', hole_pointcount
        allocate(hole_pointx(hole_pointcount))
        allocate(hole_pointy(hole_pointcount))
```

(次のページに続く)

(前のページからの続き)

```
        call iric_geo_polygon_read_holepoints_f(polygonid, hole_id, hole_pointx, hole_
↪pointy, ier)
        print *, 'hole_x: ', hole_pointx
        print *, 'hole_y: ', hole_pointy
        deallocate(hole_pointx)
        deallocate(hole_pointy)
      end do
      call iric_geo_polygon_close_f(polygonid, ier)
    end if
  end do

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

リスト 6.9　河川測量データを読み込む処理の記述例

```
program TestRiverSurvey
  implicit none
  include 'cgnslib_f.h'
  include 'iriclib_f.h'
  integer:: fin, ier
  integer:: icount, istatus

  integer:: geoid
  integer:: elevation_geo_count
  character(len=1000):: filename
  integer:: geotype
  integer:: rsid
  integer:: xsec_count
  integer:: xsec_id
  character(len=20):: xsec_name
  double precision:: xsec_x
  double precision:: xsec_y
  integer:: xsec_set
  integer:: xsec_index
  double precision:: xsec_leftshift
  integer:: xsec_altid
  integer:: xsec_altcount
  double precision, dimension(:), allocatable:: xsec_altpos
  double precision, dimension(:), allocatable:: xsec_altheight
  integer, dimension(:), allocatable:: xsec_altactive
  double precision:: xsec_wse

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

(次のページに続く)

(前のページからの続き)

```
31
32    ! iRIClib の初期化
33    call cg_iric_init_f(fin, ier)
34
35    ! 地形データの数を取得
36    call cg_iric_read_geo_count_f("Elevation", elevation_geo_count, ier)
37
38    do geoid = 1, elevation_geo_count
39      call cg_iric_read_geo_filename_f('Elevation', geoid, &
40        filename, geotype, ier)
41      if (geotype .eq. iRIC_GEO_RIVERSURVEY) then
42        call iric_geo_riversurvey_open_f(filename, rsid, ier)
43        call iric_geo_riversurvey_read_count_f(rsid, xsec_count, ier)
44        do xsec_id = 1, xsec_count
45          call iric_geo_riversurvey_read_name_f(rsid, xsec_id, xsec_name, ier)
46          print *, 'xsec ', xsec_name
47          call iric_geo_riversurvey_read_position_f(rsid, xsec_id, xsec_x, xsec_y, ier)
48          print *, 'position: ', xsec_x, xsec_y
49          call iric_geo_riversurvey_read_direction_f(rsid, xsec_id, xsec_x, xsec_y, ier)
50          print *, 'direction: ', xsec_x, xsec_y
51          call iric_geo_riversurvey_read_leftshift_f(rsid, xsec_id, xsec_leftshift, ier)
52          print *, 'leftshift: ', xsec_leftshift
53          call iric_geo_riversurvey_read_altitudecount_f(rsid, xsec_id, xsec_altcount,␣
    ↪ier)
54          print *, 'altitude count: ', xsec_altcount
55          allocate(xsec_altpos(xsec_altcount))
56          allocate(xsec_altheight(xsec_altcount))
57          allocate(xsec_altactive(xsec_altcount))
58          call iric_geo_riversurvey_read_altitudes_f( &
59            rsid, xsec_id, xsec_altpos, xsec_altheight, xsec_altactive, ier)
60          do xsec_altid = 1, xsec_altcount
61            print *, 'Altitude ', xsec_altid, ': ', &
62              xsec_altpos(xsec_altid:xsec_altid), ', ', &
63              xsec_altheight(xsec_altid:xsec_altid), ', ', &
64              xsec_altactive(xsec_altid:xsec_altid)
65          end do
66          deallocate(xsec_altpos, xsec_altheight, xsec_altactive)
67          call iric_geo_riversurvey_read_fixedpointl_f( &
68            rsid, xsec_id, xsec_set, xsec_x, xsec_y, xsec_index, ier)
69          print *, 'FixedPointL: ', xsec_set, xsec_x, xsec_y, xsec_index
70          call iric_geo_riversurvey_read_fixedpointr_f( &
71            rsid, xsec_id, xsec_set, xsec_x, xsec_y, xsec_index, ier)
72          print *, 'FixedPointR: ', xsec_set, xsec_x, xsec_y, xsec_index
73          call iric_geo_riversurvey_read_watersurfaceelevation_f( &
74            rsid, xsec_id, xsec_set, xsec_wse, ier)
75          print *, 'WaterSurfaceElevation: ', xsec_set, xsec_wse
76        end do
77        call iric_geo_riversurvey_close_f(rsid, ier)
```

(次のページに続く)

(前のページからの続き)

```
    end if
  end do

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

6.4.9 計算格子の出力

CGNS ファイルに、計算格子を出力します。

ソルバーでは、ソルバーで計算に用いる格子を生成する場合や、2 次元格子から 3 次元格子を生成する場合に行います。

格子生成プログラムでは必ず行います。

ここで示す関数は、ソルバーでは計算開始時の格子を出力するために使用します。計算中に格子形状が変化する場合の格子の出力には、**計算格子の出力 (計算開始後の格子)** (ページ 132) に示す関数を使用して下さい。

表 6.10　利用する関数

関数	備考
cg_iric_writegridcoord1d_f	1 次元構造格子を出力する
cg_iric_writegridcoord2d_f	2 次元構造格子を出力する
cg_iric_writegridcoord3d_f	3 次元構造格子を出力する
cg_iric_write_grid_real_node_f	格子点で定義された整数の属性を出力する
cg_iric_write_grid_integer_node_f	格子点で定義された倍精度実数の属性を出力する
cg_iric_write_grid_real_cell_f	セルで定義された整数の属性を出力する
cg_iric_write_grid_integer_cell_f	セルで定義された倍精度実数の属性を出力する

2 次元格子を読み込み、それを分割して生成した 3 次元格子を出力する処理の記述例を リスト 6.10 に示します。

リスト 6.10　3 次元格子を出力する処理の記述例

```
program Sample7
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, isize, jsize, ksize, i, j, k, aret
  double precision:: time
  double precision:: convergence
  double precision, dimension(:,:), allocatable::grid_x, grid_y, elevation
  double precision, dimension(:,:,:), allocatable::grid3d_x, grid3d_y, grid3d_z
```

(次のページに続く)

(前のページからの続き)

```
  double precision, dimension(:,:,:), allocatable:: velocity, density

  ! CGNS ファイルのオープン
  call cg_open_f('test3d.cgn', 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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize), elevation(isize,jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)
  call cg_iric_read_grid_real_node_f('Elevation', elevation, ier)

  ! 読み込んだ 2 次元格子を元に、3 次元格子を生成。
  ! 3 次元格子は Z 方向に、深さ 5 で、5 分割する

  ksize = 6
  allocate(grid3d_x(isize,jsize,ksize), grid3d_y(isize,jsize,ksize), grid3d_z(isize,
↪jsize,ksize))
  allocate(velocity(isize,jsize,ksize), STAT = aret)
  print *, aret
  allocate(density(isize,jsize,ksize), STAT = aret)
  print *, aret
  do i = 1, isize
    do j = 1, jsize
      do k = 1, ksize
        grid3d_x(i,j,k) = grid_x(i,j)
        grid3d_y(i,j,k) = grid_y(i,j)
        grid3d_z(i,j,k) = elevation(i,j) + (k - 1)
        velocity(i,j,k) = 0
        density(i,j,k) = 0
      end do
    end do
  end do
  ! 生成した 3 次元格子を出力
  call cg_iric_writegridcoord3d_f(isize, jsize, ksize, grid3d_x, grid3d_y, grid3d_z,␣
↪ier)

  ! 初期状態の情報を出力
  time = 0
  convergence = 0.1
  call cg_iric_write_sol_time_f(time, ier)
  ! 格子を出力
```

(次のページに続く)

(前のページからの続き)

```
  call cg_iric_write_sol_gridcoord3d_f(grid3d_x, grid3d_y, grid3d_z, ier)
  ! 計算結果を出力
  call cg_iric_write_sol_real_f('Velocity', velocity, ier)
  call cg_iric_write_sol_real_f('Density', density, ier)
  call cg_iric_write_sol_baseiterative_real_f ('Convergence', convergence, ier)


  do
    time = time + 10.0
    ! (ここで計算を実行。格子の形状も変化)
    call cg_iric_write_sol_time_f(time, ier)
    ! 格子を出力
    call cg_iric_write_sol_gridcoord3d_f(grid3d_x, grid3d_y, grid3d_z, ier)
    ! 計算結果を出力
    call cg_iric_write_sol_real_f('Velocity', velocity, ier)
    call cg_iric_write_sol_real_f('Density', density, ier)
    call cg_iric_write_sol_baseiterative_real_f ('Convergence', convergence, ier)

    If (time > 100) exit
  end do

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample7
```

6.4.10 時刻 (もしくはループ回数) の出力

CGNS ファイルに、時刻もしくはループ回数を出力します。

その時刻での計算格子の出力や計算結果の出力を行うより前に、必ず実行してください。

また、時刻とループ回数を両方出力することはできません。必ずいずれかのみ出力してください。

表 6.11　利用する関数

関数	備考
cg_iric_write_sol_time_f	時刻を出力する
cg_iric_write_sol_iteration_f	ループ回数を出力する

時刻を出力する処理の例を リスト 6.11 に示します。

リスト 6.11　時刻を出力する処理の記述例

```
program Sample4
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, i
  double precision:: time

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', 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 ***"

  ! 初期状態の情報を出力
  time = 0

  call cg_iric_write_sol_time_f(time, ier)
  ! (ここで、初期の計算格子や計算結果を出力)

  do
    time = time + 10.0
    ! (ここで計算を実行)
    call cg_iric_write_sol_time_f(time, ier)
    ! (ここで、計算格子や計算結果を出力)
    If (time > 1000) exit
  end do

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample4
```

6.4.11 計算格子の出力 (計算開始後の格子)

CGNS ファイルに、計算開始後の計算格子を出力します。計算中に格子形状が変化するソルバーでのみ行います。

特定の時間での計算格子を出力する前に、必ず **時刻 (もしくはループ回数) の出力** (ページ 131) で示した時刻 (もしくはループ回数) の出力を行ってください。

以下に示す場合の格子の出力については、**計算格子の出力** (ページ 129) で示した関数を利用してください。

- ソルバーで新たに格子を生成した

- ソルバーで格子を再分割するなどして、次元や格子点数が異なる格子を生成した
- 格子生成プログラム内で格子を生成した

表 6.12　利用する関数

関数	備考
cg_iric_write_sol_gridcoord2d_f	2 次元構造格子を出力する
cg_iric_write_sol_gridcoord3d_f	3 次元構造格子を出力する

2 次元構造格子を出力する処理の例を リスト 6.12 に示します。

表 6 11 2 次元構造格子を出力する処理の記述例

リスト 6.12　2 次元構造格子を出力する処理の記述例

```
program Sample5
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, isize, jsize
  double precision:: time
  double precision, dimension(:,:), allocatable:: grid_x, grid_y

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', 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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0

  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)

  do
    time = time + 10.0
    ! (ここで計算を実行)
```

(次のページに続く)

(前のページからの続き)

```
    call cg_iric_write_sol_time_f(time, ier)
    call cg_iric_write_sol_gridcoord2d_f (grid_x, grid_y, ier)
    If (time > 1000) exit
  end do

  ! CGNS ファイルのクローズ
  call cg_close_f(fin, ier)
  stop
end program Sample5
```

6.4.12 計算結果の出力

CGNS ファイルに、計算結果を出力します。

iRIClib で出力できる計算結果は、大きく以下に分類されます。

- 1 つのタイムステップで 1 つ値を持つ計算結果
- 格子点ごとに値を持つ計算結果
- 格子セルごとに値を持つ計算結果
- 格子辺ごとに値を持つ計算結果
- 粒子の座標ごとに値を持つ計算結果
- 粒子の座標ごとに値を持つ計算結果 (複数グループ可)
- ポリゴンもしくは折れ線ごとに値を持つ計算結果

どの種類の計算結果を出力する場合も、 表 6.13 と 表 6.14 に示す関数は必ず使用します。

計算結果の分類ごとの固有の関数は、 **1 つのタイムステップで 1 つ値を持つ計算結果** (ページ 135) 〜 **ポリゴンもしくは折れ線ごとに値を持つ計算結果** (ページ 147) をそれぞれ参照してください。

表 6.13　計算結果の出力の開始前、終了後に利用する関数

関数	備考
iric_check_cancel_f	ユーザがソルバーの実行をキャンセルしたか確認する
iric_check_lock_f	CGNS ファイルが GUI によってロックされているか確認する
iric_write_sol_start_f	計算結果の出力開始を GUI に通知する
iric_write_sol_end_f	計算結果の出力終了を GUI に通知する
cg_iric_flush_f	計算結果の出力をファイルに書き込む

表 6.14　時刻・ループ回数の出力に利用する関数

関数	備考
cg_iric_write_sol_time_f	時刻を出力する
cg_iric_write_sol_iteration_f	ループ回数を出力する

注釈: ベクトル量とスカラー量

iRIClib では、ベクトル量の計算結果とスカラー量の計算結果は、同じ関数を使って出力を行います。

ベクトル量の計算結果を出力する場合は、"VelocityX", "VelocityY" などの名前で各成分を出力してください。

スカラー量の計算結果の出力で、名前の最後に "X", "Y", "Z" を使った場合、GUI で正しく読み込まれず可視化できませんのでご注意下さい。小文字の "x", "y", "z" は問題なく使用できます。

注釈: 計算結果で使用する特別な名前

計算結果について、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 **計算結果** (ページ 258) を参照してください。

注釈: 格子点・格子セル・格子エッジ

格子に関係する計算結果は、格子点で出力する方法、格子セルで出力する方法、格子エッジで出力する方法があります。

基本的には、ソルバにおいて変数がどこで定義されているかによって、どの方法で計算結果を出力するかを選択してください。

ただし、ベクトル量については格子点で出力してください。格子セル・格子エッジでベクトル量を出力しても、矢印、流線、パーティクルの描画はできません。

---

1 つのタイムステップで 1 つ値を持つ計算結果

1 つのタイムステップで 1 つ値を持つ計算結果を出力する場合、表 6.15 に示す関数を使用します。

出力するプログラムの例は、リスト 6.13 を参照してください。

表 6.15　1 つのタイムステップで 1 つ値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_baseiterative_integer_f	整数の計算結果を出力する
cg_iric_write_sol_baseiterative_real_f	倍精度実数の計算結果を出力する
cg_iric_write_sol_baseiterative_string_f	文字列の計算結果を出力する

リスト 6.13　サンプルプログラム (1 つのタイムステップで 1 つ値を持つ計算結果)

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision:: convergence
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f (grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0
  convergence = 0.1
  call cg_iric_write_sol_time_f(time, ier)
  call cg_iric_write_sol_baseiterative_real_f('Convergence', convergence, ier)
  do
    time = time + 10.0

```

(次のページに続く)

(前のページからの続き)

```
    ! (ここで計算を実行)

    call iric_check_cancel_f(canceled)
    if (canceled == 1) exit

    ! 計算結果を出力
    call iric_write_sol_start_f(condFile, ier)
    call cg_iric_write_sol_time_f(time, 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)

    if (time > 1000) exit
  end do

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

格子点ごとに値を持つ計算結果

格子点ごとに値を持つ計算結果を出力する場合、表 6.16 に示す関数を使用します。

出力するプログラムの例は、リスト 6.14 を参照してください。

表 6.16　格子点ごとに値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_integer_f	整数の格子点ごとに値を持つ計算結果を出力する
cg_iric_write_sol_real_f	倍精度実数の格子点ごとに値を持つ計算結果を出力する

リスト 6.14　サンプルプログラム (格子点ごとに値を持つ計算結果)

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  double precision, dimension(:,:), allocatable:: velocity_x, velocity_y, depth
  integer, dimension(:,:), allocatable:: wetflag
  character(len=20):: condFile
```

(次のページに続く)

(前のページからの続き)

```
13
14    condFile = 'test.cgn'
15
16    ! CGNS ファイルのオープン
17    call cg_open_f(condFile, CG_MODE_MODIFY, fin, ier)
18    if (ier /=0) STOP "*** Open error of CGNS file ***"
19
20    ! 内部変数の初期化
21    call cg_iric_init_f(fin, ier)
22    if (ier /=0) STOP "*** Initialize error of CGNS file ***"
23
24    ! 格子のサイズを調べる
25    call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
26    ! 格子を読み込むためのメモリを確保
27    allocate(grid_x(isize, jsize), grid_y(isize, jsize))
28    ! 計算結果を保持するメモリも確保
29    allocate(velocity_x(isize, jsize), velocity_y(isize, jsize), depth(isize, jsize),␣
  ↪wetflag(isize, jsize))
30    ! 格子を読み込む
31    call cg_iric_getgridcoord2d_f (grid_x, grid_y, ier)
32
33    ! 初期状態の情報を出力
34    time = 0
35    convergence = 0.1
36    call cg_iric_write_sol_time_f(time, ier)
37    call cg_iric_write_sol_real_f('VelocityX', velocity_x, ier)
38    call cg_iric_write_sol_real_f('VelocityY', velocity_y, ier)
39    call cg_iric_write_sol_real_f('Depth', depth, ier)
40    call cg_iric_write_sol_integer_f('Wet', wetflag, ier)
41    do
42      time = time + 10.0
43
44      ! (ここで計算を実行)
45
46      call iric_check_cancel_f(canceled)
47      if (canceled == 1) exit
48
49      ! 計算結果を出力
50      call iric_write_sol_start_f(condFile, ier)
51      call cg_iric_write_sol_time_f(time, ier)
52      call cg_iric_write_sol_real_f('VelocityX', velocity_x, ier)
53      call cg_iric_write_sol_real_f('VelocityY', velocity_y, ier)
54      call cg_iric_write_sol_real_f('Depth', depth, ier)
55      call cg_iric_write_sol_integer_f('Wet', wetflag, ier)
56      call cg_iric_flush_f(condFile, fin, ier)
57      call iric_write_sol_end_f(condFile, ier)
58
59      if (time > 1000) exit
```

(次のページに続く)

(前のページからの続き)

```
  end do

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

格子セルごとに値を持つ計算結果

格子セルごとに値を持つ計算結果を出力する場合、表 6.17 に示す関数を使用します。

出力するプログラムの例は、リスト 6.15 を参照してください。

表 6.17　格子セルごとに値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_cell_integer_f	整数の格子セルごとに値を持つ計算結果を出力する
cg_iric_write_sol_real_f	倍精度実数の格子セルごとに値を持つ計算結果を出力する

リスト 6.15　サンプルプログラム (格子セルごとに値を持つ計算結果)

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  double precision, dimension(:,:), allocatable:: depth
  integer, dimension(:,:), allocatable:: wetflag
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
```

(次のページに続く)

(前のページからの続き)

