



iRIC Software

Changing River Science

Developer's Manual

Last Update: 2015.3.13

Release: 2011.12.24

Copyright 2011-2015 iRIC Project All Rights Reserved.

目 次

1. このマニュアルについて	1
2. ソルバーの開発手順	2
2.1. 概要	2
2.2. フォルダの作成	5
2.3. ソルバー定義ファイルの作成	5
2.3.1. 基本情報の作成	5
2.3.2. 計算条件の定義	8
2.3.3. 格子属性の定義	12
2.3.4. 境界条件の定義	14
2.4. ソルバーの作成	17
2.4.1. 骨組みの作成	18
2.4.2. 計算データファイルを開く処理、閉じる処理の記述	19
2.4.3. 計算条件、計算格子、境界条件の読み込み処理の記述	20
2.4.4. 時刻、計算結果の出力処理の記述	23
2.5. ソルバー定義ファイルの辞書ファイルの作成	25
2.6. 説明ファイルの作成	28
2.7. ライセンス情報ファイルの作成	29
3. 計算結果分析ソルバーの開発手順	30
3.1. 概要	30
4. 格子生成プログラムの開発手順	33
4.1. 概要	33
4.2. フォルダの作成	36
4.3. 格子生成プログラム定義ファイルの作成	36
4.3.1. 基本情報の作成	37
4.3.2. 格子生成条件の定義	40
4.3.3. エラーコードの定義	44
4.4. 格子生成プログラムの作成	45
4.4.1. 骨組みの作成	46
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	47
4.4.3. 格子の出力処理の記述	48
4.4.4. 格子生成条件の読み込み処理の記述	51
4.4.5. エラー処理の記述	53
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	54
4.6. 説明ファイルの作成	57
5. 定義ファイル (XML) について	58
5.1. 概要	58
5.2. 構造	58
5.2.1. ソルバー定義ファイル	58
5.2.2. 格子生成プログラム定義ファイル	61
5.3. 定義例	62
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	62
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	75
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	76
5.4. 要素のリファレンス	82
5.4.1. BoundaryCondition	82
5.4.2. CalculationCondition	82

5.4.3.	Condition	83
5.4.4.	Definition (計算条件・格子属性・境界条件・格子生成条件の定義)	84
5.4.5.	Definition (格子属性の定義)	85
5.4.6.	Dimension	86
5.4.7.	Enumeration	86
5.4.8.	ErrorCodes	87
5.4.9.	ErrorCode	87
5.4.10.	GroupBox	87
5.4.11.	GridGeneratingCondition	87
5.4.12.	GridGeneratorDefinition	88
5.4.13.	GridLayout	89
5.4.14.	GridTypes	89
5.4.15.	GridType	89
5.4.16.	HBoxLayout	90
5.4.17.	Item	90
5.4.18.	Label	90
5.4.19.	Param	91
5.4.20.	SolverDefinition	92
5.4.21.	Tab	93
5.4.22.	Value	93
5.4.23.	VBoxLayout	94
5.5.	ソルバーのバージョンアップ時の注意点	95
5.6.	XML の基礎	97
5.6.1.	要素の書き方	97
5.6.2.	タブ、スペース、改行について	98
5.6.3.	コメントの書き方	98
6.	iRIClib について	99
6.1.	iRIClib とは	99
6.2.	この章の読み方	99
6.3.	概要	100
6.3.1.	プログラムの処理と iRIClib の関数	100
6.3.2.	CGNS ファイルを開く	101
6.3.3.	内部変数の初期化	101
6.3.4.	計算条件 (もしくは格子生成条件) の読み込み	102
6.3.5.	計算格子の読み込み	104
6.3.6.	境界条件の読み込み	107
6.3.7.	地形データの読み込み	109
6.3.8.	計算格子の出力	112
6.3.9.	時刻 (もしくはループ回数) の出力	115
6.3.10.	計算格子の出力 (計算開始後の格子)	116
6.3.11.	計算結果の出力	118
6.3.12.	既存の計算結果の読み込み	120
6.3.13.	エラーコードの出力	122
6.3.14.	CGNS ファイルを閉じる	122
6.4.	リファレンス	123
6.4.1.	サブルーチン一覧	123
6.4.2.	cg_open_f	130
6.4.3.	cg_irc_init_f	130
6.4.4.	cg_irc_initread_f	131
6.4.5.	cg_irc_read_integer_f	131
6.4.6.	cg_irc_read_real_f	131
6.4.7.	cg_irc_read_realsingle_f	132
6.4.8.	cg_irc_read_string_f	132
6.4.9.	cg_irc_read_functionalsize_f	132
6.4.10.	cg_irc_read_functional_f	133

6.4.11.	cg_irc_read_functional_realsingle_f	133
6.4.12.	cg_irc_read_functionalwithname_f	134
6.4.13.	cg_irc_gotogridcoord2d_f	134
6.4.14.	cg_irc_gotogridcoord3d_f	134
6.4.15.	cg_irc_getgridcoord2d_f	135
6.4.16.	cg_irc_getgridcoord3d_f	135
6.4.17.	cg_irc_read_grid_integer_node_f	136
6.4.18.	cg_irc_read_grid_real_node_f	136
6.4.19.	cg_irc_read_grid_integer_cell_f	136
6.4.20.	cg_irc_read_grid_real_cell_f	137
6.4.21.	cg_irc_read_complex_count_f	137
6.4.22.	cg_irc_read_complex_integer_f	137
6.4.23.	cg_irc_read_complex_real_f	138
6.4.24.	cg_irc_read_complex_realsingle_f	138
6.4.25.	cg_irc_read_complex_string_f	139
6.4.26.	cg_irc_read_complex_functionalsize_f	139
6.4.27.	cg_irc_read_complex_functional_f	140
6.4.28.	cg_irc_read_complex_functional_realsingle_f	140
6.4.29.	cg_irc_read_complex_functionalwithname_f	141
6.4.30.	cg_irc_read_grid_complex_node_f	141
6.4.31.	cg_irc_read_grid_complex_cell_f	142
6.4.32.	cg_irc_read_grid_functionaltimesize_f	142
6.4.33.	cg_irc_read_grid_functionaltime_f	142
6.4.34.	cg_irc_read_grid_functionaldimensionsize_f	143
6.4.35.	cg_irc_read_grid_functionaldimension_integer_f	143
6.4.36.	cg_irc_read_grid_functionaldimension_real_f	144
6.4.37.	cg_irc_read_grid_functional_integer_node_f	144
6.4.38.	cg_irc_read_grid_functional_real_node_f	145
6.4.39.	cg_irc_read_grid_functional_integer_cell_f	145
6.4.40.	cg_irc_read_grid_functional_real_cell_f	146
6.4.41.	cg_irc_read_bc_count_f	146
6.4.42.	cg_irc_read_bc_indicessize_f	147
6.4.43.	cg_irc_read_bc_indices_f	148
6.4.44.	cg_irc_read_bc_integer_f	149
6.4.45.	cg_irc_read_bc_real_f	149
6.4.46.	cg_irc_read_bc_realsingle_f	150
6.4.47.	cg_irc_read_bc_string_f	150
6.4.48.	cg_irc_read_bc_functionalsize_f	151
6.4.49.	cg_irc_read_bc_functional_f	151
6.4.50.	cg_irc_read_bc_functional_realsingle_f	152
6.4.51.	cg_irc_read_bc_functionalwithname_f	152
6.4.52.	cg_irc_read_geo_count_f	153
6.4.53.	cg_irc_read_geo_filename_f	153
6.4.54.	irc_geo_polygon_open_f	154
6.4.55.	irc_geo_polygon_read_integervalue_f	154
6.4.56.	irc_geo_polygon_read_realvalue_f	154
6.4.57.	irc_geo_polygon_read_pointcount_f	155
6.4.58.	irc_geo_polygon_read_points_f	155
6.4.59.	irc_geo_polygon_read_holecount_f	155
6.4.60.	irc_geo_polygon_read_holepointcount_f	156
6.4.61.	irc_geo_polygon_read_holepoints_f	156
6.4.62.	irc_geo_polygon_close_f	157
6.4.63.	irc_geo_riversurvey_open_f	157
6.4.64.	irc_geo_riversurvey_read_count_f	157
6.4.65.	irc_geo_riversurvey_read_position_f	158
6.4.66.	irc_geo_riversurvey_read_direction_f	158

6.4.67.	iric_geo_riversurvey_read_name_f	159
6.4.68.	iric_geo_riversurvey_read_realname_f	159
6.4.69.	iric_geo_riversurvey_read_leftshift_f	159
6.4.70.	iric_geo_riversurvey_read_altitudecount_f	160
6.4.71.	iric_geo_riversurvey_read_altitudes_f	160
6.4.72.	iric_geo_riversurvey_read_fixedpointl_f	161
6.4.73.	iric_geo_riversurvey_read_fixedpointtr_f	161
6.4.74.	iric_geo_riversurvey_read_watersurfaceelevation_f	162
6.4.75.	iric_geo_riversurvey_close_f	162
6.4.76.	cg_iric_writegridcoord1d_f	162
6.4.77.	cg_iric_writegridcoord2d_f	163
6.4.78.	cg_iric_writegridcoord3d_f	163
6.4.79.	cg_iric_write_grid_integer_node_f	164
6.4.80.	cg_iric_write_grid_real_node_f	164
6.4.81.	cg_iric_write_grid_integer_cell_f	164
6.4.82.	cg_iric_write_grid_real_cell_f	165
6.4.83.	cg_iric_write_sol_time_f	165
6.4.84.	cg_iric_write_sol_iteration_f	165
6.4.85.	cg_iric_write_sol_gridcoord2d_f	166
6.4.86.	cg_iric_write_sol_gridcoord3d_f	166
6.4.87.	cg_iric_write_sol_baseiterative_integer_f	167
6.4.88.	cg_iric_write_sol_baseiterative_real_f	167
6.4.89.	cg_iric_write_sol_integer_f	167
6.4.90.	cg_iric_write_sol_real_f	168
6.4.91.	cg_iric_write_sol_particle_pos2d_f	168
6.4.92.	cg_iric_write_sol_particle_pos3d_f	169
6.4.93.	cg_iric_read_sol_count_f	169
6.4.94.	cg_iric_read_sol_time_f	169
6.4.95.	cg_iric_read_sol_iteration_f	170
6.4.96.	cg_iric_read_sol_baseiterative_integer_f	170
6.4.97.	cg_iric_read_sol_baseiterative_real_f	170
6.4.98.	cg_iric_read_sol_gridcoord2d_f	171
6.4.99.	cg_iric_read_sol_gridcoord3d_f	171
6.4.100.	cg_iric_read_sol_integer_f	172
6.4.101.	cg_iric_read_sol_real_f	172
6.4.102.	cg_iric_write_errorcode_f	172
6.4.103.	cg_close_f	173
7.	その他の情報	174
7.1.	Fortran プログラムでの引数の読み込み処理	174
7.1.1.	Intel Fortran Compiler	174
7.1.2.	GNU Fortran, G95	174
7.2.	Fortran 言語で iriclib, cgnslib とリンクしてビルドする方法	175
7.2.1.	Intel Fortran Compiler (Windows)	175
7.2.2.	GNU Fortran	175
7.3.	特別な格子属性、計算結果の名前について	176
7.3.1.	格子属性	176
7.3.2.	計算結果	177
7.4.	CGNS ファイル、CGNS ライブラリに関する情報	178
7.4.1.	CGNS ファイルフォーマットの概要	178
7.4.2.	CGNS ファイルの閲覧方法	178
7.4.3.	リンク集	182

1. このマニュアルについて

このマニュアルでは、以下のユーザに必要な情報を解説します。

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

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

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

2. ソルバーの開発手順

2.1. 概要

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

iRIC 上で動作するソルバーを開発するには、表 2-1 に示すようなファイルを作成、配置する必要があります。

表 2-1 に示した項目のうち、“iRIC 2.0” フォルダと “solvers” フォルダは、iRIC をインストールすれば既に作成されています。ソルバー開発者は、“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、関連するファイルをその下に配置します。

表 2-1 ソルバー関連ファイル、フォルダー一覧

ファイル名、フォルダ名	説明	参照
iRIC 2.0	iRIC 2.0 のインストールフォルダ (例: C:\Program Files\iRIC 2.0)	
solvers	ソルバーを格納するフォルダ	
(ソルバーフォルダ)	ソルバーごとにフォルダを作成する。フォルダ名は任意。	2.2 節
definition.xml	ソルバー定義ファイル。英語で記述する。	2.3 節
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。	2.4 節
translation_ja_JP.ts など	ソルバー定義ファイルの辞書ファイル。	2.5 節
README	ソルバーの説明ファイル	2.6 節
LICENSE	ソルバーのライセンス情報ファイル	2.7 節

各ファイルの概要は以下の通りです。

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。

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。

FORTTRAN, 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、FORTTRAN で開発する例を説明します。

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。

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。

LICENSE

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

iRIC、ソルバー、関連ファイルの関係を 図 2-1 に示します。

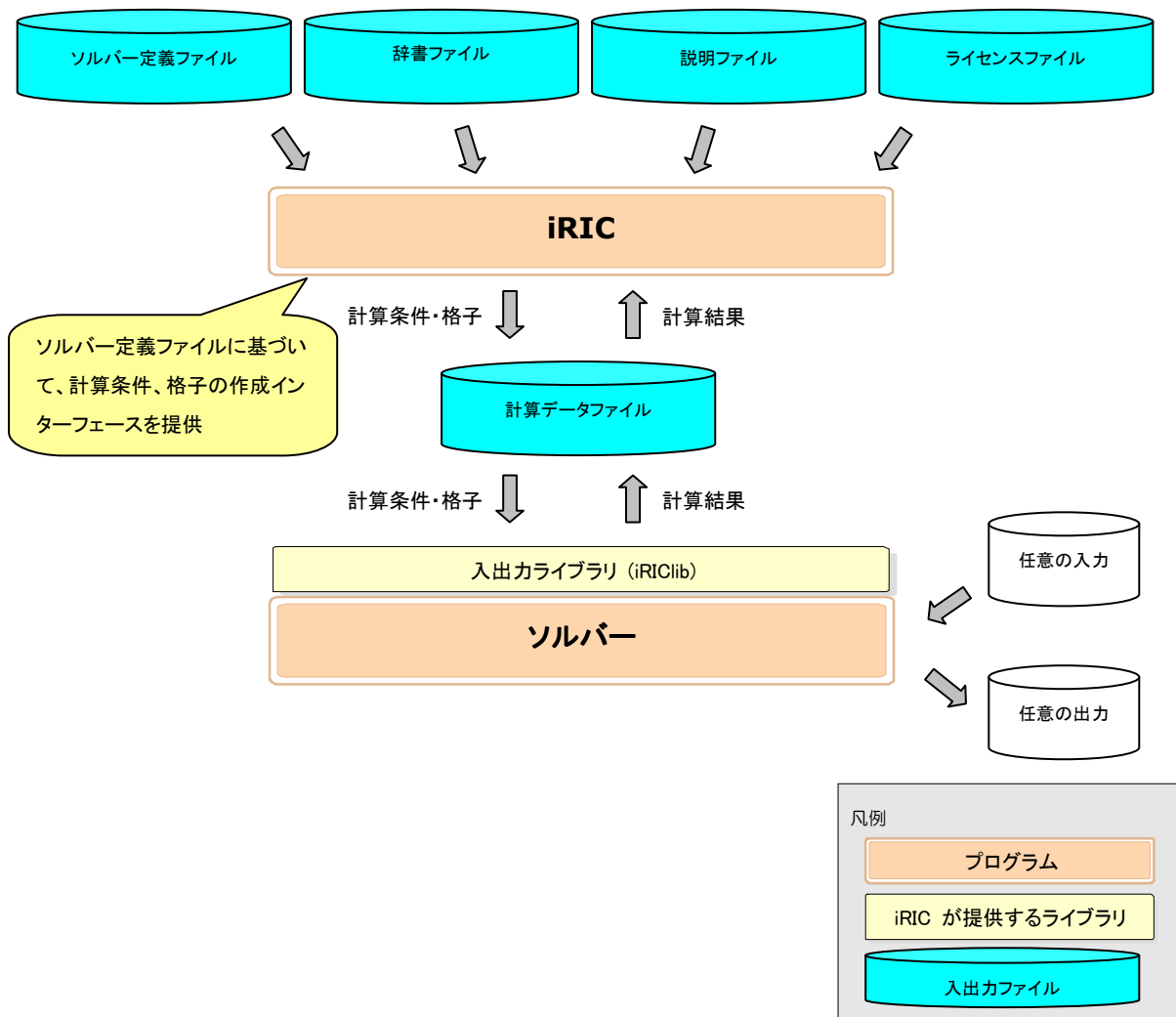


図 2-1 iRIC、ソルバー、関連ファイルの関係図

この章では、この節で説明した各ファイルを作成する手順を順番に説明します。

2.2. フォルダの作成

iRIC のインストールフォルダ（デフォルトでは “C:\Program Files\iRIC 2.0”）の下にある “solvers” フォルダの下に、開発するソルバーのための専用のフォルダを作成します。今回は、“example” というフォルダを作成します。

2.3. ソルバー定義ファイルの作成

ソルバー定義ファイルを作成します。

ソルバー定義ファイルは、ソルバーに関する 表 2-2 に示す情報を定義します。

表 2-2 ソルバー定義ファイルで定義する情報

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

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

この節では、ソルバー定義ファイルを、表 2-2 に示した順で作成していきます。

2.3.1. 基本情報の作成

ソルバーの基本情報を作成します。表 2-3 に示すようなファイルを作り、2.2 で作成した “example” フォルダの下に “definition.xml” の名前で保存します。

表 2-3 基本情報を記述したソルバー定義ファイルの例

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

この時点では、ソルバー定義ファイルの構造は 表 2-4 に示すようになっています。

表 2-4 ソルバー定義ファイルの構造

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。

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 2-2 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 2-3 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