```
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  ! 計算結果を保持するメモリも確保
  allocate(depth(isize - 1, jsize - 1), wetflag(isize - 1, jsize - 1))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f (grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0
  convergence = 0.1
  call cg_iric_write_sol_time_f(time, ier)
  call cg_iric_write_sol_cell_real_f('Depth', depth, ier)
  call cg_iric_write_sol_cell_integer_f('Wet', wetflag, ier)
  do
    time = time + 10.0

    ! (ここで計算を実行)

    call iric_check_cancel_f(canceled)
    if (canceled == 1) exit

    ! 計算結果を出力
    call iric_write_sol_start_f(condFile, ier)
    call cg_iric_write_sol_time_f(time, ier)
    call cg_iric_write_sol_cell_real_f('Depth', depth, ier)
    call cg_iric_write_sol_cell_integer_f('Wet', wetflag, ier)
    call cg_iric_flush_f(condFile, fin, ier)
    call iric_write_sol_end_f(condFile, ier)

    if (time > 1000) exit
  end do

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

格子エッジごとに値を持つ計算結果

格子エッジごとに値を持つ計算結果を出力する場合、表 6.18 に示す関数を使用します。

出力するプログラムの例は、リスト 6.16 を参照してください。

表 6.18　格子エッジごとに値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_iface_integer_f	整数の I 方向の格子エッジごとに値を持つ計算結果を出力する
cg_iric_write_sol_iface_real_f	倍精度実数の I 方向の格子エッジごとに値を持つ計算結果を出力する
cg_iric_write_sol_jface_integer_f	整数の J 方向の格子エッジごとに値を持つ計算結果を出力する
cg_iric_write_sol_jface_real_f	倍精度実数の J 方向の格子エッジごとに値を持つ計算結果を出力する

リスト 6.16　サンプルプログラム (格子エッジごとに値を持つ計算結果)

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  double precision, dimension(:,:), allocatable:: fluxi, fluxj
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  ! 計算結果を保持するメモリも確保
  allocate(fluxi(isize, jsize - 1), fluxj(isize - 1, jsize))
  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f (grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0
  convergence = 0.1
  call cg_iric_write_sol_time_f(time, ier)
  call cg_iric_write_sol_iface_real_f('FluxI', fluxi, ier)
  call cg_iric_write_sol_jface_real_f('FluxJ', fluxj, ier)
  do
```

(次のページに続く)

(前のページからの続き)

```
    time = time + 10.0

    ! (ここで計算を実行)

    call iric_check_cancel_f(canceled)
    if (canceled == 1) exit

    ! 計算結果を出力
    call iric_write_sol_start_f(condFile, ier)
    call cg_iric_write_sol_time_f(time, ier)
    call cg_iric_write_sol_iface_real_f('FluxI', fluxi, ier)
    call cg_iric_write_sol_jface_real_f('FluxJ', fluxj, ier)
    call cg_iric_flush_f(condFile, fin, ier)
    call iric_write_sol_end_f(condFile, ier)

    if (time > 1000) exit
  end do

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

粒子の座標ごとに値を持つ計算結果

粒子の座標ごとに値を持つ計算結果を出力する場合、表 6.19 に示す関数を使用します。

出力するプログラムの例は、リスト 6.17 を参照してください。

注釈: ここで示す関数群は現在は非推奨です

粒子の座標ごとに値を持つ計算結果を出力する場合は、**粒子の座標ごとに値を持つ計算結果 (複数グループ可)** (ページ 144) に示す関数を使用することをおすすめします。そちらに示した関数を使用すれば、複数のグループの粒子を出力でき、グループごとに色の設定や粒子のサイズを変えて可視化することができます。

表 6.19　粒子ごとに値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_particle_pos2d_f	粒子の位置を出力する (2 次元)
cg_iric_write_sol_particle_pos3d_f	粒子の位置を出力する (3 次元)
cg_iric_write_sol_particle_integer_f	整数の粒子ごとに値を持つ計算結果を出力する
cg_iric_write_sol_particle_real_f	倍精度実数の粒子ごとに値を持つ計算結果を出力する

リスト 6.17　サンプルプログラム (粒子の座標ごとに値を持つ計算結果)

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  integer:: numparticles = 10
  double precision, dimension(:), allocatable:: particle_x, particle_y
  double precision, dimension(:), allocatable:: velocity_x, velocity_y, temperature
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  ! 計算結果を保持するメモリを確保
  allocate(particle_x(numparticles), particle_y(numparticles))
  allocate(velocity_x(numparticles), velocity_y(numparticles),␣
↪temperature(numparticles))

  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f (grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0
  call cg_iric_write_sol_time_f(time, ier)
  call cg_iric_write_sol_particle_pos2d_f(numparticles, particle_x, particle_y, ier)
  call cg_iric_write_sol_particle_real_f('VelocityX', velocity_x, ier)
  call cg_iric_write_sol_particle_real_f('VelocityY', velocity_y, ier)
  call cg_iric_write_sol_particle_real_f('Temperature', temperature, ier)
  do
    time = time + 10.0

```

(次のページに続く)

(前のページからの続き)

```
    ! (ここで計算を実行)

    call iric_check_cancel_f(canceled)
    if (canceled == 1) exit

    ! 計算結果を出力
    call iric_write_sol_start_f(condFile, ier)
    call cg_iric_write_sol_time_f(time, ier)
    call cg_iric_write_sol_particle_pos2d_f(numparticles, particle_x, particle_y, ier)
    call cg_iric_write_sol_particle_real_f('VelocityX', velocity_x, ier)
    call cg_iric_write_sol_particle_real_f('VelocityY', velocity_y, ier)
    call cg_iric_write_sol_particle_real_f('Temperature', temperature, ier)
    call cg_iric_flush_f(condFile, fin, ier)
    call iric_write_sol_end_f(condFile, ier)

    if (time > 1000) exit
  end do

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

粒子の座標ごとに値を持つ計算結果 (複数グループ可)

粒子の座標ごとに値を持つ計算結果を出力する場合、表 6.20 に示す関数を使用します。

ここで示す関数を使うと、複数のグループの粒子を出力することができます。各グループの出力の最初と最後で、*cg_iric_write_sol_particlegroup_groupbegin_f* と *cg_iric_write_sol_particlegroup_groupend_f* を呼び出してください。

出力するプログラムの例は、リスト 6.18 を参照してください。

注釈: **粒子の座標ごとに値を持つ計算結果** (ページ 142) で示した関数とは異なり、ここで示す関数では、一度の関数呼び出しでは粒子一つ分のデータを出力します。

表 6.20　粒子ごとに値を持つ計算結果の出力に利用する関数 (複数グループ可)

関数	備考
cg_iric_write_sol_particlegroup_groupbegin_f	粒子の計算結果の出力を開始する
cg_iric_write_sol_particlegroup_groupend_f	粒子の計算結果の出力を終了する
cg_iric_write_sol_particlegroup_pos2d_f	粒子の位置を出力する (2 次元)
cg_iric_write_sol_particlegroup_pos3d_f	粒子の位置を出力する (3 次元)
cg_iric_write_sol_particlegroup_integer_f	整数の粒子ごとに値を持つ計算結果を出力する
cg_iric_write_sol_particlegroup_real_f	倍精度実数の粒子ごとに値を持つ計算結果を出力する

リスト 6.18　サンプルプログラム (粒子の座標ごとに値を持つ計算結果 (複数グループ可))

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

  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  integer:: numparticles = 10
  double precision, dimension(:), allocatable:: particle_x, particle_y,
  double precision, dimension(:), allocatable:: velocity_x, velocity_y, temperature
  integer:: i
  integer:: status = 1
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  ! 計算結果を保持するメモリを確保。
  allocate(particle_x(numparticles), particle_y(numparticles))
  allocate(velocity_x(numparticles), velocity_y(numparticles),␣
↪temperature(numparticles))
```

(次のページに続く)

(前のページからの続き)

```
34
35    ! 格子を読み込む
36    call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)
37
38    ! 初期状態の情報を出力
39    time = 0
40    call cg_iric_write_sol_time_f(time, ier)
41
42    call cg_iric_write_sol_particlegroup_groupbegin_f('driftwood', ier)
43    do i = 1, numparticles
44      ! (ここで particle_x, particle_x, velocity_x, velocity_y, temperature に適切な値を設定)
45      call cg_iric_write_sol_particlegroup_pos2d_f(particle_x(i), particle_y(i), ier)
46      call cg_iric_write_sol_particlegroup_real_f('VelocityX', velocity_x(i), ier)
47      call cg_iric_write_sol_particlegroup_real_f('VelocityY', velocity_y(i), ier)
48      call cg_iric_write_sol_particlegroup_real_f('Temperature', temperature(i), ier)
49    end do
50    call cg_iric_write_sol_particlegroup_groupend_f(ier)
51
52    do
53      time = time + 10.0
54
55      ! (ここで計算を実行)
56
57      call iric_check_cancel_f(canceled)
58      if (canceled == 1) exit
59
60      ! 計算結果を出力
61      call iric_write_sol_start_f(condFile, ier)
62      call cg_iric_write_sol_time_f(time, ier)
63      call cg_iric_write_sol_particlegroup_groupbegin_f('driftwood', ier)
64      do i = 1, numparticles
65        ! (ここで particle_x, particle_x, velocity_x, velocity_y, temperature に適切な値を設
   定)
66        call cg_iric_write_sol_particlegroup_pos2d_f(particle_x(i), particle_y(i), ier)
67        call cg_iric_write_sol_particlegroup_real_f('VelocityX', velocity_x(i), ier)
68        call cg_iric_write_sol_particlegroup_real_f('VelocityY', velocity_y(i), ier)
69        call cg_iric_write_sol_particlegroup_real_f('Temperature', temperature(i), ier)
70      end do
71      call cg_iric_write_sol_particlegroup_groupend_f(ier)
72
73      if (time > 1000) exit
74    end do
75
76    ! CGNS ファイルのクローズ
77    call cg_close_f(fin, ier)
78    stop
79  end program SampleProgram
```

ポリゴンもしくは折れ線ごとに値を持つ計算結果

ポリゴンもしくは折れ線ごとに値を持つ計算結果を出力する場合、表 6.21 に示す関数を使用します。

ポリゴンもしくは折れ線では、複数のグループを出力することができます。各グループの出力の最初と最後で、*cg_iric_write_sol_polydata_groupbegin_f* と *cg_iric_write_sol_polydata_groupend_f* を呼び出してください。

出力するプログラムの例は、リスト 6.19 を参照してください。

注釈: 粒子の座標ごとに値を持つ計算結果を出力する関数では、一度の関数の呼び出して全ての粒子の座標や値を出力しますが、ポリゴンもしくは折れ線ごとに値を持つ計算結果では、一度の関数呼び出しではポリゴンもしくは折れ線一つ分のデータを出力します。

注釈: ポリゴンもしくは折れ線ごとに値を持つ計算結果は、2 次元にのみ対応しています。

注釈: 一つのグループの中にポリゴンと折れ線を混在させることもできます。

注釈: ポリゴンもしくは折れ線では、計算結果の値はスカラー量にのみ対応しています。

表 6.21　ポリゴンもしくは折れ線ごとに値を持つ計算結果の出力に利用する関数

関数	備考
cg_iric_write_sol_polydata_groupbegin_f	ポリゴンもしくは折れ線で定義された計算結果の出力を開始する
cg_iric_write_sol_polydata_groupend_f	ポリゴンもしくは折れ線で定義された計算結果の出力を終了する
cg_iric_write_sol_polydata_polygon_f	計算結果としてポリゴンの形状を出力する
cg_iric_write_sol_polydata_polyline_f	計算結果として折れ線の形状を出力する
cg_iric_write_sol_polydata_integer_f	整数のポリゴンもしくは折れ線ごとに値を持つ計算結果を出力する
cg_iric_write_sol_polydata_real_f	倍精度実数のポリゴンもしくは折れ線ごとに値を持つ計算結果を出力する

リスト 6.19　サンプルプログラム (ポリゴンもしくは折れ線ごとに値を持つ計算結果)

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

(次のページに続く)

(前のページからの続き)

```
  integer:: fin, ier, isize, jsize
  integer:: canceled
  integer:: locked
  double precision:: time
  double precision, dimension(:,:), allocatable::grid_x, grid_y
  integer:: numpolygons = 10
  integer:: numpoints = 5
  double precision, dimension(:), allocatable:: polydata_x, polydata_y,
  double precision:: temperature = 26
  integer:: i
  integer:: status = 1
  character(len=20):: condFile

  condFile = 'test.cgn'

  ! CGNS ファイルのオープン
  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 ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)
  ! 格子を読み込むためのメモリを確保
  allocate(grid_x(isize, jsize), grid_y(isize, jsize))
  ! 計算結果を保持するメモリを確保。一つのポリゴンの点数は 5 点
  allocate(polydata_x(numpoints), polydata_y(numpoints))

  ! 格子を読み込む
  call cg_iric_getgridcoord2d_f(grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0
  call cg_iric_write_sol_time_f(time, ier)

  call cg_iric_write_sol_polydata_groupbegin_f('fish', ier)
  do i = 1, numpolygons
    ! (ここで polydata_x, polydata_y, temperature, status に適切な値を設定)
    call cg_iric_write_sol_polydata_polygon_f(numpoints, polydata_x, polydata_y, ier)
    call cg_iric_write_sol_polydata_real_f('Temperature', temperature, ier)
    call cg_iric_write_sol_polydata_integer_f('Status', status, ier)
  end do
  call cg_iric_write_sol_polydata_groupend_f(ier)

  do
    time = time + 10.0
```

(次のページに続く)

(前のページからの続き)