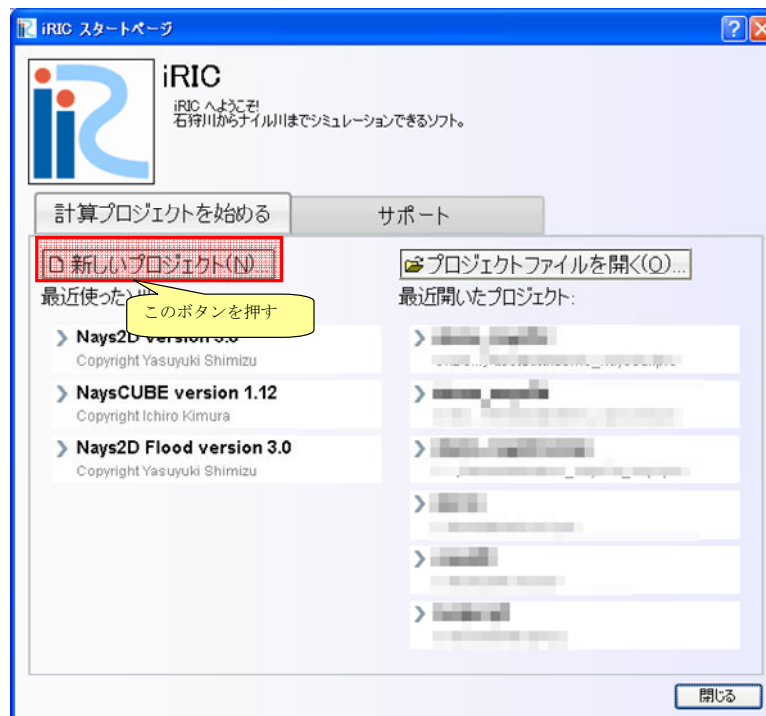


図 2-2 iRIC のスタートダイアログ

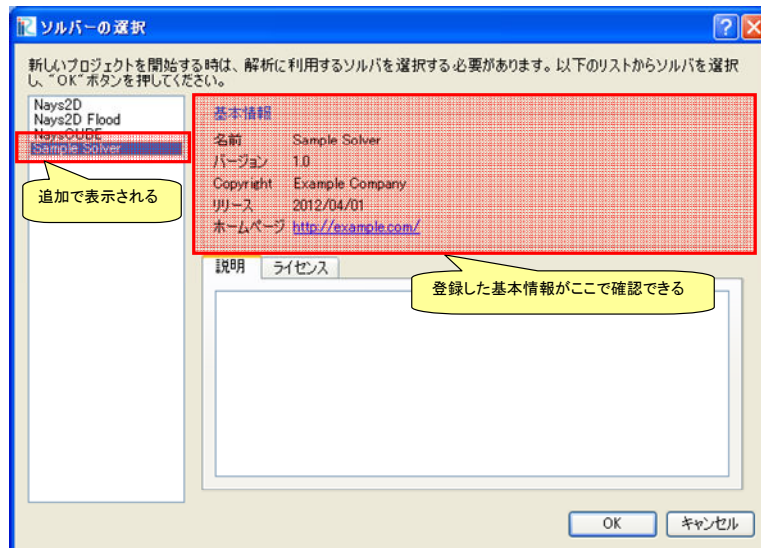


図 2-3 ソルバー選択ダイアログ

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

2.3.2. 計算条件の定義

計算条件を定義します。計算条件は、ソルバー定義ファイルの **CalculationCondition** 要素で定義します。2.3.1 で作成したソルバー定義ファイルに追記し、表 2-5 に示すようなファイルにし、保存します。追記した部分を太字で示しました。

表 2-5 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 表 2-6 に示すようになっています。

表 2-6 ソルバー定義ファイルの構造

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。

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 2-2 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

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 2-4 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。

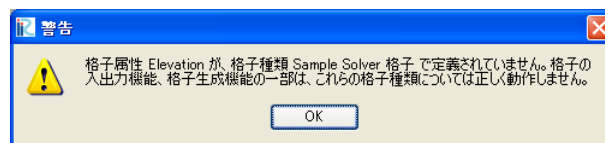


図 2-4 警告ダイアログ 表示例

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 2-5 に示すダイアログが表示されます。表 2-5 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。

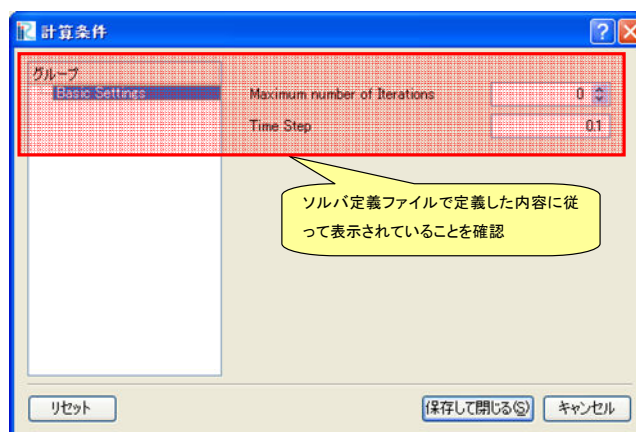


図 2-5 計算条件設定ダイアログ 表示例

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、表 2-7 に示します。追記した部分を太字で示しました。

表 2-7 計算条件を追記したソルバー定義ファイルの例 (抜粋)

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

この時点では、ソルバー定義ファイルの構造は 表 2-8 に示すようになっています。

表 2-8 ソルバー定義ファイルの構造

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。

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 2-6 に示します。

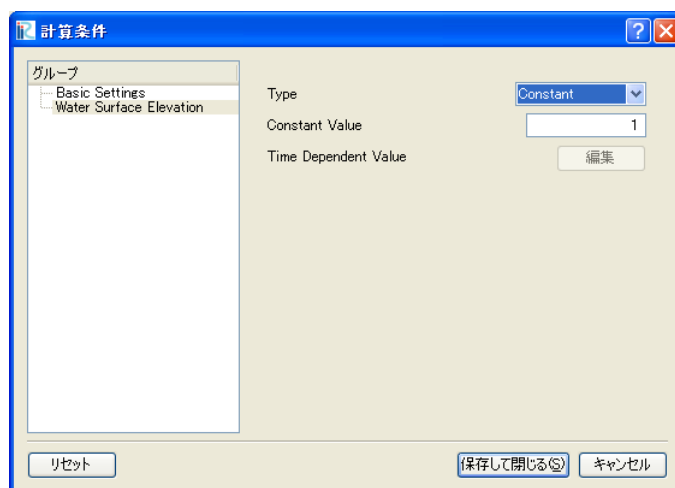


図 2-6 計算条件設定ダイアログ 表示例

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 5.3.1 を参照して下さい。
- 計算条件には、**Condition** 要素で依存関係を定義できます。**Condition** 要素では、その計算条件が有効になる条件を指定します。**Condition** 要素の定義方法の例は、5.3.2 を参照して下さい。
- この例では、計算条件のダイアログを単純なリスト形式で作成しましたが、グループボックスを使うなどしてダイアログのレイアウトをカスタマイズすることができます。ダイアログのレイアウトのカスタマイズ方法については 5.3.3 を参照して下さい。

2.3.3. 格子属性の定義

格子属性を定義します。格子属性は、ソルバー定義ファイルの `GridRelatedCondition` 要素で定義します。2.3.2 で作成したソルバー定義ファイルに追記し、`GridRelatedCondition` 要素に表 2-9 に示すように追記し、保存します。追記した部分を太字で示しました。

表 2-9 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 2-7 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 2-8 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。

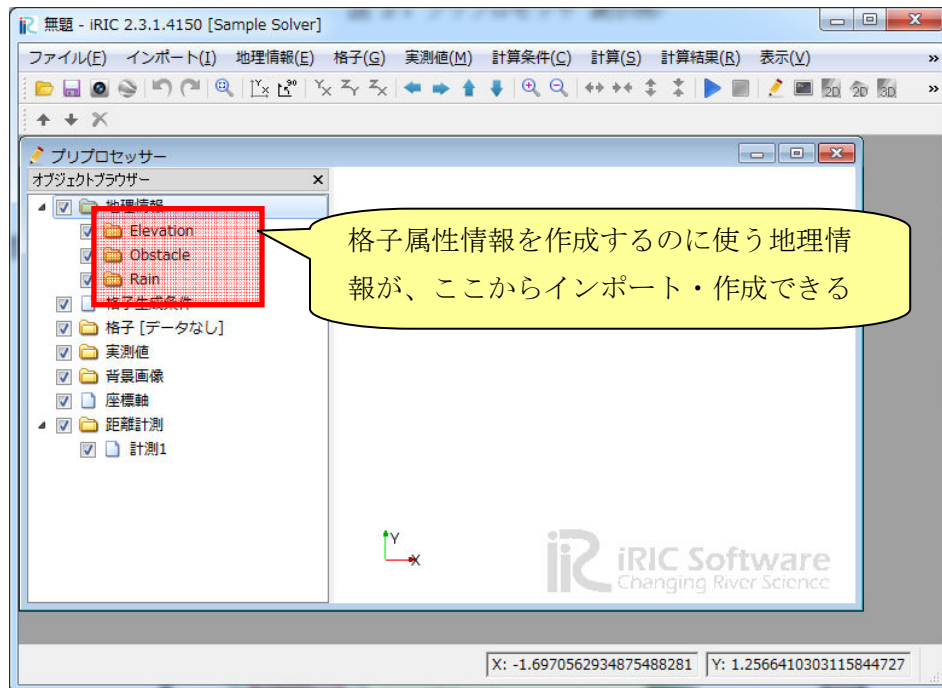


図 2-7 プリプロセッサ 表示例

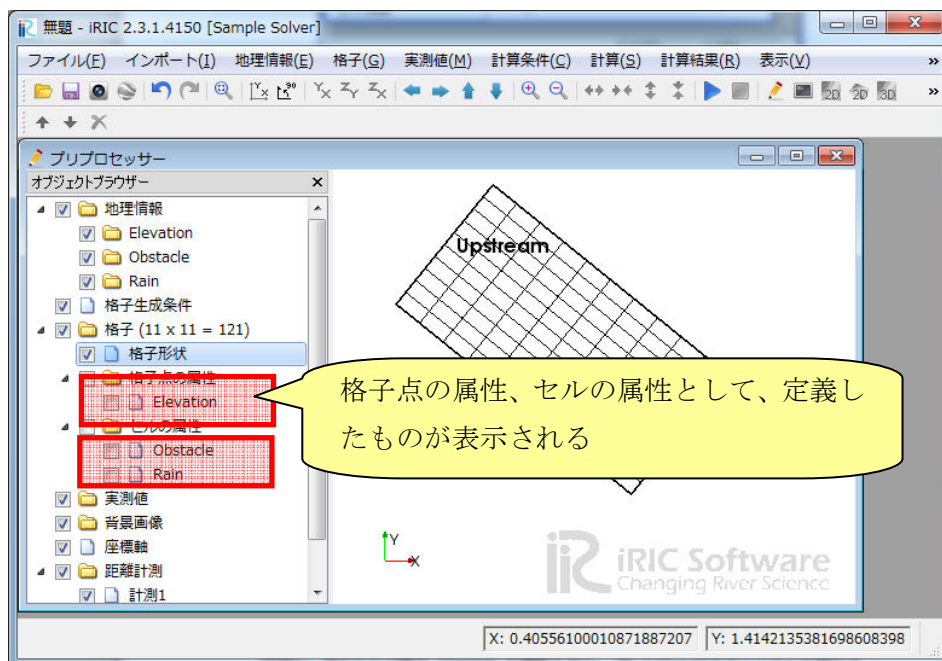


図 2-8 プリプロセッサ 表示例 (格子作成後)

以下の手順で格子点の属性 Elevation を編集すると、図 2-9 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。

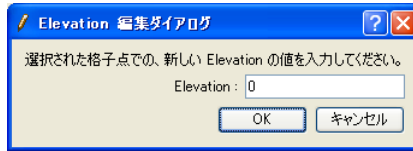


図 2-9 格子点の属性 “Elevation” の編集ダイアログ

同様に、格子セルの属性 **Obstacle** を編集すると、図 2-10 に示すダイアログが表示され、表 2-9 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。

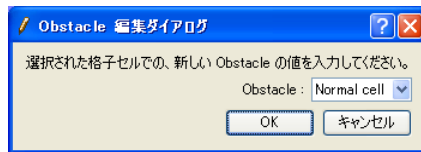


図 2-10 格子セルの属性 “Obstacle” の編集ダイアログ

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、特定の目的で使用する属性ではその名前を使用する必要があります。特別な格子属性の名前については 7.3.1 を参照してください。

2.3.4. 境界条件の定義

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

2.3.3 で作成したソルバー定義ファイルに追記し、**BoundaryCondition** 要素を表 2-10 に示すように追記し、保存します。追記した部分を太字で示しました。

表 2-10 境界条件を追記したソルバー定義ファイルの例（抜粋）

<pre> (前略) </GridRelatedCondition> <BoundaryCondition name="inflow" caption="Inflow" position="node"> <Item name="Type" caption="Type"> <Definition valueType="integer" default="0" > </pre>
--

```

<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 2-8 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。

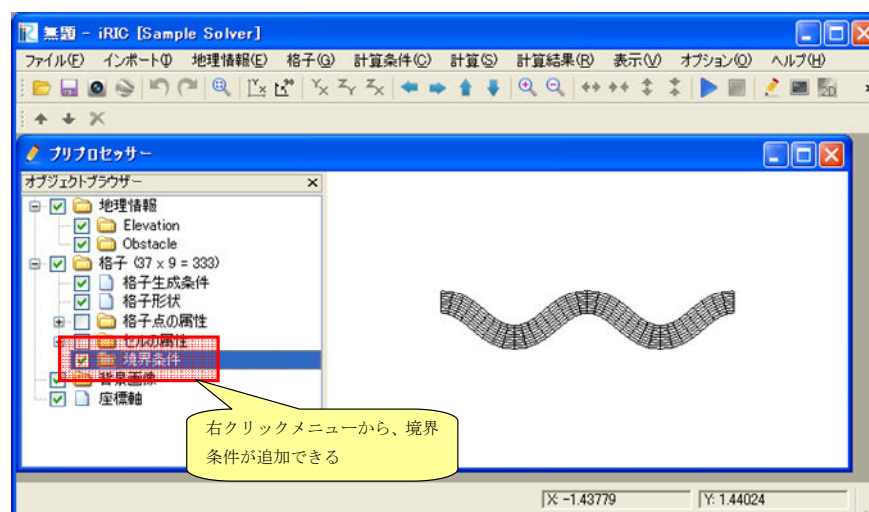


図 2-11 プリプロセッサ 表示例（格子作成後）

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 2-12 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 2-12 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 2-13 に示します。

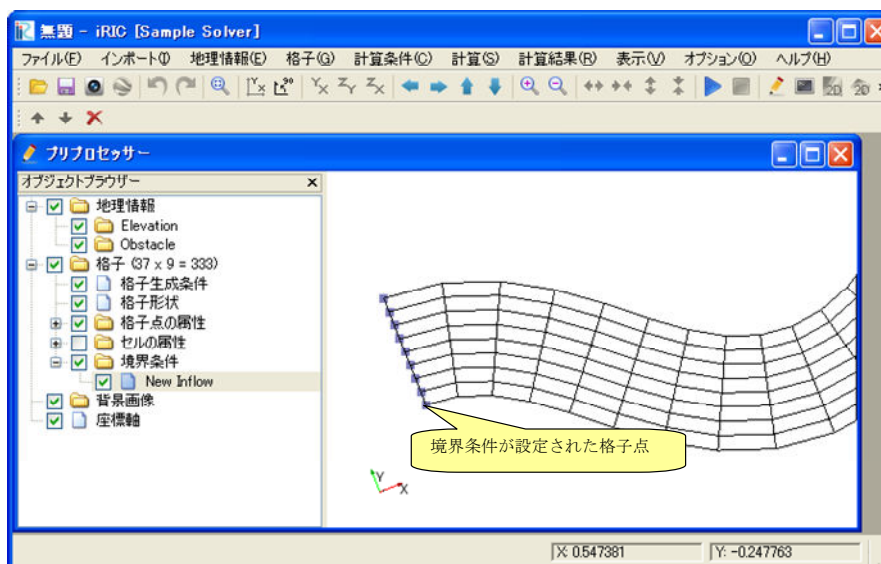


図 2-13 境界条件を設定した格子の表示例

境界条件の定義についてまとめると、以下の通りです。

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

2.4. ソルバーの作成

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

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

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

この節では、2.3 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 2-11 に示します。

表 2-11 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	○

} 複数回
繰り返す

この節では、ソルバーを以下の手順で開発していきます。

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

2.4.1. 骨組みの作成

まずは、ソルバーの骨組みを作成します。表 2-12 に示すソースコードを作成して、sample.f90 という名前で保存します。この時点では、ソルバーは何もしていません。

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

表 2-12 サンプルソルバー ソースコード

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

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

コンパイルが成功したら、できた実行プログラムを 2.2 で作成したフォルダにコピーし、名前を 2.3.1 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 2-14 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。

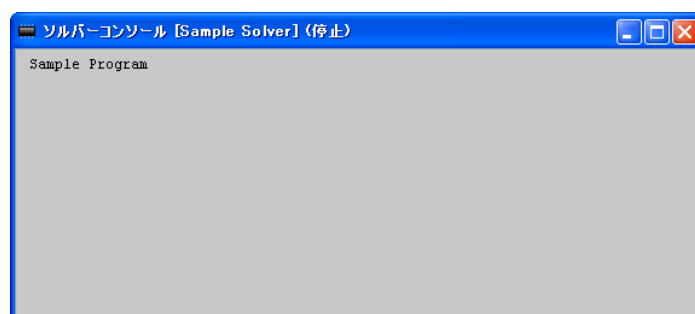


図 2-14 ソルバーコンソール表示例

2.4.2. 計算データファイルを開く処理、閉じる処理の記述

計算データファイルを開く処理、閉じる処理を記述します。

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

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

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

計算データファイルを開く処理と閉じる処理を追記したソースコードを 表 2-13 に示します。太字で示したのが追記した部分です。

表 2-13 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

  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 ****"

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

2.4.1 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。

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

この節で追加した関数の詳細については、6.3.2, 6.3.3, 6.3.14 を参照してください。

2.4.3. 計算条件、計算格子、境界条件の読み込み処理の記述

計算条件、計算格子、境界条件の読み込み処理を記述します。

iRIC は、2.3 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 表 2-14 に示します。太字で示したのが追記した部分です。

表 2-14 計算条件、計算格子、境界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
  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_irc_init_f(fin, ier)
if (ier /=0) STOP "**** Initialize error of CGNS file ****"

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

call cg_irc_read_functionalsize_f("variableSurface", variable_surface_size, ier)
allocate(variable_surface_time(variable_surface_size))
```

```

allocate(variable_surface_elevation(variable_surface_size))
call cg_irc_read_functional_f("variableSurface", variable_surface_time, variable_surface_elevation, ier)

! 格子のサイズを調べる
call cg_irc_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_irc_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_irc_read_grid_real_node_f("Elevation", elevation, ier)
call cg_irc_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_irc_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_irc_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

! 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_irc_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_irc_read_bc_integer_f('inflow', inflowid, 'Type', discharge_type(inflowid:inflowid), ier)
  ! 流入量 (一定)
  call cg_irc_read_bc_real_f('inflow', inflowid, 'ConstantDischarge',
discharge_constant(inflowid:inflowid), ier)
  ! 流入量 (時間依存)
  call cg_irc_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、2.3.2, 2.3.3 でソルバー定義ファイルに定義した **Item** 要素の **name** 属性と一致していることに注目してください。

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

また、計算条件、計算格子、格子属性の読み込みに使う関数の詳細については、6.3.4, 6.3.5 を参照してください。

2.4.4. 時刻、計算結果の出力処理の記述

時刻、計算結果の出力処理を記述します。

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

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表 2-15 に示します。太字で示したのが追記した部分です。

表 2-15 時刻、計算結果の出力処理を追記したソースコード