```
53
54      ! (ここで計算を実行)
55
56      call iric_check_cancel_f(canceled)
57      if (canceled == 1) exit
58
59      ! 計算結果を出力
60      call iric_write_sol_start_f(condFile, ier)
61      call cg_iric_write_sol_time_f(time, ier)
62      call cg_iric_write_sol_polydata_groupbegin_f('fish', ier)
63      do i = 1, numpolygons
64        ! (ここで polydata_x, polydata_y, temperature, status に適切な値を設定)
65        call cg_iric_write_sol_polydata_polygon_f(numpoints, polydata_x, polydata_y, ier)
66        call cg_iric_write_sol_polydata_real_f('Temperature', temperature, ier)
67        call cg_iric_write_sol_polydata_integer_f('Status', status, ier)
68      end do
69      call cg_iric_write_sol_polydata_groupend_f(ier)
70
71      if (time > 1000) exit
72    end do
73
74    ! CGNS ファイルのクローズ
75    call cg_close_f(fin, ier)
76    stop
77  end program SampleProgram
```

6.4.13 既存の計算結果の読み込み

既存の CGNS ファイルに格納されている計算結果を読み込みます。

表 6.22　利用する関数

関数	備考
cg_iric_read_sol_count_f	計算結果の数を取得する
cg_iric_read_sol_time_f	計算結果の時刻の値を取得する
cg_iric_read_sol_iteration_f	計算結果のループ回数の値を取得する
cg_iric_read_sol_baseiterative_integer_f	整数の計算結果の値を取得する
cg_iric_read_sol_baseiterative_real_f	倍精度実数の計算結果の値を取得する
cg_iric_read_sol_gridcoord2d_f	計算結果の 2 次元構造格子を取得する
cg_iric_read_sol_gridcoord3d_f	計算結果の 3 次元構造格子を取得する
cg_iric_read_sol_integer_f	整数の格子点ごとに値を持つ計算結果の値を取得する
cg_iric_read_sol_real_f	倍精度実数の格子点ごとに値を持つ計算結果の値を取得する
cg_iric_read_sol_cell_integer_f	整数の格子セルごとに値を持つ計算結果の値を取得する
cg_iric_read_sol_cell_real_f	倍精度実数の格子セルごとに値を持つ計算結果の値を取得する

既存の CGNS ファイルを読み込み、格納されている計算結果を標準出力に出力する処理の例を リスト 6.20 に示します。

リスト 6.20　計算結果を読み込む処理の記述例

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

  integer:: fin, ier, isize, jsize, solid, solcount, iter, i, j
  double precision, dimension(:,:), allocatable::grid_x, grid_y, result_real

  ! CGNS ファイルのオープン
  call cg_open_f('test.cgn', CG_MODE_READ, fin, ier)
  if (ier /=0) STOP "*** Open error of CGNS file ***"

  ! 内部変数の初期化
  call cg_iric_initread_f(fin, ier)
  if (ier /=0) STOP "*** Initialize error of CGNS file ***"

  ! 格子のサイズを調べる
  call cg_iric_gotogridcoord2d_f(isize, jsize, ier)

  ! 計算結果を読み込むためのメモリを確保
  allocate(grid_x(isize,jsize), grid_y(isize,jsize))
  allocate(result_real(isize, jsize))

  ! 計算結果を読み込み出力
  call cg_iric_read_sol_count_f(solcount, ier)
  do solid = 1, solcount
    call cg_iric_read_sol_iteration_f(solid, iter, ier)
    call cg_iric_read_sol_gridcoord2d_f(solid, grid_x, grid_y, ier)
    call cg_iric_read_sol_real_f(solid, 'result_real', result_real, ier)

    print *, 'iteration: ', iter
    print *, 'grid_x, grid_y, result: '
    do i = 1, isize
      do j = 1, jsize
        print *, '(', i, ', ', j, ') = (', grid_x(i, j), ', ', grid_y(i, j), ', ',␣
↪result_real(i, j), ')'
      end do
    end do
  end do

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

なお、計算結果読み込みの関数を用いて、既存の CGNS ファイルの計算結果を分析・加工することができます（計

算結果分析ソルバーの開発手順 (ページ 35) 参照)。

6.4.14 エラーコードの出力

CGNS ファイルに、エラーコードを出力します。格子生成プログラムでのみ行います。

表 6.23　利用する関数

関数	備考
cg_iric_write_errorcode_f	エラーコードを出力する。

6.4.15 CGNS ファイルを閉じる

cg_open_f で開いた CGNS ファイルを閉じます。この関数は、 cgnslib で定義された関数です。

表 6.24　利用する関数

関数	備考
cg_close_f	CGNS ファイルを閉じる。

6.5 リファレンス

リファレンスでは、各関数の機能、呼び出す際の形式、引数について解説します。

リファレンスの各関数のページでは、引数の型は FORTRAN の場合について解説しています。C/C++, Python から呼び出す際の引数の型については、表 6.25 を参照して読み替えてください。

Python では、エラーコードを格納する ier は出力されず、エラーが発生した場合は例外が発生します。エラー処理を行う場合は、try, except を利用してください。

表 6.25　引数の型の対応関係

FORTRAN	C/C++	Python
integer	int (出力の場合 int*)	int
double precision	double (出力の場合 double*)	float
real	float (出力の場合 float*)	(なし)
character(*)	char*	str
integer, dimension(:), allocatable	int*	numpy.ndarray(dtype=int32)
double precision, dimension(:), allocatable	double*	numpy.ndarray(dtype=float64)
real, dimension(:), allocatable	float*	(なし)

6.5.1 サブルーチン一覧

サブルーチンとその分類の一覧を 表 6.26 に示します。

表 6.26: iRIClib サブルーチン一覧

分類	名前	機能
CGNS ファイルを開く	cg_open_f	CGNS ファイルを開く
内部変数の初期化	cg_iric_init_f	指定したファイルを読み込み・書き
内部変数の初期化	cg_iric_initread_f	指定したファイルを読み込み専用で
オプションの設定	cg_initoption_f	ソルバーのオプションを設定する
計算条件、格子生成条件の読み込み	cg_iric_read_integer_f	整数型変数の値を取得する
計算条件、格子生成条件の読み込み	cg_iric_read_real_f	実数 (倍精度) 変数の値を取得する
計算条件、格子生成条件の読み込み	cg_iric_read_realsingle_f	実数 (単精度) 変数の値を取得する
計算条件、格子生成条件の読み込み	cg_iric_read_string_f	文字列型変数の値を取得する
計算条件、格子生成条件の読み込み	cg_iric_read_functionalsize_f	関数型変数のサイズを取得する
計算条件、格子生成条件の読み込み	cg_iric_read_functional_f	倍精度実数の関数型変数の値を取得
計算条件、格子生成条件の読み込み	cg_iric_read_functional_realsingle_f	単精度実数の関数型変数の値を取得
計算条件、格子生成条件の読み込み	cg_iric_read_functionalwithname_f	複数の値を持つ倍精度実数の関数型
計算格子の読み込み	cg_iric_gotogridcoord2d_f	格子を読み込む準備をする
計算格子の読み込み	cg_iric_gotogridcoord3d_f	格子を読み込む準備をする
計算格子の読み込み	cg_iric_getgridcoord2d_f	格子の X, Y 座標を読み込む
計算格子の読み込み	cg_iric_getgridcoord3d_f	格子の X, Y, Z 座標を読み込む
計算格子の読み込み	cg_iric_read_grid_integer_node_f	格子点で定義された整数の属性を読
計算格子の読み込み	cg_iric_read_grid_real_node_f	格子点で定義された倍精度実数の属
計算格子の読み込み	cg_iric_read_grid_integer_cell_f	セルで定義された整数の属性を読み
計算格子の読み込み	cg_iric_read_grid_real_cell_f	セルで定義された倍精度実数の属性
計算格子の読み込み	cg_iric_read_complex_count_f	複合型の属性のグループの数を読み
計算格子の読み込み	cg_iric_read_complex_integer_f	複合型の属性の整数の条件を読み込
計算格子の読み込み	cg_iric_read_complex_real_f	複合型の属性の倍精度実数の条件を
計算格子の読み込み	cg_iric_read_complex_realsingle_f	複合型の属性の単精度実数の条件を
計算格子の読み込み	cg_iric_read_complex_string_f	複合型の属性の文字列の条件を読み
計算格子の読み込み	cg_iric_read_complex_functionalsize_f	複合型の属性の関数型の条件のサイ
計算格子の読み込み	cg_iric_read_complex_functional_f	複合型の属性の倍精度実数の関数型
計算格子の読み込み	cg_iric_read_complex_functionalwithname_f	複合型の属性の単精度実数の関数型
計算格子の読み込み	cg_iric_read_complex_functional_realsingle_f	複合型の属性の値を複数持つ倍精度
計算格子の読み込み	cg_iric_read_grid_complex_node_f	格子点で定義された複合型の属性を
計算格子の読み込み	cg_iric_read_grid_complex_cell_f	セルで定義された複合型の属性を読
計算格子の読み込み	cg_iric_read_grid_functionaltimesize_f	次元「時刻」(Time) を持つ格子属性

表 6.26 – 前のページからの続き

分類	名前	機能
計算格子の読み込み	cg_iric_read_grid_functionaltime_f	次元「時刻」(Time) の値を読み込む
計算格子の読み込み	cg_iric_read_grid_functionaldimensionsize_f	次元の数を調べる
計算格子の読み込み	cg_iric_read_grid_functionaldimension_integer_f	整数の次元の値を読み込む
計算格子の読み込み	cg_iric_read_grid_functionaldimension_real_f	倍精度実数の次元の値を読み込む
計算格子の読み込み	cg_iric_read_grid_functional_integer_node_f	次元「時刻」を持つ、格子点で定義
計算格子の読み込み	cg_iric_read_grid_functional_real_node_f	次元「時刻」を持つ、格子点で定義
計算格子の読み込み	cg_iric_read_grid_functional_integer_cell_f	次元「時刻」を持つ、セルで定義さ
計算格子の読み込み	cg_iric_read_grid_functional_real_cell_f	次元「時刻」を持つ、セルで定義さ
境界条件の読み込み	cg_iric_read_bc_count_f	境界条件の数を取得する
境界条件の読み込み	cg_iric_read_bc_indicessize_f	境界条件の設定された要素 (格子点も
境界条件の読み込み	cg_iric_read_bc_indices_f	境界条件の設定された要素 (格子点も
境界条件の読み込み	cg_iric_read_bc_integer_f	整数型境界条件の値を取得する
境界条件の読み込み	cg_iric_read_bc_real_f	実数 (倍精度) 境界条件の値を取得す
境界条件の読み込み	cg_iric_read_bc_realsingle_f	実数 (単精度) 境界条件の値を取得す
境界条件の読み込み	cg_iric_read_bc_string_f	文字列型境界条件の値を取得する
境界条件の読み込み	cg_iric_read_bc_functionalsize_f	関数型境界条件のサイズを取得する
境界条件の読み込み	cg_iric_read_bc_functional_f	倍精度実数の関数型境界条件の値を
境界条件の読み込み	cg_iric_read_bc_functional_realsingle_f	単精度実数の関数型境界条件の値を
境界条件の読み込み	cg_iric_read_bc_functionalwithname_f	複数の値を持つ倍精度実数の関数型
地形データの読み込み	cg_iric_read_geo_count_f	地形データの数を返す
地形データの読み込み	cg_iric_read_geo_filename_f	地形データのファイル名と種類を返
地形データの読み込み	iric_geo_polygon_open_f	ポリゴンファイルを開く
地形データの読み込み	iric_geo_polygon_read_integervalue_f	ポリゴンの値を整数で返す
地形データの読み込み	iric_geo_polygon_read_realvalue_f	ポリゴンの値を実数で返す
地形データの読み込み	iric_geo_polygon_read_pointcount_f	ポリゴンの頂点の数を返す
地形データの読み込み	iric_geo_polygon_read_points_f	ポリゴンの頂点の座標を返す
地形データの読み込み	iric_geo_polygon_read_holecount_f	ポリゴンに開いた穴の数を返す
地形データの読み込み	iric_geo_polygon_read_holepointcount_f	ポリゴンの穴の頂点の数を返す
地形データの読み込み	iric_geo_polygon_read_holepoints_f	ポリゴンの穴の頂点の座標を返す
地形データの読み込み	iric_geo_polygon_close_f	ポリゴンファイルを閉じる
地形データの読み込み	iric_geo_riversurvey_open_f	河川測量データを開く
地形データの読み込み	iric_geo_riversurvey_read_count_f	河川横断線の数を返す
地形データの読み込み	iric_geo_riversurvey_read_position_f	横断線の中心点の座標を返す
地形データの読み込み	iric_geo_riversurvey_read_direction_f	横断線の向きを返す
地形データの読み込み	iric_geo_riversurvey_read_name_f	横断線の名前を文字列として返す
地形データの読み込み	iric_geo_riversurvey_read_realname_f	横断線の名前を実数値として返す
地形データの読み込み	iric_geo_riversurvey_read_leftshift_f	横断線の標高データのシフト量を返

表 6.26 – 前のページからの続き

分類	名前	機能
地形データの読み込み	iric_geo_riversurvey_read_altitudecount_f	横断線の標高データの数を返す
地形データの読み込み	iric_geo_riversurvey_read_altitudes_f	横断線の標高データを返す
地形データの読み込み	iric_geo_riversurvey_read_fixedpointl_f	横断線の左岸延長線のデータを返す
地形データの読み込み	iric_geo_riversurvey_read_fixedpointr_f	横断線の右岸延長線のデータを返す
地形データの読み込み	iric_geo_riversurvey_read_watersurfaceelevation_f	横断線での水面標高のデータを返す
地形データの読み込み	iric_geo_riversurvey_close_f	河川測量データを閉じる
計算格子の出力	cg_iric_writegridcoord1d_f	1 次元構造格子を出力する
計算格子の出力	cg_iric_writegridcoord2d_f	2 次元構造格子を出力する
計算格子の出力	cg_iric_writegridcoord3d_f	3 次元構造格子を出力する
計算格子の出力	cg_iric_write_grid_integer_node_f	格子点で定義された整数の属性を出
計算格子の出力	cg_iric_write_grid_real_node_f	格子点で定義された倍精度実数の属
計算格子の出力	cg_iric_write_grid_integer_cell_f	セルで定義された整数の属性を出力
計算格子の出力	cg_iric_write_grid_real_cell_f	セルで定義された倍精度実数の属性
時刻 (ループ回数) の出力	cg_iric_write_sol_time_f	時刻を出力する
時刻 (ループ回数) の出力	cg_iric_write_sol_iteration_f	ループ回数を出力する
計算結果の出力	cg_iric_write_sol_gridcoord2d_f	2 次元構造格子を出力する
計算結果の出力	cg_iric_write_sol_gridcoord3d_f	3 次元構造格子を出力する
計算結果の出力	cg_iric_write_sol_baseiterative_integer_f	整数の計算結果を出力する
計算結果の出力	cg_iric_write_sol_baseiterative_real_f	倍精度実数の計算結果を出力する
計算結果の出力	cg_iric_write_sol_baseiterative_string_f	文字列の計算結果を出力する
計算結果の出力	cg_iric_write_sol_integer_f	整数の格子点ごとに値を持つ計算結
計算結果の出力	cg_iric_write_sol_real_f	倍精度実数の格子点ごとに値を持つ
計算結果の出力	cg_iric_write_sol_cell_integer_f	整数の格子セルごとに値を持つ計算
計算結果の出力	cg_iric_write_sol_cell_real_f	倍精度実数の格子セルごとに値を持
計算結果の出力	cg_iric_write_sol_iface_integer_f	整数の I 方向格子エッジごとに値を
計算結果の出力	cg_iric_write_sol_iface_real_f	倍精度実数の I 方向格子エッジごと
計算結果の出力	cg_iric_write_sol_jface_integer_f	整数の J 方向格子エッジごとに値を
計算結果の出力	cg_iric_write_sol_jface_real_f	倍精度実数の J 方向格子エッジごと
計算結果の出力 (粒子)	cg_iric_write_sol_particle_pos2d_f	粒子の位置を出力する (2 次元)
計算結果の出力 (粒子)	cg_iric_write_sol_particle_pos3d_f	粒子の位置を出力する (3 次元)
計算結果の出力 (粒子)	cg_iric_write_sol_particle_integer_f	整数の粒子ごとに値を持つ計算結果
計算結果の出力 (粒子)	cg_iric_write_sol_particle_real_f	倍精度実数の粒子ごとに値を持つ計
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_groupbegin_f	粒子で定義された計算結果の出力を
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_groupend_f	粒子で定義された計算結果の出力を
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_pos2d_f	粒子の位置を出力する (2 次元)
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_pos3d_f	粒子の位置を出力する (3 次元)
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_integer_f	整数の粒子ごとに値を持つ計算結果

表 6.26 – 前のページからの続き

分類	名前	機能
計算結果の出力 (粒子)	cg_iric_write_sol_particlegroup_real_f	倍精度実数の粒子ごとに値を持つ計
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_groupbegin_f	ポリゴンもしくは折れ線で定義され
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_groupend_f	ポリゴンもしくは折れ線で定義され
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_polygon_f	計算結果としてポリゴンの形状を出
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_polyline_f	計算結果として折れ線の形状を出力
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_integer_f	整数のポリゴンもしくは折れ線ごと
計算結果の出力 (ポリゴン・折れ線)	cg_iric_write_sol_polydata_real_f	倍精度実数のポリゴンもしくは折れ
計算結果の出力の前後に利用する関数	iric_check_cancel_f	ユーザがソルバーの実行をキャンセ
計算結果の出力の前後に利用する関数	iric_check_lock_f	CGNS ファイルが GUI によってロッ
計算結果の出力の前後に利用する関数	iric_write_sol_start_f	計算結果の出力開始を GUI に通知す
計算結果の出力の前後に利用する関数	iric_write_sol_end_f	計算結果の出力終了を GUI に通知す
計算結果の出力の前後に利用する関数	cg_iric_flush_f	計算結果の出力をファイルに書き込
既存の計算結果の読み込み	cg_iric_read_sol_count_f	計算結果の数を取得する
既存の計算結果の読み込み	cg_iric_read_sol_time_f	計算結果の時刻の値を取得する
既存の計算結果の読み込み	cg_iric_read_sol_iteration_f	計算結果のループ回数の値を取得す
既存の計算結果の読み込み	cg_iric_read_sol_baseiterative_integer_f	整数の計算結果の値を取得する
既存の計算結果の読み込み	cg_iric_read_sol_baseiterative_real_f	倍精度実数の計算結果の値を取得す
既存の計算結果の読み込み	cg_iric_read_sol_baseiterative_string_f	文字列の計算結果の値を取得する
既存の計算結果の読み込み	cg_iric_read_sol_gridcoord2d_f	計算結果の 2 次元構造格子を取得す
既存の計算結果の読み込み	cg_iric_read_sol_gridcoord3d_f	計算結果の 3 次元構造格子を取得す
既存の計算結果の読み込み	cg_iric_read_sol_integer_f	整数の格子点ごとに値を持つ計算結
既存の計算結果の読み込み	cg_iric_read_sol_real_f	倍精度実数の格子点ごとに値を持つ
既存の計算結果の読み込み	cg_iric_read_sol_cell_integer_f	整数の格子セルごとに値を持つ計算
既存の計算結果の読み込み	cg_iric_read_sol_cell_real_f	倍精度実数の格子セルごとに値を持
既存の計算結果の読み込み	cg_iric_read_sol_iface_integer_f	整数の I 方向格子エッジごとに値を
既存の計算結果の読み込み	cg_iric_read_sol_iface_real_f	倍精度実数の I 方向格子エッジごと
既存の計算結果の読み込み	cg_iric_read_sol_jface_integer_f	整数の J 方向格子エッジごとに値を
既存の計算結果の読み込み	cg_iric_read_sol_jface_real_f	倍精度実数の J 方向格子エッジごと
エラーコードの出力	cg_iric_write_errorcode_f	エラーコードを出力する
CGNS ファイルを閉じる	cg_close_f	CGNS ファイルを閉じる

「複数版」欄が「○」となっているサブルーチン (単一 CGNS ファイル用) には、ファイル ID を第一引数とする、類似のサブルーチン (複数 CGNS ファイル用) があります。名前は、末尾の "_f" を "_mul_f" に変えたものです。

例えば、CGNS ファイルから整数型の計算条件・格子生成条件の値を読み込む関数には、以下のものがあります。

- 単一 CGNS ファイルを扱うプログラム用

```
call cg_iric_read_integer_f(label, intvalue, ier)
```

- 複数 CGNS ファイルを扱うプログラム用

```
call cg_iric_read_integer_mul_f(fid, label, intvalue, ier)
```

単一 CGNS ファイル用、複数 CGNS ファイル用の違いを 表 6.27 に示します。

表 6.27　単一・複数 CGNS ファイル用サブルーチンの違い

項目	単一 CGNS ファイル用	複数 CGNS ファイル用
名前	末尾が "_f" ("_mul" が付かない)	末尾が "_mul_f"
引数	次節以降参照	第一引数 = ファイル ID (integer)
操作対象ファイル	最後に cg_iric_init_f または cg_iric_initread_f で指定したファイル	第一引数で指定したファイル

6.5.2 cg_open_f

CGNS ファイルを開く。

形式 (FORTRAN)

```
call cg_open_f(filename, mode, fid, ier)
```

形式 (C/C++)

```
ier = cg_open(filename, mode, fid);
```

形式 (Python)

```
fid = cg_open(filename, mode)
```

引数

表 6.28　cg_open_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
mode	integer	I	オープンモード
fid	integer	O	ファイル ID
ier	integer	O	エラーコード。0 なら成功

備考

表 6.29　mode の値

オープンモード	内容
CG_MODE_MODIFY	読み書き可
CG_MODE_READ	読み込みのみ

6.5.3 cg_iric_init_f

指定したファイルを読み込み・書き込み用に iRIClib から利用するため、内部変数を初期化し、ファイルを初期化する。

形式 (FORTRAN)

```
call cg_iric_init_f(fid, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Init(fid);
```

形式 (Python)

```
cg_iRIC_Init(fid)
```

引数

表 6.30　cg_iric_init_f の引数

変数名	型	I/O	内容
fid	integer	I	ファイル ID
ier	integer	O	エラーコード。0 なら成功。ただし、格子生成プログラムで利用する場合は、 1 で成功。

6.5.4 cg_iric_initread_f

指定したファイルを読み込み専用で iRIClib から利用するため、内部変数を初期化する。

形式 (FORTRAN)

```
call cg_iric_initread_f(fid, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_InitRead(fid);
```

形式 (Python)

```
cg_iRIC_InitRead(fid)
```

引数

表 6.31　cg_iric_initread_f の引数

変数名	型	I/O	内容
fid	integer	I	ファイル ID
ier	integer	O	エラーコード。0 なら成功。ただし、格子生成プログラムで利用する場合は、 1 で成功。

6.5.5 iric_initoption_f

ソルバーのオプションを指定する。

形式 (FORTRAN)

```
call iric_initoption_f(optionval, ier)
```

形式 (C/C++)

```
ier = iRIC_InitOption(optionval);
```

形式 (Python)

```
iRIC_InitOption(optionval)
```

引数

表 6.32　iric_initoption_f の引数

変数名	型	I/O	内容
optionval	integer	I	指定するオプションの値
ier	integer	O	エラーコード。0 なら成功

備考

表 6.33　optionval の値

optionval の値	内容
IRIC_OPTION_CANCEL	iric_check_cancel_f を利用してキャンセルを検知できることを GUI に伝えます。
IRIC_OPTION_DIVIDESOLUTIONS	計算結果を複数の CGNS ファイルに分割して保存する。

6.5.6 cg_iric_read_integer_f

CGNS ファイルから整数型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_integer_f(label, intvalue, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Integer(label, &intvalue);
```

形式 (Python)

```
intvalue = cg_iRIC_Read_Integer(label)
```

引数

表 6.34　cg_iric_read_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
intvalue	integer	O	CGNS ファイルから読み込まれた整数
ier	integer	O	エラーコード。0 なら成功

6.5.7 cg_iric_read_real_f

CGNS ファイルから倍精度の実数型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_real_f(label, realvalue, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Real(label, &realvalue);
```

形式 (Python)

```
realvalue = cg_iRIC_Read_Real(label)
```

引数

表 6.35　cg_iric_read_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
realvalue	double precision	O	CGNS ファイルから読み込まれた実数
ier	integer	O	エラーコード。0 なら成功

6.5.8 cg_iric_read_realsingle_f

CGNS ファイルから単精度の実数型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_realsingle_f(label, realvalue, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_RealSingle(label, &realvalue);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.36　cg_iric_read_realsingle_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
realvalue	real	O	CGNS ファイルから読み込まれた実数
ier	integer	O	エラーコード。0 なら成功

6.5.9 cg_iric_read_string_f

CGNS ファイルから文字列型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_string_f(label, strvalue, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_String(label, strvalue);
```