```
(前略)
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
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_irc_read_grid_real_node_f("Elevation", elevation, ier)
call cg_irc_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 cg_irc_write_sol_time_f(time, ier)
  ! 格子を出力
  call cg_irc_write_sol_gridcoord2d_f(grid_x, grid_y, ier)
  ! 計算結果を出力
  call cg_irc_write_sol_real_f('VelocityX', velocity_x, ier)
  call cg_irc_write_sol_real_f('VelocityY', velocity_y, ier)
  call cg_irc_write_sol_real_f('Depth', depth, ier)
  call cg_irc_write_sol_integer_f('Wet', wetflag, ier)
  call cg_irc_write_sol_baseiterative_real_f('Convergence', convergence, ier)
  iteration = iteration + 1
  if (iteration > maxiterations) exit
end do

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

時刻、計算結果の出力に使う関数の詳細については、6.3.9, 6.3.11 を参照してください。
計算実行中に格子形状が変化する場合、6.3.10 で説明する関数を使用してください。

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

2.5. ソルバー定義ファイルの辞書ファイルの作成

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

まず、iRIC から、以下のメニューを起動します。すると、ソルバー定義ファイルの辞書更新ウィザードが表示されます。ダイアログの表示例を、図 2-15 ～ 図 2-17 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)

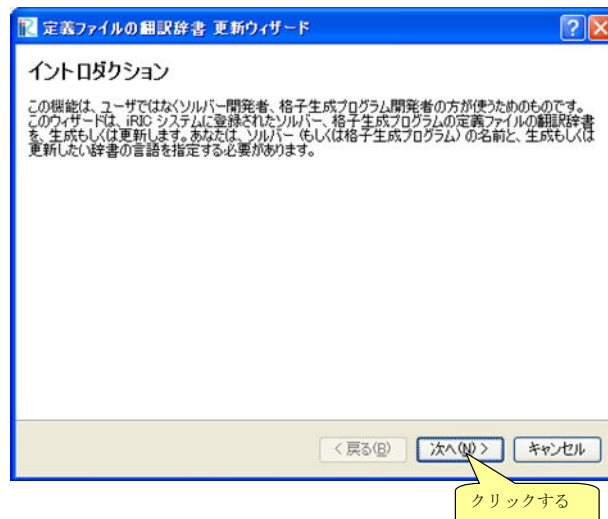


図 2-15 ソルバー定義ファイルの辞書更新ウィザード 表示例 (1 ページ目)

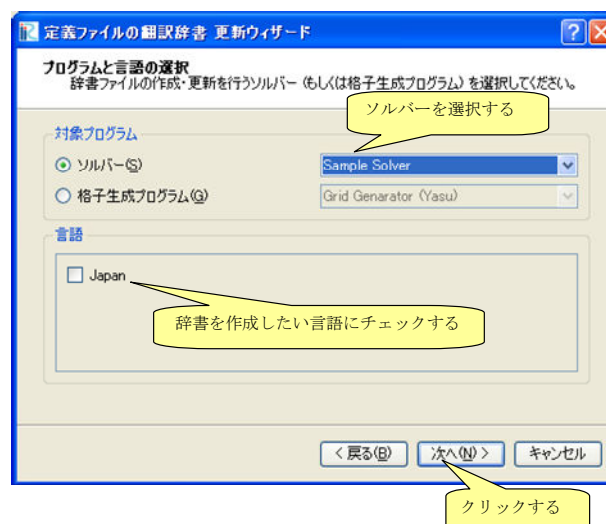


図 2-16 ソルバー定義ファイルの辞書更新ウィザード 表示例 (2 ページ目)

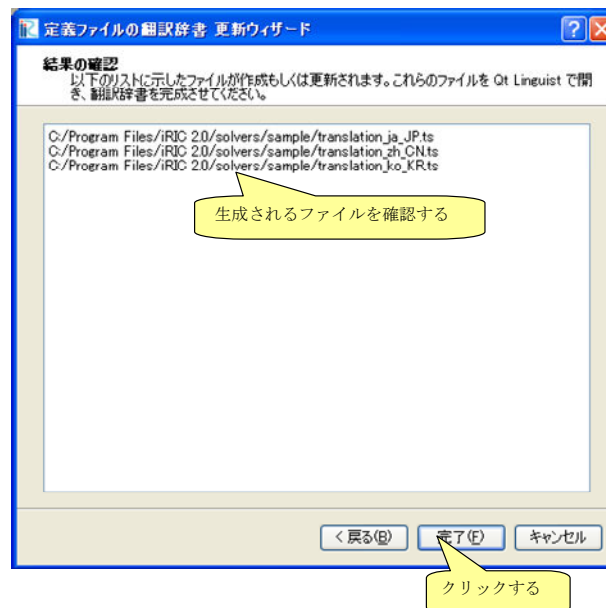


図 2-17 ソルバー定義ファイルの辞書更新ウィザード 表示例 (3 ページ目)

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、表 2-16、表 2-17 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)

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

表 2-17 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。

<http://qt.nokia.com/downloads/windows-cpp-vs2008>

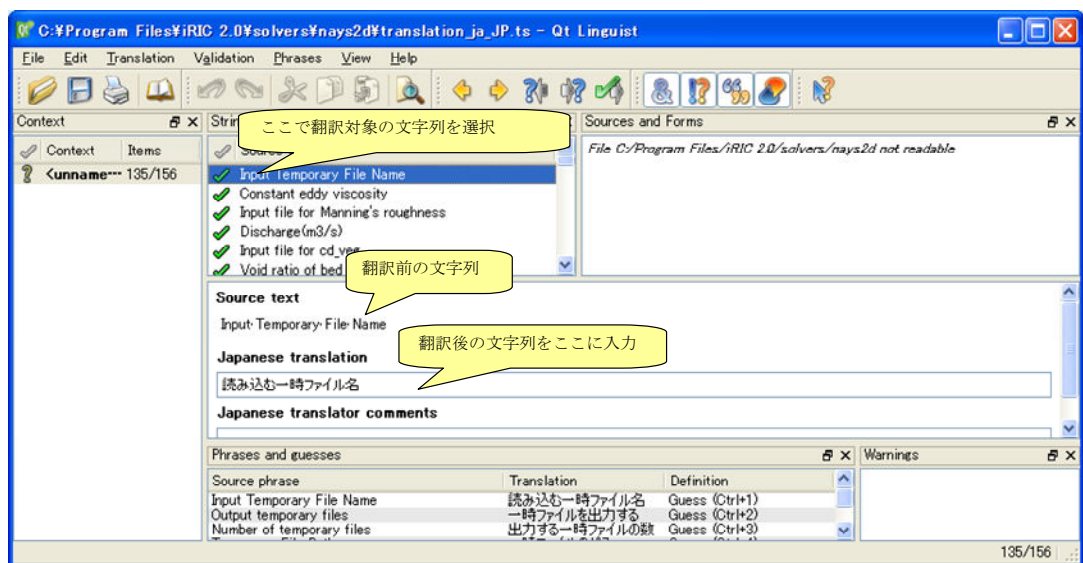


図 2-18 Qt Linguist 画面表示例

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 2-19、図 2-20 に示します。

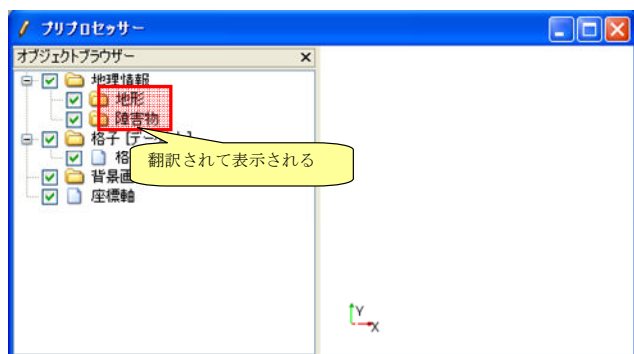


図 2-19 翻訳完了後のプリプロセッサ 表示例

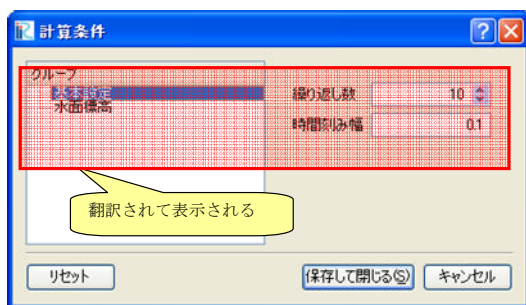


図 2-20 翻訳完了後の計算条件設定ダイアログ 表示例

2.6. 説明ファイルの作成

ソルバーの概要などについて説明するファイルを作成します。

README というファイル名のテキストファイルを、2.2 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 2-21 に示します。