形式 (Python)

```
strvalue = cg_iRIC_Read_String(label)
```

引数

表 6.37　cg_iric_read_string_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
strvalue	character(*)	O	CGNS ファイルから読み込まれた文字列
ier	integer	O	エラーコード。0 なら成功

6.5.10 cg_iric_read_functionalsize_f

CGNS ファイルから関数型の計算条件・格子生成条件のサイズを読み込む。

形式 (FORTRAN)

```
call cg_iric_read_functionalsize_f(label, size, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_FunctionalSize(label, &size);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.38　cg_iric_read_functionalsize_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
size	integer	O	CGNS ファイルから読み込まれた配列の長さ
ier	integer	O	エラーコード。0 なら成功

6.5.11 cg_iric_read_functional_f

CGNS ファイルから倍精度実数の関数型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_functional_f(label, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Functional(label, x, y);
```

形式 (Python)

```
x, y = cg_iRIC_Read_Functional(label)
```

引数

表 6.39　cg_iric_read_functional_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
x	double precision, dimension(:), allocatable	O	X の値の配列
y	double precision, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.12 cg_iric_read_functional_realsingle_f

CGNS ファイルから単精度実数の関数型の計算条件・格子生成条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_functional_realsingle_f(label, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Functional_RealSingle(label, x, y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.40　cg_iric_read_functional_realsingle_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
x	real, dimension(:), allocatable	O	X の値の配列
y	real, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.13 cg_iric_read_functionalwithname_f

CGNS ファイルから関数型の計算条件・格子生成条件の値を読み込む。変数が 1 つ、値が複数の関数型の計算条件・格子生成条件の読み込みに利用する。

形式 (FORTRAN)

```
call cg_iric_read_functionalwithname_f(label, name, data, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_FunctionalWithName(label, name, data);
```

形式 (Python)

```
data = cg_iRIC_Read_FunctionalWithName(label, name)
```

引数

表 6.41　cg_iric_read_functionalwithname_f の引数

変数名	型	I/O	内容
label	character(*)	I	ソルバー定義ファイルで定義した変数名
name	character(*)	I	ソルバー定義ファイルで定義した値の名前
data	double precision, dimension(:), allocatable	O	値の配列
Ier	integer	O	エラーコード。0 なら成功

6.5.14 cg_iric_gotogridcoord2d_f

二次元構造格子を読み込む準備を行う。

形式 (FORTRAN)

```
call cg_iric_gotogridcoord2d_f(nx, ny, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_GotoGridCoord2d(nx, ny);
```

形式 (Python)

```
nx, ny = cg_iRIC_GotoGridCoord2d()
```

引数

表 6.42　cg_iric_gotogridcoord2d_f の引数

変数名	型	I/O	内容
nx	integer	O	i 方向格子点数
ny	integer	O	j 方向格子点数
ier	integer	O	エラーコード。0 なら成功

6.5.15 cg_iric_gotogridcoord3d_f

三次元構造格子を読み込む準備を行う。

形式 (FORTRAN)

```
call cg_iric_gotogridcoord3d_f(nx, ny, nz, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_GotoGridCoord3d(nx, ny, nz);
```

形式 (Python)

```
nx, ny, nz = cg_iRIC_GotoGridCoord3d()
```

引数

表 6.43　cg_iric_gotogridcoord3d_f の引数

変数名	型	I/O	内容
nx	integer	O	i 方向格子点数
ny	integer	O	j 方向格子点数
nz	integer	O	k 方向格子点数
ier	integer	O	エラーコード。0 なら成功

6.5.16 cg_iric_getgridcoord2d_f

二次元構造格子を読み込む。

形式 (FORTRAN)

```
call cg_iric_getgridcoord2d_f(x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_GetGridCoord2d(x, y);
```

形式 (Python)

```
x, y = cg_iRIC_GetGridCoord2d()
```

引数

表 6.44　cg_iric_getgridcoord2d_f の引数

変数名	型	I/O	内容
x	double precision, dimension(:), allocatable	O	格子点の x 座標値
y	double precision, dimension(:), allocatable	O	格子点の y 座標値
ier	integer	O	エラーコード。0 なら成功

6.5.17 cg_iric_getgridcoord3d_f

三次元構造格子を読み込む。

形式 (FORTRAN)

```
call cg_iric_getgridcoord3d_f(x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_GetGridCoord3d(x, y, z);
```

形式 (Python)

```
x, y, z = cg_iRIC_GetGridCoord3d()
```

引数

表 6.45　cg_iric_getgridcoord3d_f の引数

変数名	型	I/O	内容
x	double precision, dimension(:), allocatable	O	格子点の x 座標値
y	double precision, dimension(:), allocatable	O	格子点の y 座標値
z	double precision, dimension(:), allocatable	O	格子点の z 座標値
ier	integer	O	エラーコード。0 なら成功

6.5.18 cg_iric_read_grid_integer_node_f

構造格子の格子点で定義された整数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_integer_node_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Integer_Node(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Integer_Node(label)
```

引数

表 6.46　cg_iric_read_grid_integer_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.19 cg_iric_read_grid_real_node_f

構造格子の格子点で定義された倍精度実数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_real_node_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Real_Node(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Real_Node(label)
```

引数

表 6.47　cg_iric_read_grid_real_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	double precision, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.20 cg_iric_read_grid_integer_cell_f

構造格子のセルで定義された整数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_integer_cell_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Integer_Cell(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Integer_Cell(label)
```

引数

表 6.48　cg_iric_read_grid_integer_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.21 cg_iric_read_grid_real_cell_f

構造格子のセルで定義された倍精度実数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_real_cell_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Real_Cell(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Real_Cell(label)
```

引数

表 6.49　cg_iric_read_grid_real_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	double precision, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.22 cg_iric_read_complex_count_f

複合型格子属性の、グループの数を取得する。

形式 (FORTRAN)

```
call cg_iric_read_complex_count_f(type, num, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_Count(type, &num);
```

形式 (Python)

```
num = cg_iRIC_Read_Complex_Count(type)
```

引数

表 6.50　cg_iric_read_complex_count_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	O	グループの数
ier	integer	O	エラーコード。0 なら成功

6.5.23 cg_iric_read_complex_integer_f

複合型格子属性の、整数型の条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_integer_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_Integer(type, num, name, &value);
```

形式 (Python)

```
value = cg_iRIC_Read_Complex_Integer(type, num, name)
```

引数

表 6.51　cg_iric_read_complex_integer_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
value	integer	O	読み込まれた条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.24 cg_iric_read_complex_real_f

複合型格子属性の、実数 (倍精度) 型の条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_real_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_Real(type, num, name, &value);
```

形式 (Python)

```
value = cg_iRIC_Read_Complex_Real(type, num, name)
```

引数

表 6.52　cg_iric_read_complex_real_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
value	double precision	O	読み込まれた条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.25 cg_iric_read_complex_realsingle_f

複合型格子属性の、実数 (単精度) 型の条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_realsingle_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_RealSingle(type, num, name, &value);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.53　cg_iric_read_complex_realsingle_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
value	real	O	読み込まれた条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.26 cg_iric_read_complex_string_f

複合型格子属性の、文字列型の条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_string_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_String(type, num, name, value);
```

形式 (Python)

```
value = cg_iRIC_Read_Complex_String(type, num, name)
```

引数

表 6.54　cg_iric_read_complex_string_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
value	character(*)	O	読み込まれた条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.27 cg_iric_read_complex_functionalsize_f

複合型格子属性の、関数型の条件の変数のサイズを読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_functionalsize_f(type, num, name, size, ier)
```

形式 (C/C++)

```
ier = cg_iric_read_complexunctionalsize(type, num, name, &size);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.55　cg_iric_read_complex_functionalsize_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
size	integer	O	条件の配列の長さ
ier	integer	O	エラーコード。0 なら成功

6.5.28 cg_iric_read_complex_functional_f

複合型格子属性の、倍精度関数型の条件の変数の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_functional_f(type, num, name, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iric_read_complexunctional(type, num, name, x, y);
```

形式 (Python)

```
x, y = cg_iric_read_complexunctional(type, num, name)
```

引数

表 6.56　cg_iric_read_complex_functional_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
x	double precision, dimension(:), allocatable	O	X の値の配列
y	double precision, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.29 cg_iric_read_complex_functional_realsingle_f

複合型格子属性の、単精度関数型の条件の変数の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_complex_functional_realsingle_f(type, num, name, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_Functional_RealSingle(type, num, name, x, y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.57　cg_iric_read_complex_functional_realsingle_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
x	real, dimension(:), allocatable	O	X の値の配列
y	real, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.30 cg_iric_read_complex_functionalwithname_f

複合型格子属性の、倍精度関数型の条件の変数の値を読み込む。変数が 1 つ、値が複数の関数型の境界条件の読み込みに利用する。

形式 (FORTRAN)

```
call cg_iric_read_complex_functionalwithname_f(type, num, name, paramname, data, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Complex_FunctionalWithName(type, num, name, paramname, data);
```

形式 (Python)

```
data = cg_iRIC_Read_Complex_FunctionalWithName(type, num, name, paramname)
```

引数

表 6.58　cg_iric_read_complex_functionalwithname_f の引数

変数名	型	I/O	内容
type	character(*)	I	属性名
num	integer	I	グループの番号
name	character(*)	I	条件の名前
paramname	character(*)	I	値の名前
data	double precision, dimension(:), allocatable	O	値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.31 cg_iric_read_grid_complex_node_f

構造格子の格子点で定義された複合型の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_complex_node_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Complex_Node(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Complex_Node(label)
```

引数

表 6.59　cg_iric_read_grid_complex_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.32 cg_iric_read_grid_complex_cell_f

構造格子のセルで定義された複合型の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_complex_cell_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Complex_Cell(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Complex_Cell(label)
```

引数

表 6.60　cg_iric_read_grid_complex_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.33 cg_iric_read_grid_functionaltimesize_f

次元「時刻」(Time) を持つ格子属性の、時刻の数を調べる。

形式 (FORTRAN)

```
call cg_iric_read_grid_functionaltimesize_f(label, count, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_FunctionalTimeSize(label, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.61　cg_iric_read_grid_functionaltimesize_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
count	integer	O	時刻の数
ier	integer	O	エラーコード。0 なら成功

6.5.34 cg_iric_read_grid_functionaltime_f

次元「時刻」(Time) の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_functionaltime_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_FunctionalTime(label, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_FunctionalTime(label)
```

引数

表 6.62　cg_iric_read_grid_functionaltime_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	double precision, dimension(:), allocatable	O	時刻の値
ier	integer	O	エラーコード。0 なら成功

6.5.35 cg_iric_read_grid_functionaldimensionsize_f

次元の数を調べる。

形式 (FORTRAN)

```
call cg_iric_read_grid_functionaltime_f(label, dimname, count, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_FunctionalTime(label, dimname, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.63　cg_iric_read_grid_functionaldimensionsize_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimname	character(*)	I	次元名
count	integer	O	時刻の数
ier	integer	O	エラーコード。0 なら成功

6.5.36 cg_iric_read_grid_functionaldimension_integer_f

整数の次元の値を読み込む

形式 (FORTRAN)

```
call cg_iric_read_grid_functionaldimension_integer_f(label, dimname, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_FunctionalDimension_Integer(label, dimname, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_FunctionalDimension_Integer(label, dimname)
```

引数

表 6.64　cg_iric_read_grid_functionaldimension_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimname	character(*)	I	次元名
values	integer, dimension(:), allocatable	O	次元の値
ier	integer	O	エラーコード。0 なら成功

6.5.37 cg_iric_read_grid_functionaldimension_real_f

実数の次元の値を読み込む

形式 (FORTRAN)

```
call cg_iric_read_grid_functionaldimension_real_f(label, dimname, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_FunctionalDimension_Real(label, dimname, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_FunctionalDimension_Real(label, dimname)
```

引数

表 6.65　cg_iric_read_grid_functionaldimension_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimname	character(*)	I	次元名
values	double precision, dimension(:), allocatable	O	時刻の値
ier	integer	O	エラーコード。0 なら成功

6.5.38 cg_iric_read_grid_functional_integer_node_f

次元「時刻」を持つ、格子点で定義された整数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_functional_integer_node_f(label, dimid, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Functional_Integer_Node(label, dimid, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Functional_Integer_Node(label, dimid)
```

引数

表 6.66　cg_iric_read_grid_functional_integer_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimid	integer	I	時刻の ID (1 〜 時刻の数)
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.39 cg_iric_read_grid_functional_real_node_f

次元「時刻」を持つ、格子点で定義された倍精度実数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_functional_real_node_f(label, dimid, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Functional_Real_Node(label, dimid, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Functional_Real_Node(label, dimid)
```

引数

表 6.67　cg_iric_read_grid_functional_real_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimid	integer	I	時刻の ID (1 〜 時刻の数)
values	double precision, dimension(:), allocatable	O	属性値
Ier	integer	O	エラーコード。0 なら成功

6.5.40 cg_iric_read_grid_functional_integer_cell_f

次元「時刻」を持つ、セルで定義された整数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_functional_integer_cell_f(label, dimid, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Functional_Integer_Cell(label, dimid, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Functional_Integer_Cell(label, dimid)
```

引数

表 6.68　cg_iric_read_grid_functional_integer_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimid	integer	I	時刻の ID (1 〜 時刻の数)
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.41 cg_iric_read_grid_functional_real_cell_f

次元「時刻」を持つ、セルで定義された倍精度実数の属性を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_grid_functional_real_cell_f(label, dimid, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Grid_Functional_Real_Cell(label, dimid, values);
```

形式 (Python)

```
values = cg_iRIC_Read_Grid_Functional_Real_Cell(label, dimid)
```

引数

表 6.69　cg_iric_read_grid_functional_real_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
dimid	integer	I	時刻の ID (1 〜 時刻の数)
values	double precision, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.42 cg_iric_read_bc_count_f

境界条件の数を取得する。

形式 (FORTRAN)

```
call cg_iric_read_bc_count_f(type, num)
```

形式 (C/C++)

```
cg_iRIC_Read_BC_Count(type, &num);
```

形式 (Python)

```
num = cg_iRIC_Read_BC_Count(type)
```

引数

表 6.70　cg_iric_read_bc_count_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	O	境界条件の数

6.5.43 cg_iric_read_bc_indicessize_f

境界条件が設定された要素 (格子点もしくはセル) の数を取得する。

形式 (FORTRAN)

```
call cg_iric_read_bc_indicessize_f(type, num, size, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_IndicesSize(type, num, &size);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.71　cg_iric_read_bc_indicessize_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
size	integer	O	境界条件が設定された要素の数
ier	integer	O	エラーコード。0 なら成功

備考

size に返される値は、境界条件が設定される位置によって、 表 6.72 に示すように異なります。

表 6.72　境界条件を設定された位置と size に返される値の関係

境界条件を設定された位置	size に返される値
格子点 (node)	格子点の数
セル (cell)	セルの数
辺 (edge)	辺の数× 2

6.5.44 cg_iric_read_bc_indices_f

境界条件が設定された要素 (格子点もしくはセル)

形式 (FORTRAN)

```
call cg_iric_read_bc_indices_f(type, num, indices, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_Indices(type, num, indices);
```

形式 (Python)

```
indices = cg_iRIC_Read_BC_Indices(type, num)
```

引数

表 6.73　cg_iric_read_bc_indices_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
indices	integer, dimension(2,:), allocatable	O	境界条件が設定された要素のインデックスの配列。
ier	integer	O	エラーコード。0 なら成功

備考

indices に返される値は、境界条件が設定される位置によって、表 6.74 に示すように異なります。格子点、セルでは、値 2 つで一つの要素を定義しているのに対し、辺では値 4 つで 1 つの要素を定義している点にご注意下さい。

表 6.74　境界条件を設定された位置と indices に返される値の関係

境界条件を設定された位置	indices に返される値
格子点 (node)	(格子点 1 の I), (格子点 1 の J) ..., (格子点 N の I), (格子点 N の J)
セル (cell)	(セル 1 の I), (セル 1 の J) ..., (セル N の I), (セル N の J)
辺 (edge)	(辺 1 の開始格子点の I), (辺 1 の開始格子点の J), (辺 1 の終了格子点の I), (辺 1 の終了格子点の J), ..., (辺 N の開始格子点の I), (辺 N の開始格子点の J), (辺 N の終了格子点の I), (辺 N の終了格子点の J)

6.5.45 cg_iric_read_bc_integer_f

整数型の境界条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_integer_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_Integer(type, num, name, &value);
```

形式 (Python)

```
value = cg_iRIC_Read_BC_Integer(type, num, name)
```

引数

表 6.75　cg_iric_read_bc_integer_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
value	integer	O	読み込まれた境界条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.46 cg_iric_read_bc_real_f

実数 (倍精度) 型の境界条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_real_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_Real(type, num, name, &value);
```

形式 (Python)

```
value = cg_iRIC_Read_BC_Real(type, num, name)
```

引数

表 6.76　cg_iric_read_bc_real_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
value	double precision	O	読み込まれた境界条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.47 cg_iric_read_bc_realsingle_f

実数 (単精度) 型の境界条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_realsingle_f(type, num, label, realvalue, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_RealSingle(type, num, label, &realvalue);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.77　cg_iric_read_bc_realsingle_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
realvalue	real	O	読み込まれた境界条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.48 cg_iric_read_bc_string_f

文字列型の境界条件の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_string_f(type, num, name, value, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_String(type, num, name, value);
```

形式 (Python)

```
value = cg_iRIC_Read_BC_String(type, num, name)
```

引数

表 6.78　cg_iric_read_bc_string_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
value	character(*)	O	読み込まれた境界条件の値
ier	integer	O	エラーコード。0 なら成功

6.5.49 cg_iric_read_bc_functionalsize_f

関数型の境界条件の変数のサイズを読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_functionalsize_f(type, num, name, size, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_FunctionalSize(type, num, name, &size);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.79　cg_iric_read_bc_functionalsize_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
size	integer	O	境界条件の配列の長さ
ier	integer	O	エラーコード。0 なら成功

6.5.50 cg_iric_read_bc_functional_f

倍精度関数型の境界条件の変数の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_functional_f(type, num, name, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_Functional(type, num, name, x, y);
```

形式 (Python)

```
x, y = cg_iRIC_Read_BC_Functional(type, num, name)
```

引数

表 6.80　cg_iric_read_bc_functional_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
x	double precision, dimension(:), allocatable	O	X の値の配列
y	double precision, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.51 cg_iric_read_bc_functional_realsingle_f

単精度関数型の境界条件の変数の値を読み込む。

形式 (FORTRAN)

```
call cg_iric_read_bc_functional_realsingle_f(type, num, name, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_Functional_RealSingle(type, num, name, x, y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.81　cg_iric_read_bc_functional_realsingle_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
x	real, dimension(:), allocatable	O	X の値の配列
y	real, dimension(:), allocatable	O	Y の値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.52 cg_iric_read_bc_functionalwithname_f

倍精度関数型の境界条件の変数の値を読み込む。変数が 1 つ、値が複数の関数型の境界条件の読み込みに利用する。

形式 (FORTRAN)

```
call cg_iric_read_bc_functionalwithname_f(type, num, name, paramname, data, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_BC_FunctionalWithName(type, num, name, paramname, data);
```

形式 (Python)

```
data = cg_iRIC_Read_BC_FunctionalWithName(type, num, name, paramname)
```

引数

表 6.82　cg_iric_read_bc_functionalwithname_f の引数

変数名	型	I/O	内容
type	character(*)	I	境界条件の識別名
num	integer	I	境界条件の番号
name	character(*)	I	境界条件の属性の名前
paramname	character(*)	I	値の名前
data	double precision, dimension(:), allocatable	O	値の配列
ier	integer	O	エラーコード。0 なら成功

6.5.53 cg_iric_read_geo_count_f

CGNS ファイルから地形データの数を読み込みます。

形式 (FORTRAN)

```
call cg_iric_read_geo_count_f(name, geocount, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Geo_Count(name, geocount);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.83　cg_iric_read_geo_count_f の引数

変数名	型	I/O	内容
name	character(*)	I	地理情報種類
geocount	integer	O	地理情報の数
ier	integer	O	エラーコード。0 なら成功

6.5.54 cg_iric_read_geo_filename_f

CGNS ファイルから地形データのファイル名と種類を読み込みます。

形式 (FORTRAN)

```
call cg_iric_read_geo_filename_f(name, geoid, geofilename, geotype, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Geo_Filename(name, geoid, geofilename, &geotype);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.84　cg_iric_read_geo_filename_f の引数

変数名	型	I/O	内容
name	character(*)	I	地理情報種類
geoid	integer	I	読み込む地形データの番号
geofilename	character(*)	O	ファイル名
geotype	integer	O	地形データの種類
ier	integer	O	エラーコード。0 なら成功

備考

なお、geotype で読み込まれる地形データの種類は、iriclib_f.h で定義された、 表 6.85 に示す値のいずれかです。

表 6.85　geotype で返される、地形データ種類を表す定数

geotype の定数名	値	内容
IRIC_GEO_POLYGON	1	ポリゴン
IRIC_GEO_RIVERSURVEY	2	河川測量データ

6.5.55 iric_geo_polygon_open_f

ポリゴンファイルを開く。

形式 (FORTRAN)

```
call iric_geo_polygon_open_f(filename, pid, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Open(filename, &pid);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.86　iric_geo_polygon_open_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
pid	integer	O	開いたポリゴンの ID
ier	integer	O	エラーコード。0 なら成功

6.5.56 iric_geo_polygon_read_integervalue_f

ポリゴンの値を整数で返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_integervalue_f(pid, intval, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_IntegerValue(pid, &intval);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.87　iric_geo_polygon_read_integervalue_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
intval	integer	O	ポリゴンの値
ier	integer	O	エラーコード。0 なら成功

6.5.57 iric_geo_polygon_read_realvalue_f

ポリゴンの値を実数で返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_realvalue_f(pid, realval, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_RealValue(pid, &realval);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.88　iric_geo_polygon_read_realvalue_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
realval	double precision	O	ポリゴンの値
ier	integer	O	エラーコード。0 なら成功

6.5.58 iric_geo_polygon_read_pointcount_f

ポリゴンの頂点の数を返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_pointcount_f(pid, count, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_PointCount(pid, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.89　iric_geo_polygon_read_pointcount_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
count	integer	O	ポリゴンの頂点の数
ier	integer	O	エラーコード。0 なら成功

6.5.59 iric_geo_polygon_read_points_f

ポリゴンの頂点の座標を返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_points_f(pid, x, y, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_Points(pid, x, y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.90　iric_geo_polygon_read_points_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
x	double precision , dimension(:), allocatable	O	ポリゴン頂点の X 座標
y	double precision , dimension(:), allocatable	O	ポリゴン頂点の Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.60 iric_geo_polygon_read_holecount_f

ポリゴンに開いた穴の数を返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_holecount_f(pid, holecount, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_HoleCount(pid, &holecount);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.91　iric_geo_polygon_read_holecount_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
holecount	integer	O	ポリゴンに開いた穴の数
ier	integer	O	エラーコード。0 なら成功

6.5.61 iric_geo_polygon_read_holepointcount_f

ポリゴンの穴の頂点の数を返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_holepointcount_f(pid, holeid, count, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_HolePointCount(pid, holeid, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.92　iric_geo_polygon_read_holepointcount_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
holeid	integer	I	穴の ID
count	integer	O	ポリゴンの頂点の数
ier	integer	O	エラーコード。0 なら成功

6.5.62 iric_geo_polygon_read_holepoints_f

ポリゴンの穴の頂点の座標を返す。

形式 (FORTRAN)

```
call iric_geo_polygon_read_holepoints_f(pid, holeid, x, y, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Read_HolePoints(pid, holeid, x, y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.93　iric_geo_polygon_read_holepoints_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
holeid	integer	I	穴の ID
x	double precision , dimension(:), allocatable	O	ポリゴン頂点の X 座標
y	double precision , dimension(:), allocatable	O	ポリゴン頂点の Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.63 iric_geo_polygon_close_f

ポリゴンファイルを閉じる。

形式 (FORTRAN)

```
call iric_geo_polygon_close_f(pid, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_Polygon_Close(pid);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.94　iric_geo_polygon_close_f の引数

変数名	型	I/O	内容
pid	integer	I	ポリゴンの ID
ier	integer	O	エラーコード。0 なら成功

6.5.64 iric_geo_riversurvey_open_f

河川測量データを開く。

形式 (FORTRAN)

```
call iric_geo_riversurvey_open_f(filename, rid, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Open(filename, rid);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.95　iric_geo_riversurvey_open_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
rid	integer	O	河川測量データの ID
ier	integer	O	エラーコード。0 なら成功

6.5.65 iric_geo_riversurvey_read_count_f

河川横断線の数を返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_count_f(rid, count, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_Count(rid, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.96　iric_geo_riversurvey_read_count_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
count	integer	O	河川横断線の数
ier	integer	O	エラーコード。0 なら成功

6.5.66 iric_geo_riversurvey_read_position_f

横断線の中心点の座標を返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_position_f(rid, pointid, x, y, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_Position(rid, pointid, &x, &y);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.97　iric_geo_riversurvey_read_position_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
x	double precision	O	中心点の X 座標
y	double precision	O	中心点の Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.67 iric_geo_riversurvey_read_direction_f

横断線の向きを返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_direction_f(rid, pointid, vx, vy, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_Direction(rid, pointid, &vx, &vy);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.98　iric_geo_riversurvey_read_direction_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
vx	double precision	O	向きの X 座標
vx	double precision	O	向きの Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.68 iric_geo_riversurvey_read_name_f

横断線の名前を文字列として返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_name_f(rid, pointid, name, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_Name(rid, pointid, name);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.99　iric_geo_riversurvey_read_name_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
name	character(*)	O	横断線の名前
ier	integer	O	エラーコード。0 なら成功

6.5.69 iric_geo_riversurvey_read_realname_f

横断線の名前を実数値として返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_realname_f(rid, pointid, realname, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_RealName(rid, pointid, &realname);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.100　iric_geo_riversurvey_read_realname_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
realname	double precision	O	横断線の名前
ier	integer	O	エラーコード。0 なら成功

6.5.70 iric_geo_riversurvey_read_leftshift_f

横断線の標高データのシフト量を返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_leftshift_f(rid, pointid, shift, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_LeftShift(rid, pointid, &shift);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.101　iric_geo_riversurvey_read_leftshift_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
shift	double precision	O	シフト量
ier	integer	O	エラーコード。0 なら成功

6.5.71 iric_geo_riversurvey_read_altitudecount_f

横断線の標高データの数を返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_altitudecount_f(rid, pointid, count, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_AltitudeCount(rid, pointid, &count);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.102　iric_geo_riversurvey_read_altitudecount_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
count	integer	O	横断線の標高データの数
ier	integer	O	エラーコード。0 なら成功

6.5.72 iric_geo_riversurvey_read_altitudes_f

横断線の中心点の座標を返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_altitudes_f(rid, pointid, position, height, active, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_Altitudes(rid, pointid, position, height, active);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.103　iric_geo_riversurvey_read_altitudes_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	pointid	I	横断線の ID
position	double precision, dimension(:), allocatable	O	標高データの位置 (0 より小さい = 左岸側, 0 より大きい = 右岸側)
height	double precision, dimension(:), allocatable	O	標高データの高さ
active	integer, dimension(:), allocatable	O	標高データの有効/無効 (1: 有効, 0: 無効)
ier	integer	O	エラーコード。0 なら成功

6.5.73 iric_geo_riversurvey_read_fixedpointl_f

横断線の左岸延長線のデータを返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_fixedpointl_f(rid, pointid, set, directionx, directiony,
↪ index, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_FixedPointL(rid, pointid, &set, &directionx, &
↪directiony, &index);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.104　iric_geo_riversurvey_read_fixedpointl_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
set	integer	O	左岸延長線が登録されていたら 1
directionx	double precision	O	左岸延長線の向きの X 成分
directiony	double precision	O	左岸延長線の向きの Y 成分
index	integer	O	左岸延長線の開始位置の標高データの番号
ier	integer	O	エラーコード。0 なら成功

6.5.74 iric_geo_riversurvey_read_fixedpointr_f

横断線の右岸延長線のデータを返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_fixedpointr_f(rid, pointid, set, directionx, directiony,
↪ index, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_FixedPointR(rid, pointid, &set, &directionx, &
↪directiony, &index);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.105　iric_geo_riversurvey_read_fixedpointr_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
set	integer	O	右岸延長線が登録されていたら 1
directionx	double precision	O	右岸延長線の向きの X 成分
directiony	double precision	O	右岸延長線の向きの Y 成分
index	integer	O	右岸延長線の開始位置の標高データの番号
ier	integer	O	エラーコード。0 なら成功

6.5.75 iric_geo_riversurvey_read_watersurfaceelevation_f

横断線での水面標高のデータを返す。

形式 (FORTRAN)

```
call iric_geo_riversurvey_read_watersurfaceelevation_f(rid, pointid, set, value, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Read_WaterSurfaceElevation(rid, pointid, set, &value);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.106　iric_geo_riversurvey_read_watersurfaceelevation_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
pointid	integer	I	横断線の ID
set	integer	O	水面標高が登録されていたら 1
value	double precision	O	水面標高
ier	integer	O	エラーコード。0 なら成功

6.5.76 iric_geo_riversurvey_close_f

河川測量データファイルを閉じる。

形式 (FORTRAN)

```
call iric_geo_riversurvey_close_f(pid, ier)
```

形式 (C/C++)

```
ier = iRIC_Geo_RiverSurvey_Close(pid);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.107　iric_geo_riversurvey_close_f の引数

変数名	型	I/O	内容
rid	integer	I	河川測量データの ID
ier	integer	O	エラーコード。0 なら成功

6.5.77 cg_iric_writegridcoord1d_f

1 次元構造格子を出力する。

形式 (FORTRAN)

```
call cg_iric_writegridcoord1d_f(nx, x, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_WriteGridCoord1d(nx, x);
```

形式 (Python)

```
cg_iRIC_WriteGridCoord1d(nx, x)
```

引数

表 6.108　cg_iric_writegridcoord1d_f の引数

変数名	型	I/O	内容
nx	integer	I	i 方向格子点数
x	double precision, dimension(:), allocatable	I	格子点の x 座標値
ier	integer	O	エラーコード。0 なら成功

6.5.78 cg_iric_writegridcoord2d_f

2 次元構造格子を出力する。

形式 (FORTRAN)

```
call cg_iric_writegridcoord2d_f(nx, ny, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_WriteGridCoord2d(nx, ny, x, y);
```

形式 (Python)

```
cg_iRIC_WriteGridCoord2d(nx, ny, x, y)
```

引数

表 6.109　cg_iric_writegridcoord2d_f の引数

変数名	型	I/O	内容
nx	integer	I	i 方向格子点数
ny	integer	I	j 方向格子点数
x	double precision, dimension(:,:), allocatable	I	格子点の x 座標値
y	double precision, dimension(:,:), allocatable	I	格子点の y 座標値
ier	integer	O	エラーコード。0 なら成功

6.5.79 cg_iric_writegridcoord3d_f

3 次元構造格子を出力する。

形式 (FORTRAN)

```
call cg_iric_writegridcoord3d_f(nx, ny, nz, x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_WriteGridCoord3d(nx, ny, nz, x, y, z);
```

形式 (Python)

```
cg_iRIC_WriteGridCoord3d(nx, ny, nz, x, y, z)
```

引数

表 6.110　cg_iric_writegridcoord3d_f の引数

変数名	型	I/O	内容
nx	integer	I	i 方向格子点数
ny	integer	I	j 方向格子点数
nz	integer	I	k 方向格子点数
x	double precision, dimension(:), allocatable	I	格子点の x 座標値
y	double precision, dimension(:), allocatable	I	格子点の y 座標値
z	double precision, dimension(:), allocatable	I	格子点の z 座標値
ier	integer	O	エラーコード。0 なら成功

6.5.80 cg_iric_write_grid_integer_node_f

構造格子の格子点で定義された整数の属性を出力する。

形式 (FORTRAN)

```
call cg_iric_write_grid_integer_node_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Grid_Integer_Node(label, values);
```

形式 (Python)

```
cg_iRIC_Write_Grid_Integer_Node(label, values)
```

引数

表 6.111　cg_iric_write_grid_integer_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), llocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.81 cg_iric_write_grid_real_node_f

構造格子の格子点で定義された倍精度実数の属性を出力する。

形式 (FORTRAN)

```
call cg_iric_write_grid_real_node_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Grid_Real_Node(label, values);
```

形式 (Python)

```
cg_iRIC_Write_Grid_Real_Node(label, values)
```

引数

表 6.112　cg_iric_write_grid_real_node_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	double precision, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.82 cg_iric_write_grid_integer_cell_f

構造格子のセルで定義された整数の属性を出力する。

形式 (FORTRAN)

```
call cg_iric_write_grid_integer_cell_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Grid_Integer_Cell(label, values);
```

形式 (Python)

```
cg_iRIC_Write_Grid_Integer_Cell(label, values)
```

引数

表 6.113　cg_iric_write_grid_integer_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	integer, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.83 cg_iric_write_grid_real_cell_f

構造格子のセルで定義された倍精度実数の属性を出力する。

形式 (FORTRAN)

```
call cg_iric_write_grid_real_cell_f(label, values, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Grid_Real_Cell(label, values);
```

形式 (Python)

```
cg_iRIC_Write_Grid_Real_Cell(label, values)
```

引数

表 6.114　cg_iric_write_grid_real_cell_f の引数

変数名	型	I/O	内容
label	character(*)	I	属性名
values	double precision, dimension(:), allocatable	O	属性値
ier	integer	O	エラーコード。0 なら成功

6.5.84 cg_iric_write_sol_time_f

時刻を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_time_f(time, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Time(time);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Time(time)
```

引数

表 6.115　cg_iric_write_sol_time_f の引数

変数名	型	I/O	内容
time	double precision	I	時刻
ier	integer	O	エラーコード。0 なら成功

6.5.85 cg_iric_write_sol_iteration_f

ループ回数を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_iteration_f(iteration, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Iteration(iteration);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Iteration(iteration)
```

引数

表 6.116　cg_iric_write_sol_iteration_f の引数

変数名	型	I/O	内容
iteration	integer	I	ループ回数
ier	integer	O	エラーコード。0 なら成功

6.5.86 cg_iric_write_sol_gridcoord2d_f

2 次元構造格子を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_gridcoord2d_f(x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_GridCoord2d(x, y);
```

形式 (Python)

```
cg_iRIC_Write_Sol_GridCoord2d(x, y)
```

引数

表 6.117　cg_iric_write_sol_gridcoord2d_f の引数

変数名	型	I/O	内容
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.87 cg_iric_write_sol_gridcoord3d_f

3 次元構造格子を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_gridcoord3d_f(x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_GridCoord3d(x, y, z);
```

形式 (Python)

```
cg_iRIC_Write_Sol_GridCoord3d(x, y, z)
```

引数

表 6.118　cg_iric_write_sol_gridcoord3d_f の引数

変数名	型	I/O	内容
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
z	double precision, dimension(:), allocatable	I	Z 座標
ier	integer	O	エラーコード。0 なら成功

6.5.88 cg_iric_write_sol_baseiterative_integer_f

整数の計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_baseiterative_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_BaseIterative_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_BaseIterative_Integer(label, val)
```

引数

表 6.119　cg_iric_write_sol_baseiterative_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.89 cg_iric_write_sol_baseiterative_real_f

倍精度実数の計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_baseiterative_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_BaseIterative_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_BaseIterative_Real(label, val)
```

引数

表 6.120　cg_iric_write_sol_baseiterative_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.90 cg_iric_write_sol_baseiterative_string_f

文字列の計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_baseiterative_string_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_BaseIterative_String(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_BaseIterative_String(label, val)
```

引数

表 6.121　cg_iric_write_sol_baseiterative_string_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	character(*)	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.91 cg_iric_write_sol_integer_f

整数の格子点ごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Integer(label, val)
```

引数

表 6.122　cg_iric_write_sol_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer, dimension(:,:), allocatable	I	出力する値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.92 cg_iric_write_sol_real_f

倍精度実数の格子点ごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Real(label, val)
```

引数

表 6.123　cg_iric_write_sol_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision, dimension(:,:), allocatable	I	出力する値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.93 cg_iric_write_sol_cell_integer_f

整数の格子セルごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_cell_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Cell_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Cell_Integer(label, val)
```

引数

表 6.124　cg_iric_write_sol_cell_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer, dimension(:,:), allocatable	I	出力する値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.94 cg_iric_write_sol_cell_real_f

倍精度実数の格子セルごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_cell_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Cell_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Cell_Real(label, val)
```

引数

表 6.125　cg_iric_write_sol_cell_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision, dimension(:,:), allocatable	I	出力する値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.95 cg_iric_write_sol_iface_integer_f

整数の I 方向格子エッジごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_iface_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_IFace_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_IFace_Integer(label, val)
```

引数

表 6.126　cg_iric_write_sol_iface_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer, dimension(:,:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.96 cg_iric_write_sol_iface_real_f

倍精度実数の I 方向格子エッジごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_iface_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_IFace_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_IFace_Real(label, val)
```

引数

表 6.127　cg_iric_write_sol_iface_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision, dimension(:,:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.97 cg_iric_write_sol_jface_integer_f

整数の J 方向格子エッジごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_jface_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_JFace_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_JFace_Integer(label, val)
```

引数

表 6.128　cg_iric_write_sol_jface_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer, dimension(:,:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.98 cg_iric_write_sol_jface_real_f

倍精度実数の J 方向格子エッジごとに値を持つ計算結果を出力する。

形式 (FORTRAN)

```
call cg_iric_write_sol_jface_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_JFace_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_JFace_Real(label, val)
```

引数

表 6.129　cg_iric_write_sol_jface_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision, dimension(:,:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.99 cg_iric_write_sol_particle_pos2d_f

粒子の位置を出力する。 (2 次元)

形式 (FORTRAN)