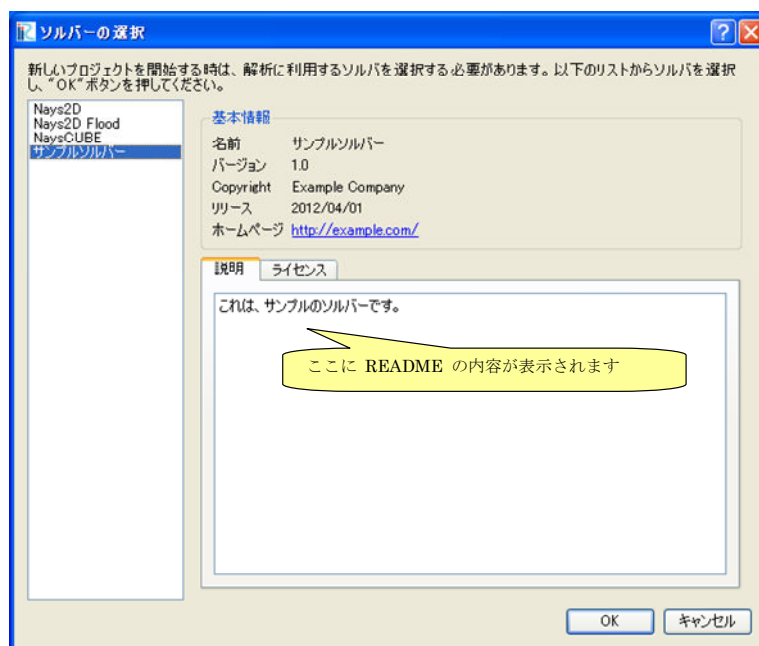


図 2-21 ソルバー選択ダイアログ 表示例

2.7. ライセンス情報ファイルの作成

ソルバーの利用ライセンスについて説明するファイルを作成します。

LICENSE というファイル名のテキストファイルを、2.2 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 2-22 に示します。

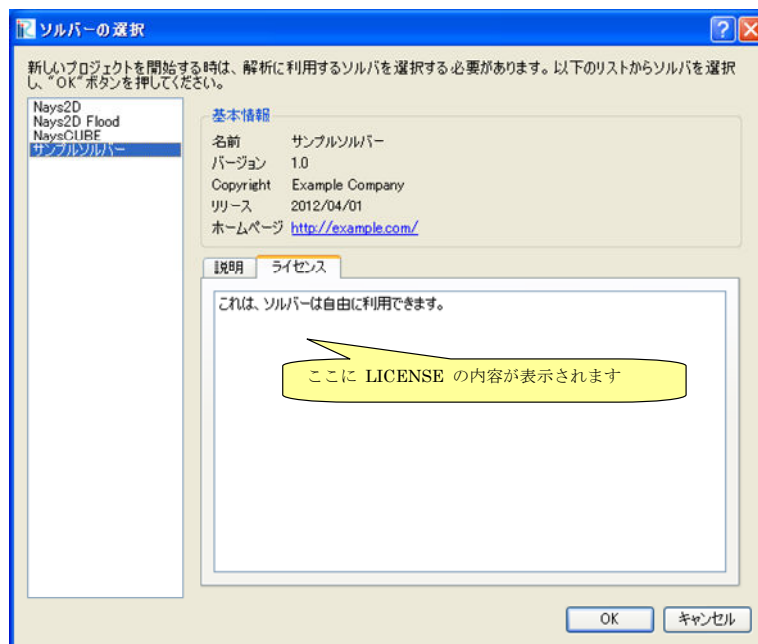


図 2-22 ソルバー選択ダイアログ 表示例

3. 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（2 章参照）。

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、2 章で使った関数とは別の関数を用います（6.4.1 参照）。複数 CGNS ファイル用の関数は、末尾が"_mul_f"で終わっており、ファイル ID を第一引数とします。また、計算結果読み込み用に既存の CGNS を開く際は、cg_iric_init_f の代わりに cg_iric_initread_f を用いて初期化を行います。複数の CGNS ファイルを扱ったソースコードの例（抜粋）を表 3-1 に示します。

表 3-1 複数 CGNS ファイルを扱ったソースコード（抜粋）

(前略)

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

(略)

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

(略)

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

(略)

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

(略)

```
! 計算結果の分析等
```

(略)

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

(略)

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

(後略)

既存の CGNS の計算結果をもとに、「魚の生息しやすさ」を算出するソルバーのソースコードを表 3-2 に示します。

表 3-2 既存の CGNS ファイルを読み込み、分析するソルバーのソースコード

```

program SampleProgram2
  implicit none
  include 'cgnslib_f.h'

  integer icount
  character(len=300) cgnsfile

  integer:: fin1, fin2, ier, istatus

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

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

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

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

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

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

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

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

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

```

```

! 格子を読み込む
call cg_irc_gotogridcoord2d_mul_f(fin2, isize, jsize, ier)
allocate(grid_x(isize, jsize), grid_y(isize, jsize))
call cg_irc_getgridcoord2d_mul_f(fin2, grid_x, grid_y, ier)

! 読み込んだ格子を cgnsfile に出力する
call cg_irc_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_irc_read_sol_count_mul_f(fin2, solcount, ier)

do solid = 1, solcount
    ! 計算結果を読み込み
    call cg_irc_read_sol_time_mul_f(fin2, solid, t, ier)
    call cg_irc_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_irc_write_sol_time_mul_f(fin1, t, ier)
    call cg_irc_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

```

4. 格子生成プログラムの開発手順

4.1. 概要

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

iRIC 上で動作する格子生成プログラムを開発するには、表 2-1 に示すようなファイルを作成、配置する必要があります。

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

表 4-1 格子生成プログラム関連ファイル、フォルダー一覧

ファイル名、フォルダ名	説明	参照
iRIC 2.0	iRIC 2.0 のインストールフォルダ (例: C:\Program Files\iRIC 2.0)	
gridcreators	格子生成プログラムを格納するフォルダ	
(格子生成プログラムフォルダ)	格子生成プログラムごとにフォルダを作成する。フォルダ名は任意。	4.2 節
definition.xml	格子生成プログラム定義ファイル。英語で記述する。	4.3 節
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。	4.4 節
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。	4.5 節
README	格子生成プログラムの説明ファイル	4.6 節

各ファイルの概要は以下の通りです。

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。

格子生成プログラム

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

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

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

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。

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。

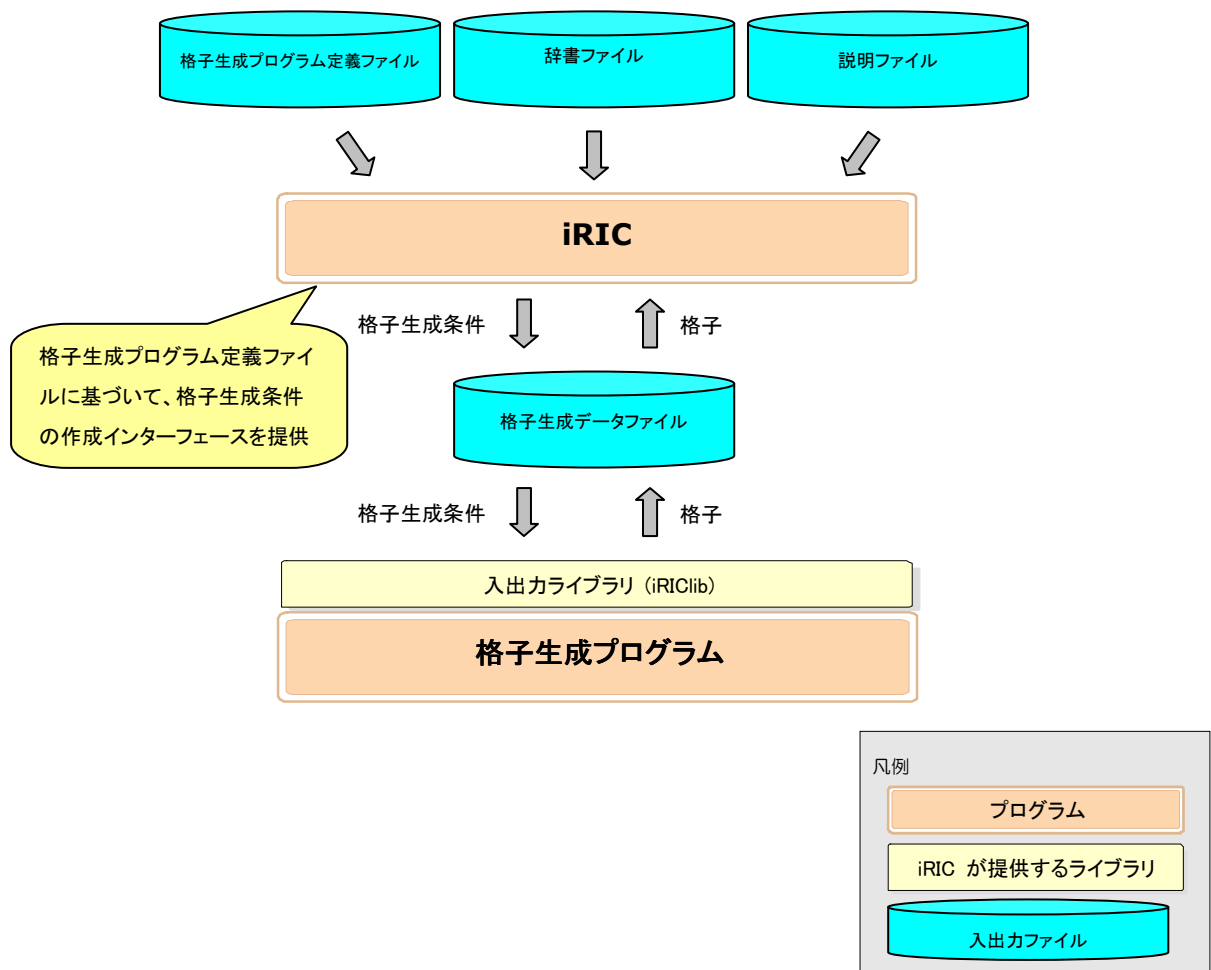


図 4-1 iRIC、格子生成プログラム、関連ファイルの関係図

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（デフォルトでは “C:\Program Files\iRIC 2.0”）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 5.5 を参照してください。