```
call cg_iric_write_sol_particle_pos2d_f(count, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Particle_Pos2d(count, x, y);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Particle_Pos2d(x, y)
```

引数

表 6.130　cg_iric_write_sol_particle_pos2d_f の引数

変数名	型	I/O	内容
count	integer	I	粒子の数
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.100 cg_iric_write_sol_particle_pos3d_f

粒子の位置を出力する。 (3 次元)

形式 (FORTRAN)

```
call cg_iric_write_sol_particle_pos3d_f(count, x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Particle_Pos3d(count, x, y, z);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Particle_Pos3d(x, y, z)
```

引数

表 6.131　cg_iric_write_sol_particle_pos3d_f の引数

変数名	型	I/O	内容
count	integer	I	粒子の数
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
z	double precision, dimension(:), allocatable	I	Z 座標
ier	integer	O	エラーコード。0 なら成功

6.5.101 cg_iric_write_sol_particle_integer_f

整数の粒子ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_particle_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Particle_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Particle_Integer(label, val)
```

引数

表 6.132　cg_iric_write_sol_particle_integer_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	integer, dimension(:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.102 cg_iric_write_sol_particle_real_f

倍精度実数の粒子ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_particle_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_Particle_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_Particle_Real(label, val)
```

引数

表 6.133　cg_iric_write_sol_particle_real_f の引数

変数名	型	I/O	内容
label	character(*)	I	出力する値の名前
val	double precision, dimension(:), allocatable	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.103 cg_iric_write_sol_particlegroup_groupbegin_f

粒子で定義された計算結果の出力を開始する

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_groupbegin_f(name, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_GroupBegin(name);
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_GroupBegin(name)
```

引数

表 6.134　cg_iric_write_sol_particlegroup_groupbegin_f の引数

変数名	型	I/O	内容
name	character*	I	出力するグループの名前
ier	integer	O	エラーコード。0 なら成功

6.5.104 cg_iric_write_sol_particlegroup_groupend_f

粒子で定義された計算結果の出力を終了する

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_groupend_f(ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_GroupEnd();
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_GroupEnd()
```

引数

表 6.135　cg_iric_write_sol_particlegroup_groupend_f の引数

変数名	型	I/O	内容
ier	integer	O	エラーコード。0 なら成功

6.5.105 cg_iric_write_sol_particlegroup_pos2d_f

粒子の位置を出力する (2 次元)

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_pos2d_f(x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_Pos2d(x, y);
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_Pos2d(x, y)
```

引数

表 6.136　cg_iric_write_sol_particlegroup_pos2d_f の引数

変数名	型	I/O	内容
x	double precision	I	X 座標
y	double precision	I	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.106 cg_iric_write_sol_particlegroup_pos3d_f

粒子の位置を出力する (3 次元)

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_pos3d_f(x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_Pos3d(x, y, z);
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_Pos3d(x, y, z)
```

引数

表 6.137　cg_iric_write_sol_particlegroup_pos3d_f の引数

変数名	型	I/O	内容
x	double precision	I	X 座標
y	double precision	I	Y 座標
z	double precision	I	Z 座標
ier	integer	O	エラーコード。0 なら成功

6.5.107 cg_iric_write_sol_particlegroup_integer_f

整数の粒子ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_Integer(label, val)
```

引数

表 6.138　cg_iric_write_sol_particlegroup_integer_f の引数

変数名	型	I/O	内容
label	character*	I	出力する値の名前
val	integer	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.108 cg_iric_write_sol_particlegroup_real_f

倍精度実数の粒子ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_particlegroup_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_ParticleGroup_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_ParticleGroup_Real(label, val)
```

引数

表 6.139　cg_iric_write_sol_particlegroup_real_f の引数

変数名	型	I/O	内容
label	character*	I	出力する値の名前
val	double precision	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.109 cg_iric_write_sol_polydata_groupbegin_f

ポリゴンもしくは折れ線で定義された計算結果の出力を開始する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_groupbegin_f(name, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_GroupBegin(name);
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_GroupBegin(name)
```

引数

表 6.140　cg_iric_write_sol_polydata_groupbegin_f の引数

変数名	型	I/O	内容
name	character*	I	出力するグループの名前
ier	integer	O	エラーコード。0 なら成功

6.5.110 cg_iric_write_sol_polydata_groupend_f

ポリゴンもしくは折れ線で定義された計算結果の出力を終了する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_groupend_f(ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_GroupEnd();
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_GroupEnd()
```

引数

表 6.141　cg_iric_write_sol_polydata_groupend_f の引数

変数名	型	I/O	内容
ier	integer	O	エラーコード。0 なら成功

6.5.111 cg_iric_write_sol_polydata_polygon_f

計算結果としてポリゴンの形状を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_polygon_f(numpoints, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_Polygon(numpoints, x, y);
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_Polygon(x, y)
```

引数

表 6.142　cg_iric_write_sol_polydata_polygon_f の引数

変数名	型	I/O	内容
numpoints	integer	I	ポリゴンを構成する点の数
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.112 cg_iric_write_sol_polydata_polyline_f

計算結果として折れ線の形状を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_polyline_f(numpoints, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_Polyline(numpoints, x, y);
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_Polyline(x, y)
```

引数

表 6.143　cg_iric_write_sol_polydata_polyline_f の引数

変数名	型	I/O	内容
numpoints	integer	I	折れ線を構成する点の数
x	double precision, dimension(:), allocatable	I	X 座標
y	double precision, dimension(:), allocatable	I	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.113 cg_iric_write_sol_polydata_integer_f

整数のポリゴンもしくは折れ線ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_integer_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_Integer(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_Integer(label, val)
```

引数

表 6.144　cg_iric_write_sol_polydata_integer_f の引数

変数名	型	I/O	内容
label	character*	I	出力する値の名前
val	integer	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.114 cg_iric_write_sol_polydata_real_f

倍精度実数のポリゴンもしくは折れ線ごとに値を持つ計算結果を出力する

形式 (FORTRAN)

```
call cg_iric_write_sol_polydata_real_f(label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_Sol_PolyData_Real(label, val);
```

形式 (Python)

```
cg_iRIC_Write_Sol_PolyData_Real(label, val)
```

引数

表 6.145　cg_iric_write_sol_polydata_real_f の引数

変数名	型	I/O	内容
label	character*	I	出力する値の名前
val	double precision	I	出力する値
ier	integer	O	エラーコード。0 なら成功

6.5.115 iric_check_cancel_f

ユーザがソルバーをキャンセルしたか確認する。

形式 (FORTRAN)

```
call iric_check_cancel_f(canceled)
```

形式 (C/C++)

```
iRIC_Check_Cancel(&canceled);
```

形式 (Python)

```
canceled = iRIC_Check_Cancel()
```

引数

表 6.146　iric_check_cancel_f の引数

変数名	型	I/O	内容
canceled	integer	O	キャンセルされていたら 1

6.5.116 iric_check_lock_f

CGNS ファイルが GUI によってロックされているか確認する。

注釈: この関数は、現在は何もせず、呼び出しは不要。

形式 (FORTRAN)

```
call iric_check_lock_f(filename, locked)
```

形式 (C/C++)

```
iRIC_Check_Lock(filename, locked);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.147　iric_check_lock_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
locked	integer	O	ロックされていたら 1

6.5.117 iric_write_sol_start_f

計算結果の出力開始を GUI に通知する。

注釈: この関数は、現在は何もせず、呼び出しは不要。

形式 (FORTRAN)

```
call iric_write_sol_start_f(filename, ier)
```

形式 (C/C++)

```
ier = iRIC_Write_Sol_Start(filename);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.148　iric_write_sol_start_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
ier	integer	O	エラーコード。0 なら成功

6.5.118 iric_write_sol_end_f

計算結果の出力終了を GUI に通知する。

注釈: この関数は、現在は何もせず、呼び出しは不要。

形式 (FORTRAN)

```
call iric_write_sol_end_f(filename, ier)
```

形式 (C/C++)

```
ier = iRIC_Write_Sol_End(filename);
```

形式 (Python)

Python にはこの関数は存在しない

引数

表 6.149　iric_write_sol_end_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
ier	integer	O	エラーコード。0 なら成功

6.5.119 cg_iric_flush_f

計算結果の出力をファイルに書き込む。

形式 (FORTRAN)

```
call cg_iric_flush_f(filename, fin, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Flush(filename, fin);
```

形式 (Python)

```
fin = cg_iRIC_Flush(filename, fin)
```

引数

表 6.150　cg_iric_flush_f の引数

変数名	型	I/O	内容
filename	character(*)	I	ファイル名
fid	integer	I/O	ファイル ID
ier	integer	O	エラーコード。0 なら成功

6.5.120 cg_iric_read_sol_count_f

計算結果の数を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_count_f(count, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Count(&count);
```

形式 (Python)

```
count = cg_iRIC_Read_Sol_Count()
```

引数

表 6.151　cg_iric_read_sol_count_f の引数

変数名	型	I/O	内容
count	integer	O	計算結果の数
ier	integer	O	エラーコード。0 なら成功

6.5.121 cg_iric_read_sol_time_f

計算結果の時刻の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_time_f(step, time, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Time(step, &time);
```

形式 (Python)

```
time = cg_iRIC_Read_Sol_Time(step)
```

引数

表 6.152　cg_iric_read_sol_time_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
time	double precision	O	時刻
ier	integer	O	エラーコード。0 なら成功

6.5.122 cg_iric_read_sol_iteration_f

計算結果のループ回数の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_iteration_f(step, iteration, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Iteration(step, &iteration);
```

形式 (Python)

```
iteration = cg_iRIC_Read_Sol_Iteration(step)
```

引数

表 6.153　cg_iric_read_sol_iteration_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
iteration	integer	O	ループ回数
ier	integer	O	エラーコード。0 なら成功

6.5.123 cg_iric_read_sol_baseiterative_integer_f

整数の計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_baseiterative_integer_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_BaseIterative_Integer(step, label, &val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_BaseIterative_Integer(step, label)
```

引数

表 6.154　cg_iric_read_sol_baseiterative_integer_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	integer	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.124 cg_iric_read_sol_baseiterative_real_f

倍精度実数の計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_baseiterative_real_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_BaseIterative_Real(step, label, &val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_BaseIterative_Real(step, label)
```

引数

表 6.155　cg_iric_read_sol_baseiterative_real_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	double precision	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.125 cg_iric_read_sol_baseiterative_string_f

文字列の計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_baseiterative_string_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_BaseIterative_String(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_BaseIterative_String(step, label)
```

引数

表 6.156　cg_iric_read_sol_baseiterative_string_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	character(*)	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.126 cg_iric_read_sol_gridcoord2d_f

計算結果の 2 次元構造格子を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_gridcoord2d_f(step, x, y, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_GridCoord2d(step, x, y);
```

形式 (Python)

```
x, y = cg_iRIC_Read_Sol_GridCoord2d(step)
```

引数

表 6.157　cg_iric_read_sol_gridcoord2d_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
x	double precision, dimension(:), allocatable	O	X 座標
y	double precision, dimension(:), allocatable	O	Y 座標
ier	integer	O	エラーコード。0 なら成功

6.5.127 cg_iric_read_sol_gridcoord3d_f

計算結果の 3 次元構造格子を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_gridcoord3d_f(step, x, y, z, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_GridCoord3d(step, x, y, z);
```

形式 (Python)

```
x, y, z = cg_iRIC_Read_Sol_GridCoord3d(step)
```

引数

表 6.158　cg_iric_read_sol_gridcoord3d_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
x	double precision, dimension(:), allocatable	O	X 座標
y	double precision, dimension(:), allocatable	O	Y 座標
z	double precision, dimension(:), allocatable	O	Z 座標
ier	integer	O	エラーコード。0 なら成功

6.5.128 cg_iric_read_sol_integer_f

整数の格子点ごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_integer_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Integer(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_Integer(step, label)
```

引数

表 6.159　cg_iric_read_sol_integer_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	integer, dimension(:,:), allocatable	O	値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.129 cg_iric_read_sol_real_f

倍精度実数の格子点ごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_real_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Real(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_Real(step, label)
```

引数

表 6.160　cg_iric_read_sol_real_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	double precision, dimension(:,:), allocatable	O	値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.130 cg_iric_read_sol_cell_integer_f

整数の格子セルごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_cell_integer_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Cell_Integer(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_Cell_Integer(step, label)
```

引数

表 6.161　cg_iric_read_sol_cell_integer_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	integer, dimension(:,:), allocatable	O	値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.131 cg_iric_read_sol_cell_real_f

倍精度実数の格子セルごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_cell_real_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_Cell_Real(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_Cell_Real(step, label)
```

引数

表 6.162　cg_iric_read_sol_cell_real_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	double precision, dimension(:,:), allocatable	O	値 (3 次元格子の場合は dimension(:,:,:))
ier	integer	O	エラーコード。0 なら成功

6.5.132 cg_iric_read_sol_iface_integer_f

整数の J 方向格子エッジごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_iface_integer_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_IFace_Integer(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_IFace_Integer(step, label)
```

引数

表 6.163　cg_iric_read_sol_iface_integer_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	integer, dimension(:,:), allocatable	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.133 cg_iric_read_sol_iface_real_f

倍精度実数の I 方向格子エッジごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_iface_real_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_IFace_Real(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_IFace_Real(step, label)
```

引数

表 6.164　cg_iric_read_sol_iface_real_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	double precision, dimension(:,:), allocatable	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.134 cg_iric_read_sol_jface_integer_f

整数の J 方向格子エッジごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_jface_integer_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_JFace_Integer(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_JFace_Integer(step, label)
```

引数

表 6.165　cg_iric_read_sol_jface_integer_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	integer, dimension(:,:), allocatable	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.135 cg_iric_read_sol_jface_real_f

倍精度実数の J 方向格子エッジごとに値を持つ計算結果の値を取得する。

形式 (FORTRAN)

```
call cg_iric_read_sol_jface_real_f(step, label, val, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Read_Sol_JFace_Real(step, label, val);
```

形式 (Python)

```
val = cg_iRIC_Read_Sol_JFace_Real(step, label)
```

引数

表 6.166　cg_iric_read_sol_jface_real_f の引数

変数名	型	I/O	内容
step	integer	I	ステップ数
label	character(*)	I	名前
val	double precision, dimension(:,:), allocatable	O	値
ier	integer	O	エラーコード。0 なら成功

6.5.136 cg_iric_write_errorcode_f

エラーコードを出力する。

形式 (FORTRAN)

```
call cg_iric_write_errorcode_f(code, ier)
```

形式 (C/C++)

```
ier = cg_iRIC_Write_ErrorCode(code);
```

形式 (Python)

```
cg_iRIC_Write_ErrorCode(code)
```

引数

表 6.167　cg_iric_write_errorcode_f の引数

変数名	型	I/O	内容
code	integer	I	格子生成プログラムが返すエラーコード
ier	integer	O	エラーコード。0 なら成功

6.5.137 cg_close_f

CGNS ファイルを閉じる。

形式 (FORTRAN)

```
call cg_close_f(fid, ier)
```

形式 (C/C++)

```
ier = cg_close(fid);
```

形式 (Python)

```
cg_close(fid)
```

引数

表 6.168　cg_close_f の引数

変数名	型	I/O	内容
fid	integer	I	ファイル ID
ier	integer	O	エラーコード。0 なら成功

第 7 章

その他の情報

7.1 Fortran プログラムでの引数の読み込み処理

iRIC は、ソルバーや格子生成プログラムを起動する時、コマンドライン引数として計算データファイルもしくは格子生成データファイルの名前を渡すため、これを読み込む必要があります。

Fortran では、コマンドライン引数を読み込む方法がコンパイラによって異なります。ここでは、Intel Fortran Compiler と、GNU Fortran (gfortran), G95 での引数の読み込み処理について説明します。

7.1.1 Intel Fortran Compiler

nargs() でコマンドライン引数の個数を取得し、引数がある場合、getarg() で引数を取得します。

リスト 7.1　Intel Fortran Compiler での引数読み込み処理例

```
icount = nargs()  ! コマンド名が数に含まれるので、引数が 1 つなら 2 を返す
if ( icount.eq.2 ) then
  call getarg(1, condFile, istatus)
else
  write(*,*) "Input File not specified."
  stop
endif
```

7.1.2 GNU Fortran, G95

iargc() でコマンドライン引数の個数を取得し、引数がある場合、getarg() で引数を取得します。

Intel Fortran Compiler の nargs(), getarg() とは仕様が異なりますので注意して下さい。

リスト 7.2　GNU Fortran, G95 での引数読み込み処理例

```
icount = iargc()  ! コマンド名は数に含まれないので、引数が 1 つなら 1 を返す
if ( icount.eq.1 ) then
  call getarg(0, str1)       ! 実行プログラムのファイル名
  call getarg(1, condfile)   ! 引数
else
  write(*,*) "Input File not specified."
  stop
endif
```

7.2 Fortran 言語で iriclib, cgnslib とリンクしてビルドする方法

iRIC と連携して動作するソルバー、格子生成プログラムをコンパイルするには、cgnslib, iriclib とリンクする必要があります。それぞれ、Intel Fortran Compiler と GNU Fortran では異なるライブラリを利用する必要があります。それぞれで必要なライブラリのファイル名は表 7.1 のとおりです。ヘッダファイルは共通で、 "libcgns_f.h"、"iriclib_f.h" です。

表 7.1　コンパイラ別の、iRIClib, cgnslib 関連のファイル名

コンパイラ	iRIClib ライブラリ	cgnslib ライブラリ
Intel Fortran Compiler	iriclib_x64_ifort.lib	cgnsdll_x64_ifort.lib
GNU Fortran(gfortran)	iriclib.lib	cgnsdll.lib

ソースコードのファイルが solver.f の時のコンパイル手順について以下に示します。ただし、コンパイラの設定 (path の設定など) は完了しているものとします。

7.2.1 Intel Fortran Compiler (Windows)

solver.f, cgnsdll_x64_ifort.lib, iriclib_x64_ifort.lib, cgnslib_f.h, iriclib_f.h を同じフォルダに置き、そこに移動して以下のコマンドを実行することで、実行ファイル solver.exe が生成されます。

```
ifort solver.f cgnsdll_x64_ifort.lib iriclib_x64_ifort.lib /MD
```

コンパイル時には、同時に solver.exe.manifest というファイルも作成されます。ソルバーをコピーする時はこのファイルも一緒にコピーし、同じフォルダに配置してください。

7.2.2 GNU Fortran

solver.f, cgnsdll.lib, iriclib.lib, cgnslib_f.h, iriclib_f.h を同じフォルダに置き、そこに移動して以下のコマンドを実行することで、実行ファイル solver.exe が生成されます。

```
gfortran -c solver.f
g++ -o solver.exe -lgfortran solver.o cgnsdll.lib iriclib.lib
```

7.3 特別な格子属性、計算結果の名前について

iRIC では、特別な目的で用いる格子属性、計算結果について、特別な名前を用います。開発するソルバーで、以下の目的に合致する属性を入出力する場合、ここで示す名前を使ってください。

7.3.1 格子属性

入力格子の属性について定義された特別な名前を 表 7.2 に示します。

表 7.2　格子属性について定義された特別な名前

名前	説明
Elevation	格子点の標高 (単位: m) を保持する格子属性です。格子点の、実数の属性として定義します。

ソルバーで Elevation を使用する場合は、 GridRelatedCondition 要素の子要素の、Item 要素の name 属性に指定します。caption 属性は任意に設定できます。定義例を リスト 7.3 に示します。

リスト 7.3　Elevation 要素の定義例

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

一方格子生成プログラムで標高情報を出力する場合、 Elevation という名前を使って出力すれば iRIC で読み込まれます。格子生成プログラムで Elavtion を出力する処理の例をリスト 7.4 に示します。

リスト 7.4　格子生成プログラムでの、Elevation を出力するソースコードの例

```
cg_iric_write_grid_real_node_f("Elevation", elevation, ier);
```

7.3.2 計算結果

計算結果について定義された特別な名前を表 7.3 に示します。ここで示す名前は、 iRIClib の関数の引数に指定してください。

これらの特別な計算結果を全て出力するソルバーの例をリスト 7.5 に示します。

表 7.3　計算結果について定義された特別な名前

名前	説明	必須
Elevation	河床の標高 (単位: m)。実数の計算結果として出力します。"Elevation(m)" などのように、後ろに単位などの文字列を付加してもかまいません。	○
WaterSurfaceElevation	水面の標高 (単位: m)。実数の計算結果として出力します。"WaterSurfaceElevation(m)" などのように、後ろに単位などの文字列を付加してもかまいません。	
IBC	計算結果の有効・無効フラグ。無効な (水がない) 領域では 0、有効な (水がある) 領域では 1 を出力します。	

リスト 7.5　特別な名前の計算結果を出力するソースコードの例