この節では、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

格子生成プログラムの基本情報を作成します。表 4-3 に示すようなファイルを作り、4.2 で作成した“example”フォルダの下に“definition.xml”の名前で保存します。

表 4-3 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、定義ファイルの構造は表 4-4 に示すようになっています。

表 4-4 格子生成プログラム定義ファイルの構造

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。

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 4-2 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 4-3 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画面の表示例を図 4-4 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。

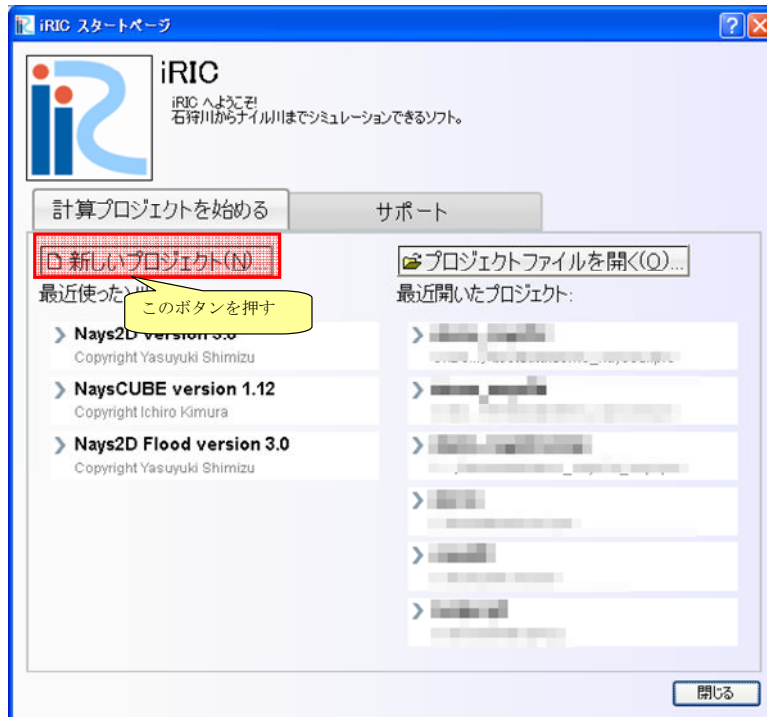


図 4-2 iRIC のスタートダイアログ

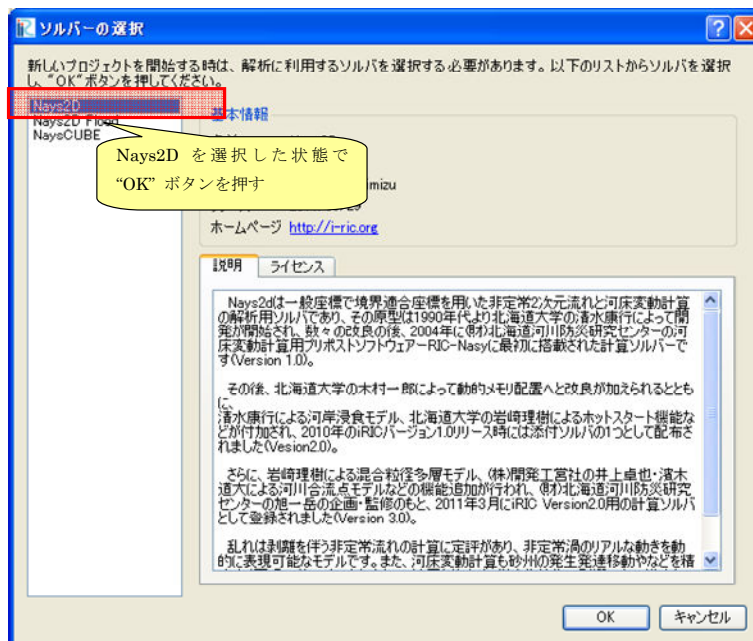


図 4-3 ソルバー選択ダイアログ

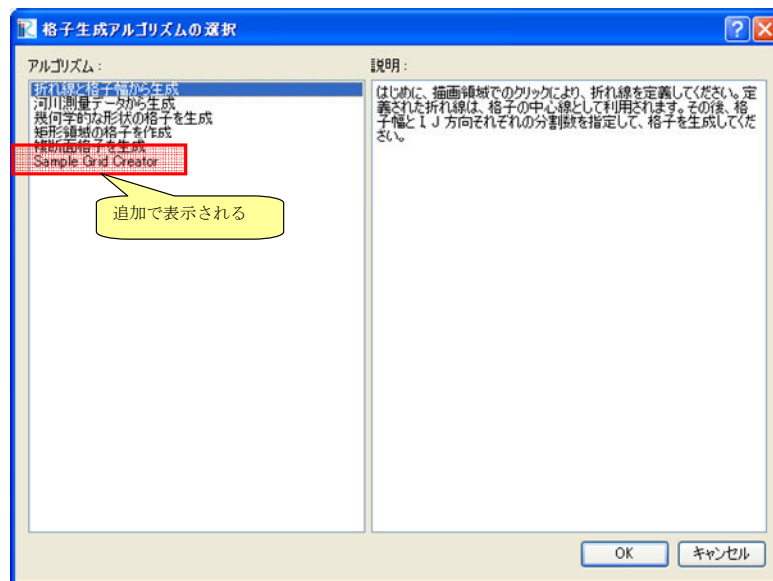


図 4-4 格子生成アルゴリズム選択画面

4.3.2. 格子生成条件の定義

格子生成条件を定義します。格子生成条件は、定義ファイルの `GridGeneratingCondition` 要素で定義します。2.3.1 で作成した定義ファイルに追記し、表 2-5 に示すようなファイルにし、保存します。追記した部分を太字で示しました。

表 4-5 計算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、定義ファイルの構造は 表 2-6 に示すようになっています。

表 4-6 格子生成プログラム定義ファイルの構造

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。

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、4.3.1 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 4-5 に示すダイアログが表示されます。表 4-5 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。

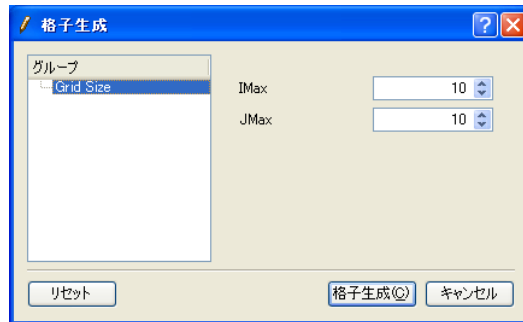


図 4-5 格子生成ダイアログ 表示例

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、表 4-7 に示します。追記した部分を太字で示しました。

表 4-7 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 表 4-8 に示す通りです。

表 4-8 格子生成プログラム定義ファイルの構造

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

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 4-6 に示します。

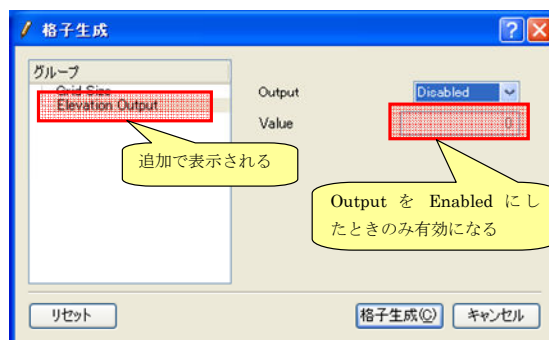


図 4-6 格子生成ダイアログ 表示例

格子生成条件の定義についてまとめると、以下の通りです。

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

きます。ダイアログのレイアウトのカスタマイズ方法については 5.3.3 を参照して下さい。

4.3.3. エラーコードの定義

格子生成プログラムで発生するエラーのコードと、対応するメッセージを定義します。エラーコードは、定義ファイルの **ErrorCode** 要素で定義します。4.3.2 で作成した定義ファイルに追記し、表 4-9 に示すようなファイルにし、保存します。追記した部分を太字で示しました。

表 4-9 エラーコードを追記した格子生成プログラム定義ファイルの例

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

この時点では、定義ファイルの構造は表 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。

表 4-10 格子生成プログラム定義ファイルの構造

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。

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 4.4.5 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 **FORTTRAN** 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、4.3 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 4-11 に示します。