```
call cg_iric_write_sol_real_f('Elevation(m)', elevation_values, ier)
call cg_iric_write_sol_real_f('WaterSurfaceElevation(m)', surface_values, ier)
call cg_iric_write_sol_integer_f('IBC', IBC_values, ier)
```

7.4 CGNS ファイル、 CGNS ライブラリに関する情報

7.4.1 CGNS ファイルフォーマットの概要

CGNS は、CFG General Notation System の略で、数値流体力学で用いられるデータを格納するための汎用ファイルフォーマットです。OS や CPU の種類が異なるコンピュータの間で、共通して利用することができます。数値流体力学で用いられる標準的なデータ形式が定義されているほか、ソルバーごとに独自の要素を追加する拡張性が備わっています。

CGNS ファイルの入出力ライブラリは cgnslib として提供されており、以下の言語から利用することができます。

- C, C++
- FORTRAN
- Python

元は Boeing 社と NASA が共同で開発しましたが、現在はオープンソースのコミュニティによって機能追加やメンテナンスが行われています。

7.4.2 CGNS ファイルの閲覧方法

ここでは、iRIC により作成した CGNS ファイルを HDFView を用いて閲覧する方法を説明します。HDFView は、フリーソフトとして公開されているソフトウェアです。

HDFView のインストール

まず、HDFView をインストールします。HDFView のインストーラは、以下のサイトからダウンロードできます。

https://www.hdfgroup.org/downloads/index.html

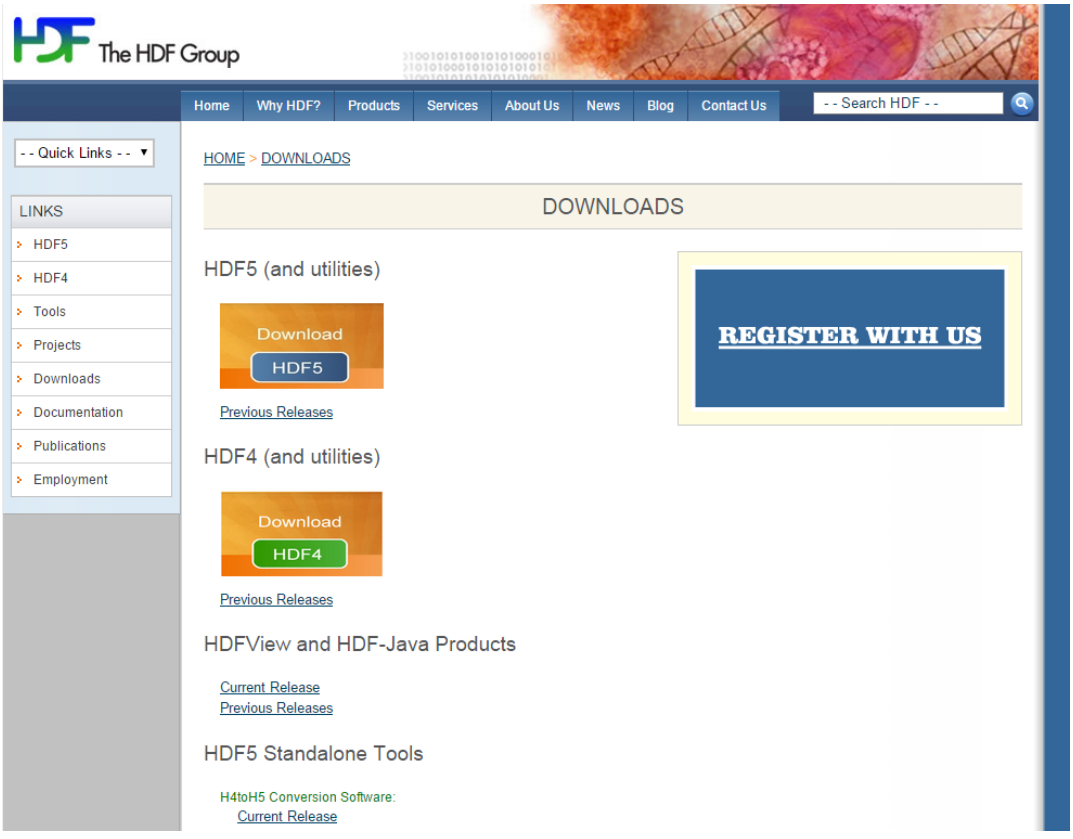

図 7.1　HDF group ウェブページ

HDF のホームページから、「Current Release」のリンクをクリックします。すると、様々なファイルのダウンロード画面に移動します。ここで、適切な (64 bit or 32 bit) プラットフォームのインストーラをクリックしてダウンロードしてください。解凍してインストーラを実行することで、 HDFView がインストールされます。

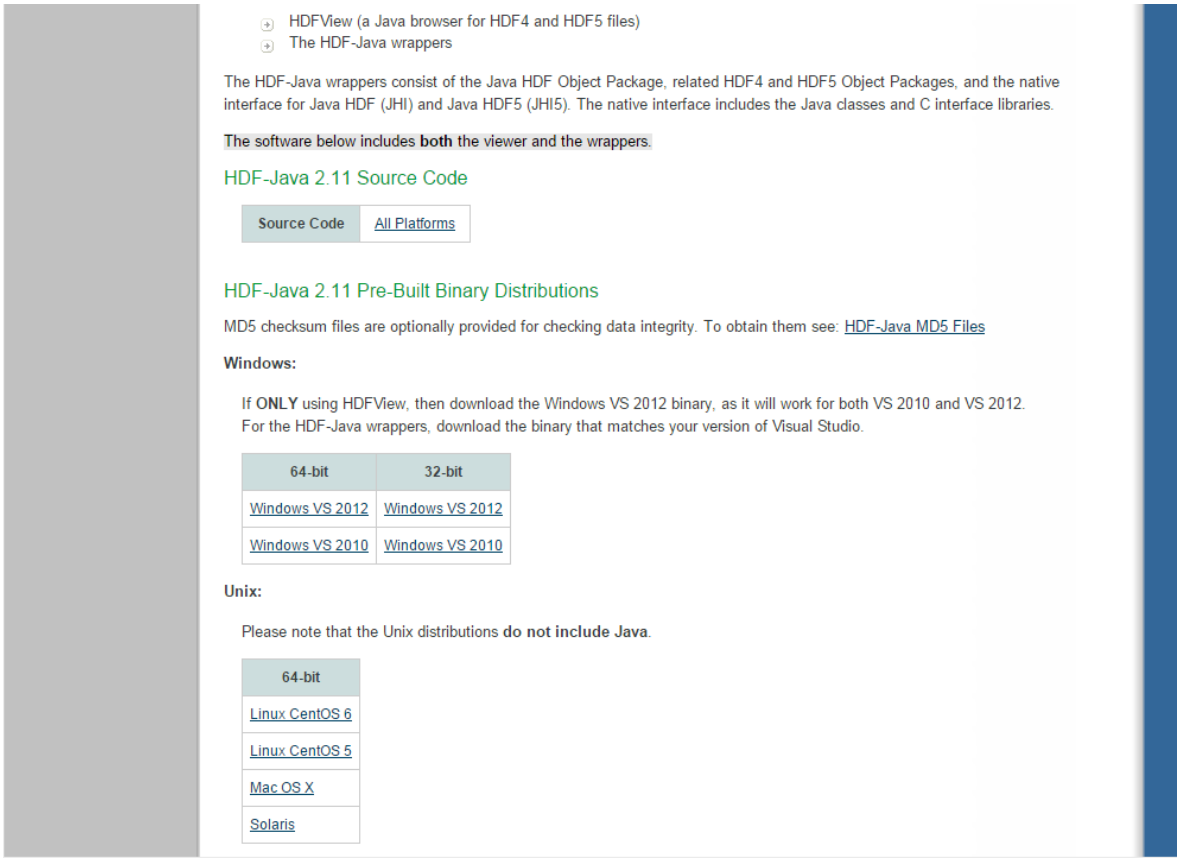

図 7.2　ダウンロードページ 表示例

HDFView を利用した CGNS ファイルの閲覧

HDFView を起動して CGNS ファイルを閲覧します。

まず、スタートメニューから HDFView を起動します。次に、以下のメニューから、開く CGNS ファイルを選択します。

File –> Open

HDFView では、拡張子 "*.cgn" のファイルはデフォルトでは開く対象に含まれないため、ファイルのタイプで「すべてのファイル」を指定した上で、CGNS ファイルを選択して開いて下さい。CGNS ファイルを開いた後の HDFView の画面表示例を図 7.3 に示します。

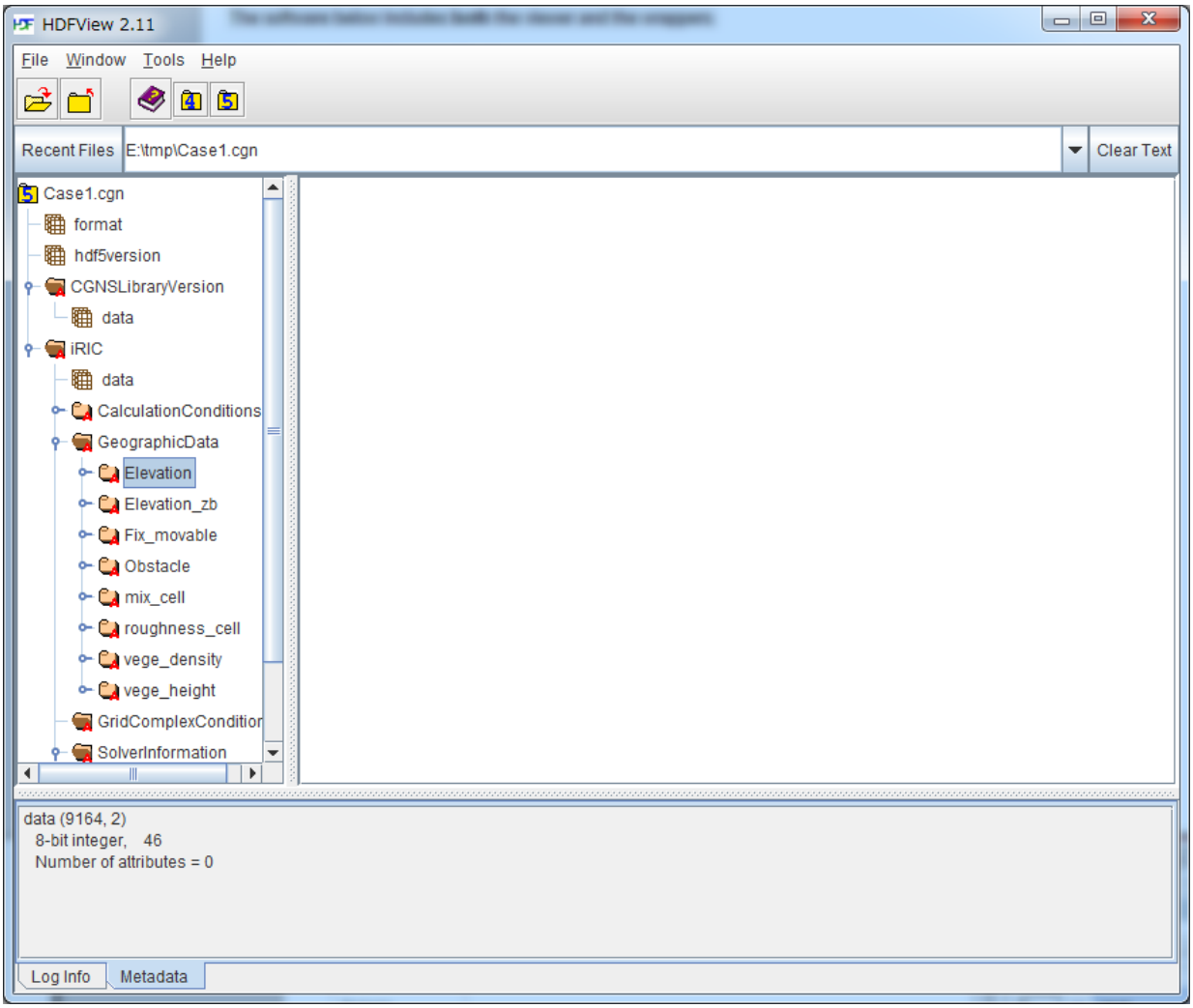

図 7.3　HDFView 表示例

画面の左側には、CGNS ファイル内のツリー構造が表示されます。ツリー構造で、閲覧したい項目を選択すると、画面右側に選択した項目の内部のデータが表示されます。

7.4.3 リンク集

CGNS ファイル及び CGNS ライブラリに関する情報は、 表 7.4 を参照してください。

表 7.4　CGNS ファイル、CGNS ライブラリ関連リンク

項目	URL
ホームページ	http://cgns.sourceforge.net/
関数リファレンス	http://cgns.github.io/CGNS_docs_current/midlevel/index.html
CGNS ファイルの内部構造	http://cgns.github.io/CGNS_docs_current/sids/index.html
記述例集	http://cgns.github.io/CGNS_docs_current/user/examples.html

7.5 iRIC インストーラ作成用リポジトリへの登録

7.5.1 はじめに

iRIC は、インストーラの作成及びオンラインアップデートのために公開するファイルの管理を、github というウェブサービスを利用して行っています。

ソルバ開発者は、 github に最新のソルバのファイルを登録することで、以下を行なうことができます。

- 次回インストーラが作成される際に同梱されるソルバを更新します
- 既に iRIC をインストールしているユーザが、 オプション -> メンテナンス 機能から、ソルバの更新版をオンラインアップデートで入手するできるようにします

この節では、ソルバ開発者が github に最新のソルバを登録する手順を説明します。

7.5.2 作業の流れ

github に最新のソルバを登録するには、以下の流れで作業を行います。

1. Subversion のクライアントのインストール (初回のみ)
2. サーバからのフォルダの取得 (チェックアウト)
3. 新しいファイルのコピー
4. 新しいファイルのサーバへの登録 (コミット)

github へのファイルの登録は、 Subversion と git の 2 つのバージョン管理システムを使って行なうことができますが、ここでは操作の簡単な Subversion を利用した場合の手順をご説明します。

以下で、詳しい手順を説明します。

7.5.3 Subversion のクライアントのインストール (初回のみ)

インストール

Subversion に関連した操作を行うためのクライアントをインストールします。この手順では、Windows 用の Subversion クライアントである TortoiseSVN をインストールします。

以下の URL にアクセスし、 TortoiseSVN のインストーラを入手します。

https://tortoisesvn.net/downloads.html

画面上には、 32bit 版 OS 用と 64bit 版 OS 用の 2 つのダウンロード用ボタンがあります。お使いの環境に合わせて適切なインストーラをダウンロードして下さい。

日本語でお使いになりたい場合は、画面の少し下にある Language packs を追加でインストールすることで、メニューが日本語になります。"Japanese" という行の "Setup" リンクをクリックしてダウンロードしてください。

インストーラがダウンロードできたら、まずは TortoiseSVN 本体、その後 Language pack の順番でインストールして下さい。

インストールが完了したら、一度 Windows を再起動します。

環境設定

インターネットに接続するのに、プロキシサーバを経由する必要がある環境では、設定を行います。

エクスプローラで右クリックメニューから以下を選択します。

TortoiseSVN –> 設定

すると、設定ダイアログが表示されます。左側のツリービューで「ネットワーク」を選択すると、図 7.4 に示すような画面が表示されますので、お使いの環境に合わせた設定を行い、「OK」ボタンを押します。

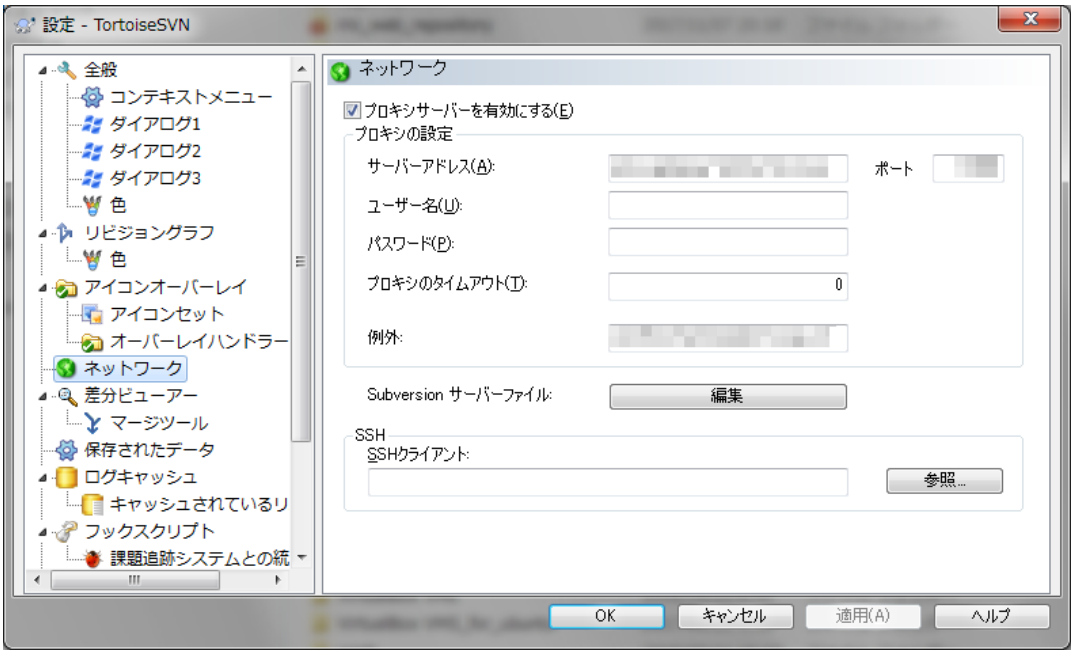

図 7.4　TortoiseSVN 設定ダイアログ

7.5.4 サーバからのフォルダの取得 (チェックアウト)

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages 以下のフォルダのうち、更新したいソルバが含まれているフォルダをチェックアウトします。

例えば、 FaSTMECH なら以下のフォルダをチェックアウトします。

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages/solver.fastmech

以下では、 FaSTMECH のフォルダを取得する際の例を示します。

フォルダの作成

サーバから取得したファイルを保存するためのフォルダを作成します。

この例では、e:tmpfastmech にフォルダを作成します。

サーバからのフォルダの取得

TortoiseSVN を利用して、サーバからフォルダを取得します。

上記で作成したフォルダをエクスプローラで選択し、右クリックメニューから以下を選択します。

SVN チェックアウト

すると、 図 7.5 に示すダイアログが表示されます。

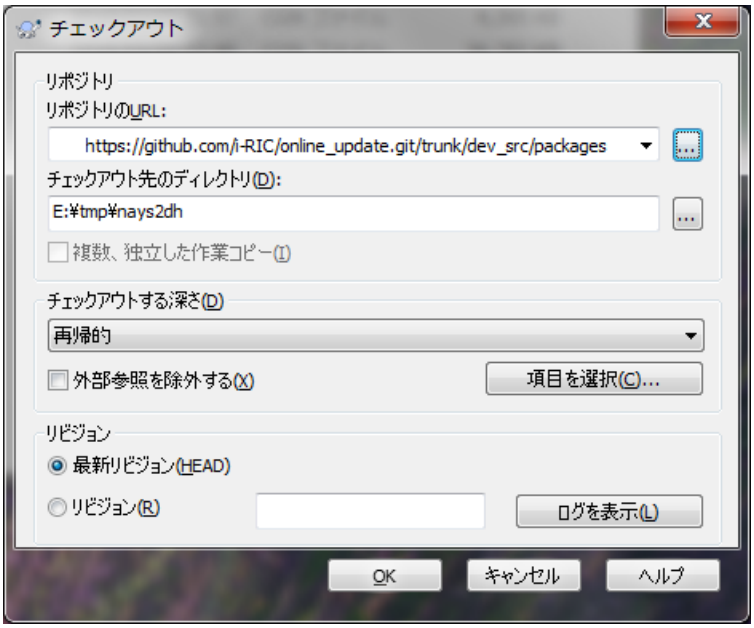

図 7.5　ファイルのチェックアウト用ダイアログ

「リポジトリの URL」欄に、以下の URL を入力します。

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages

その後、その右にある「...」ボタンを押します。すると、図 7.6 に示すダイアログが表示されます。

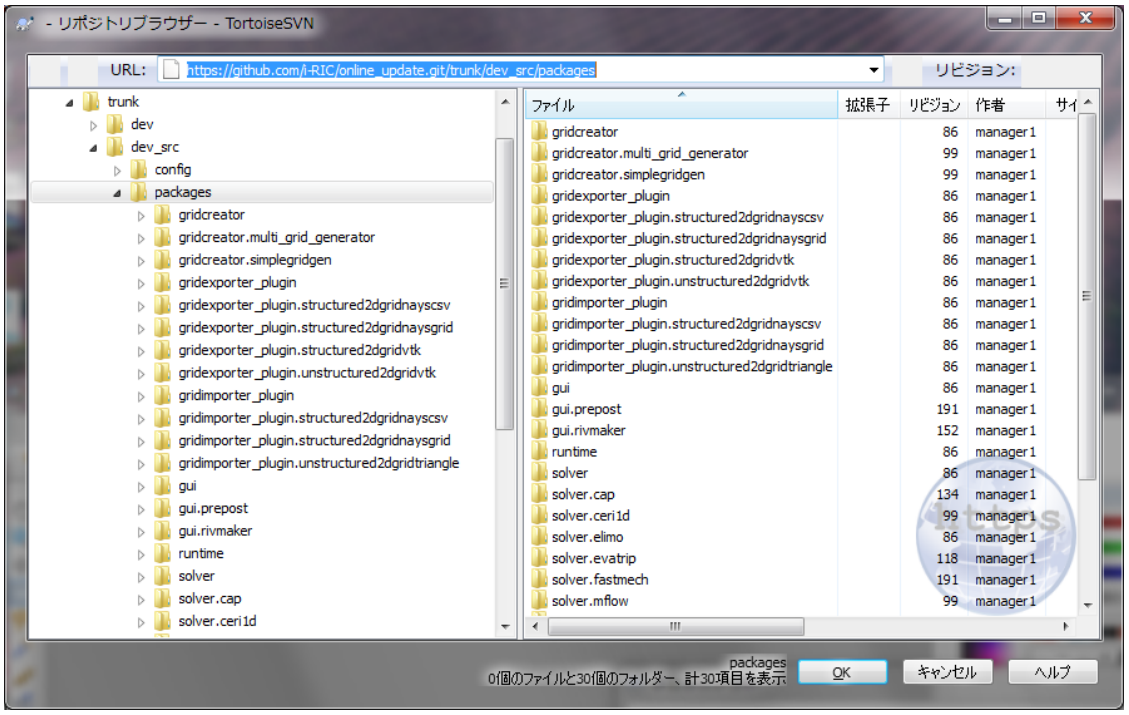

図 7.6　ファイルのチェックアウト用ダイアログ (フォルダの選択)

このダイアログで、自分が更新したいソルバが含まれているフォルダ (今回の例なら "solver.fastmech") を選択し、「OK」ボタンを押します。すると、「リポジトリの URL」 が更新されます。

図 7.5 に示すダイアログで、「リポジトリの URL」「チェックアウト先のディレクトリ」が正しく設定されていることを確認したら、「OK」ボタンを押します。すると、図 7.7 に示すようなダイアログが表示され、フォルダ内のファイルの取得が始まります。

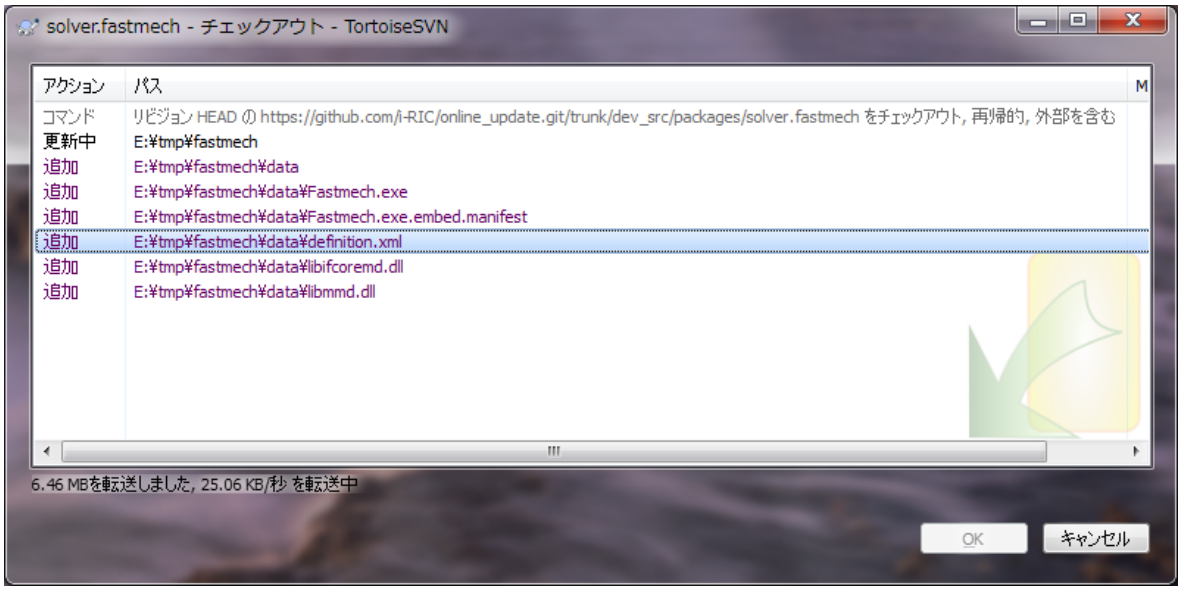

図 7.7　ファイルの取得進捗ダイアログ

ファイルの取得が完了すると、エクスプローラでは図 7.8 に示すように表示されます。チェックアウトされたファイルの横には、チェックマークのついたアイコンが表示されます。

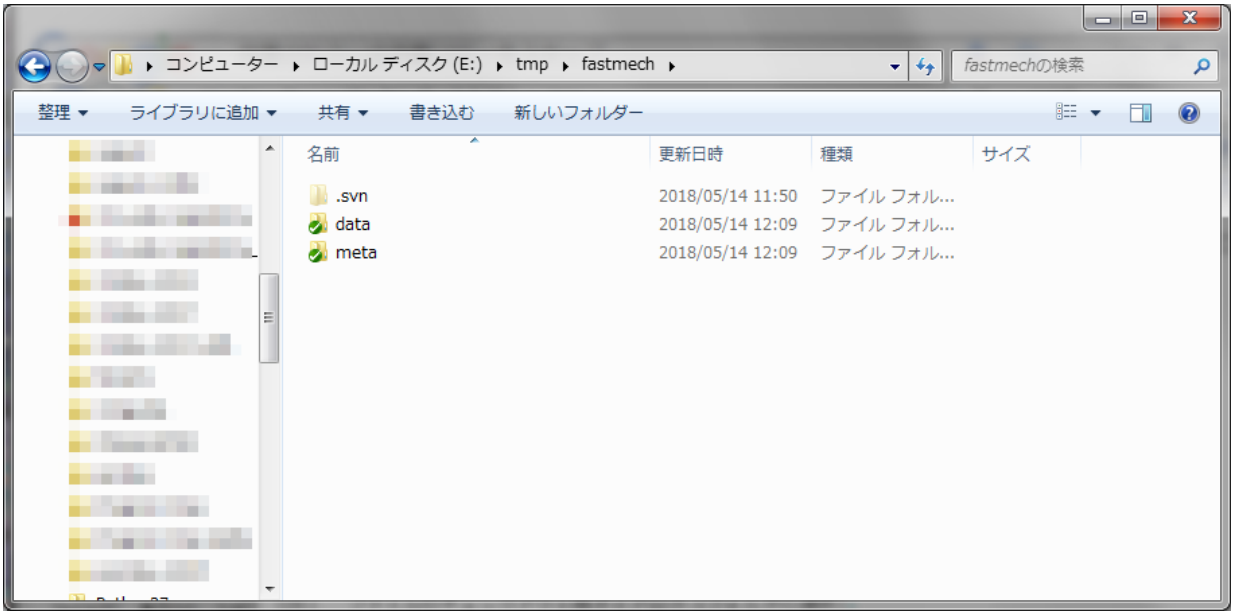

図 7.8　ファイルのチェックアウト後のエクスプローラ表示例

7.5.5 新しいファイルのコピー

チェックアウトしたフォルダに、インストーラに同梱したいファイルをコピーします。ファイルをコピーすると、ファイルの横のアイコンが以下のようになります。

- 上書きされたファイルの横には、「!」マークのついたアイコンが表示されます
- 新しくコピーされたファイルの横には、アイコンにマークが表示されません

新しくコピーされたファイルをサーバに送信するには、ファイルを右クリックして、右クリックメニューから以下を選択します。

TortoiseSVN –> 追加

追加を行うと、ファイルの横に「+」マークが表示されます。

"Fastmech.exe" を上書きし、 "newdll.dll" を追加した後のエクスプローラの表示例を図 7.9 に示します。

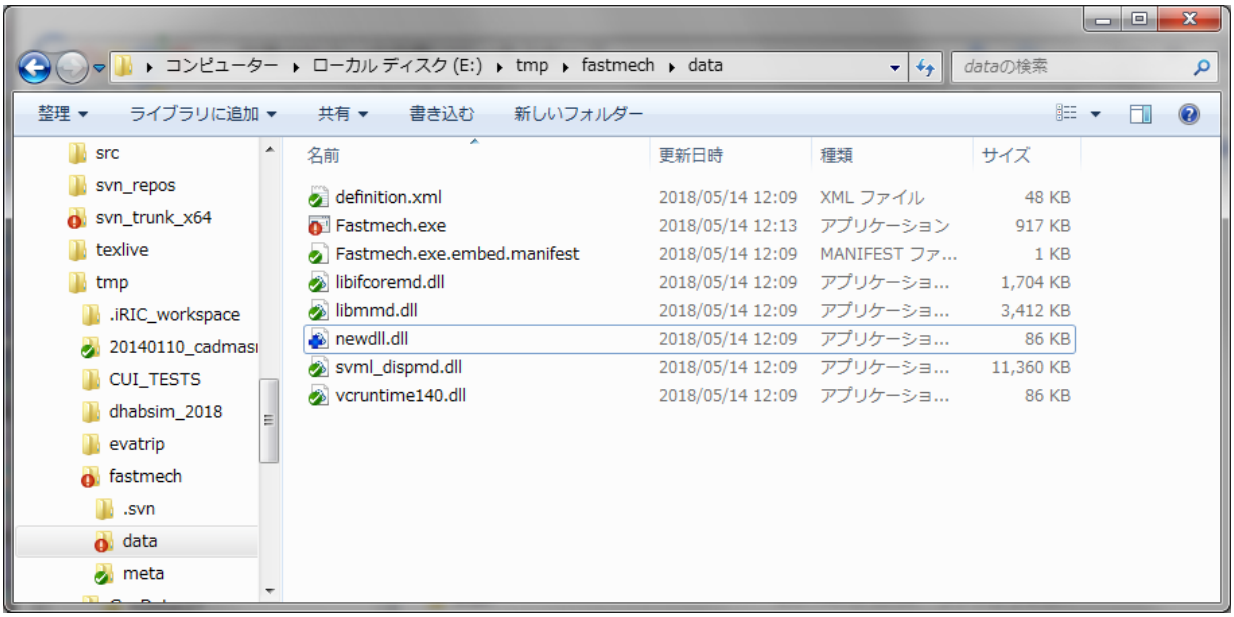

図 7.9　新しいファイルをコピーした後のエクスプローラ表示例

ご注意

ソルバの更新をする時は、ソルバの実行ファイルなどを更新するだけでなく、*definition.xml* に記述されたバージョン番号も更新してください。これは、バージョン番号が変わっていないと、 iRIC メンテナンスがファイルが更新されていることを認識できないためです。

バージョン番号は *definition.xml* の *SolverDefinition* 要素で、 *version* という名前の属性で指定されています。

7.5.6 新しいファイルのサーバへの登録 (コミット)

新しいファイルを、サーバへ登録します。

上記でファイルを登録したフォルダをエクスプローラで選択し、右クリックメニューから以下を選択します。

SVN コミット

すると、図 7.10 に示すダイアログが表示されます。

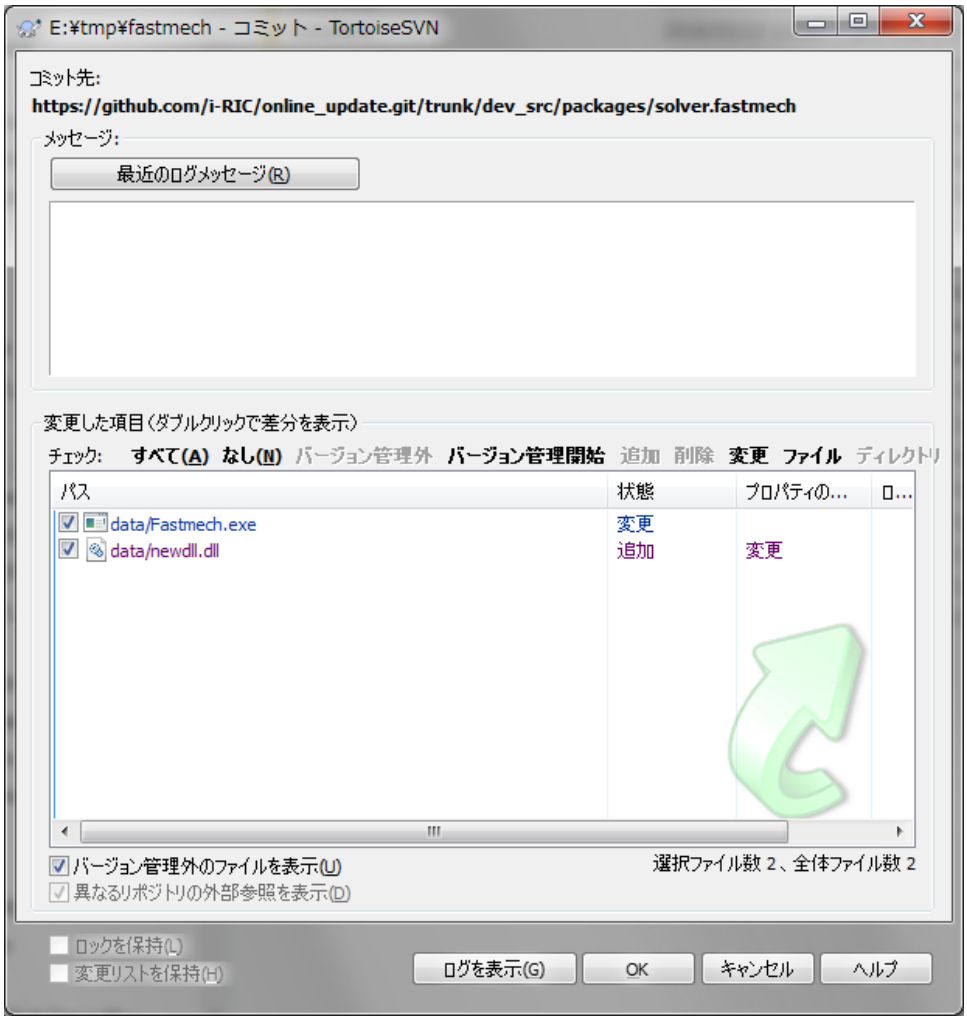

図 7.10　新しいファイルのコミット用ダイアログ

更新したいファイルが全てチェックされていることを確認したら、更新に関するログを追記して、「OK」ボタンを押します。

図 7.11 が表示されますので、ユーザ名とパスワードを入力して、「OK」ボタンを押します。

図 7.11　認証ダイアログ

ソルバの登録に使用する ユーザ名とパスワードについては、 iRIC の管理者にお問い合わせ下さい。

第 8 章

ご利用にあたって

- 本ソフトウェアを利用した成果を用いて論文、報告書、記事等の出版物を作成する場合は、本ソフトウェアを使用したことを適切な位置に示してください。
- iRIC サイトで提供している河川の地形データなどはサンプルデータであり、実際のものとは異なる場合があります。あくまでもテスト用としてご試用下さい。
- ご感想、ご意見、ご指摘は http://i-ric.org にて受け付けております。