表 4-11 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。

- 骨組みの作成
- 格子生成データファイルを開く処理、閉じる処理の記述
- 格子の出力処理の記述
- 格子生成条件の読み込み処理の記述
- 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。表 2-12 に示すソースコードを作成して、`sample.f90` という名前で保存します。この時点では、このプログラムは何もしていません。

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

表 4-12 サンプル格子生成プログラム ソースコード

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

end program SampleProgram
```

コンパイルが成功することを確認してください。

4.4.2. 格子生成データファイルを開く処理、閉じる処理の記述

格子生成データファイルを開く処理、閉じる処理を記述します。

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

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

処理を追記したソースコードを 表 4-13 に示します。追記した部分を太字で示します。

表 4-13 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
```

4.4.1 と同様に、ファイルのコンパイルを行います。問題なくコンパイルが成功することを確認してください。

この節で追加した関数の詳細については、6.3.2, 6.3.3, 6.3.14 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 表 4-14 に示します。追記した部分を太字で示します。

表 4-14 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 4.2 で作成したフォルダにコピーし、名前を 4.3.1 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 などと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 ”Nays2D” を指定して、新しいプロジェクトを開始し、4.3 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 4-7 に示します。

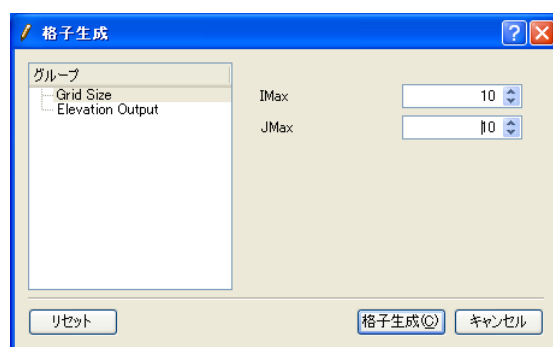


図 4-7 格子生成条件設定ダイアログ 表示例

“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 10×10 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 4-8 に示します。

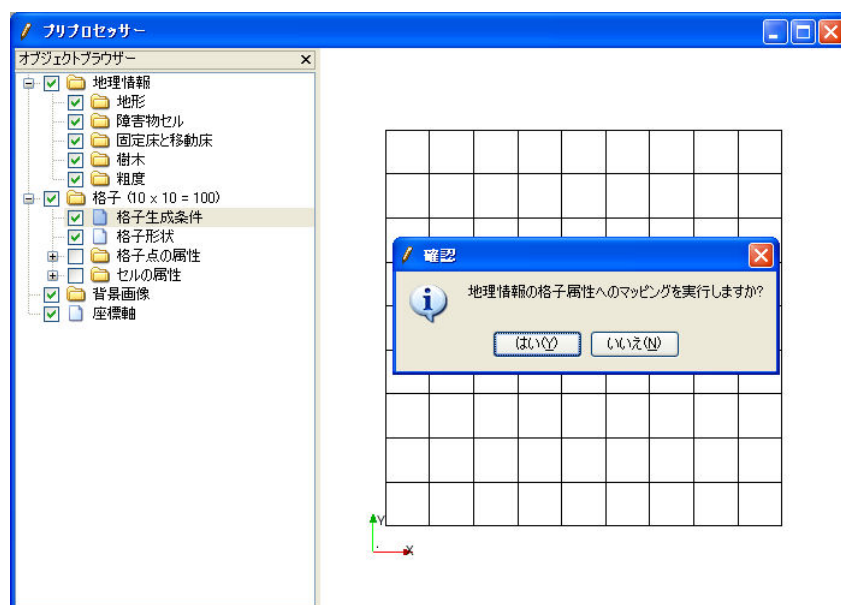


図 4-8 プリプロセッサ表示例

なお、この節で追加した格子出力用の関数の詳細については、6.3.7 を参照してください。
ただし 6.3.7 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、4.3 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 表 4-15 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、4.3.2 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、4.4.3 の時と同様の手順で格子を生成し、指定した通りの格子生成条件で格子が生成することを確認してください。

定義ファイルで定義する格子生成条件と、それを読み込むための iRIClib の関数の対応関係については、5.3.1 を参照してください。格子生成条件の読み込みに使う関数の詳細については、6.3.4 を参照してください。

表 4-15 格子生成条件の読み込み処理を追記したソースコード

```

program SampleProgram
  implicit none
  include 'cgnslib_f.h'

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

  character(200)::condFile

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

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

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

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

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

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

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

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

```

4.4.5. エラー処理の記述

格子生成条件に問題があった場合のエラー処理を記述します。

エラー処理を追記したソースコードを 表 4-16 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、4.4.4 の時と同様の手順で格子を生成し、`imax×jmax` が 100000 より大きくなる条件の時には、図 4-9 に示すようなダイアログが表示されることを確認してください。例えば、`IMax, JMax` にそれぞれ 10000 を指定してみてください。

エラー処理に使う関数の詳細については 6.3.13 を参照してください。

表 4-16 エラー処理を追記したソースコード (抜粋)

(前略)
<pre>! 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</pre>
<pre>! 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))</pre>
(後略)

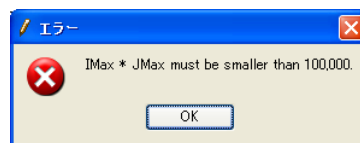


図 4-9 格子生成エラーダイアログ 表示例

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 4-10 ～ 図 4-12 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)

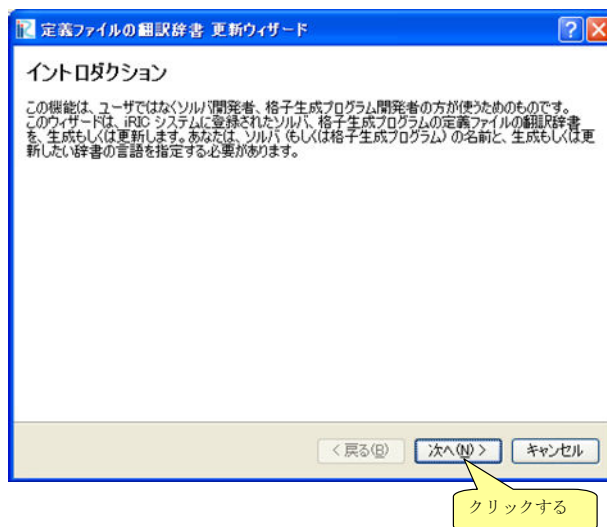


図 4-10 定義ファイルの翻訳辞書 更新ウィザード 表示例 (1 ページ目)

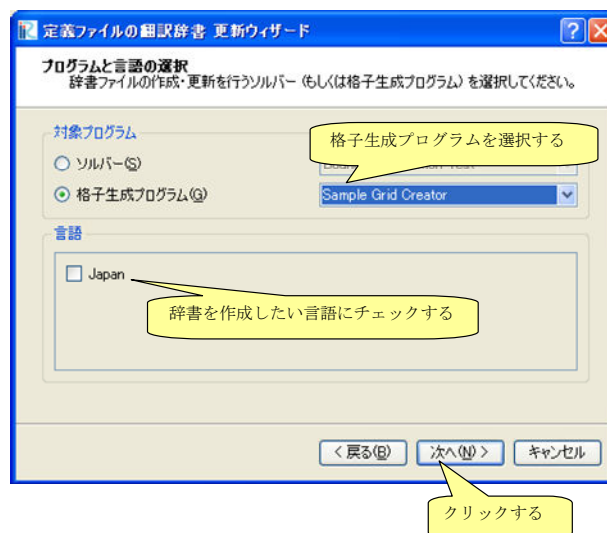


図 4-11 定義ファイルの翻訳辞書 更新ウィザード 表示例 (2 ページ目)

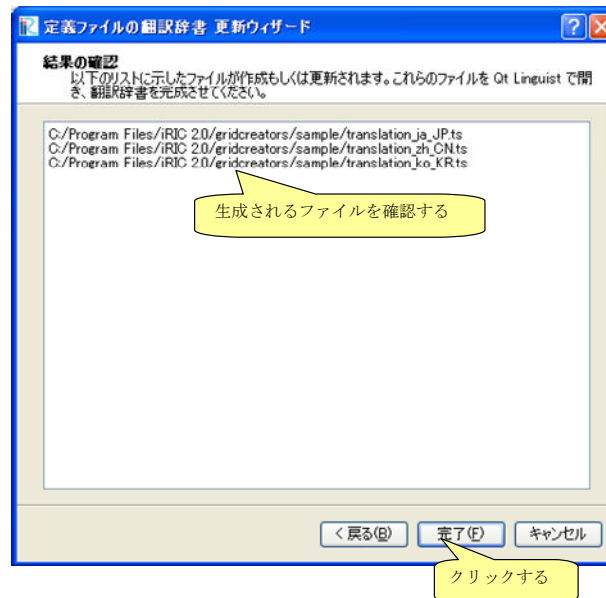


図 4-12 定義ファイルの翻訳辞書 更新ウィザード 表示例 (3 ページ目)

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、表 4-17、表 4-18 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 4-17 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>
```

表 4-18 格子生成プログラム定義ファイルの辞書ファイルの一部 (編集後)

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

なお、辞書ファイルは、Qt に付属する Qt Linguist を利用して編集することもできます。Qt Linguist の画面表示例を 図 4-13 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。

<http://qt.nokia.com/downloads/windows-cpp-vs2008>

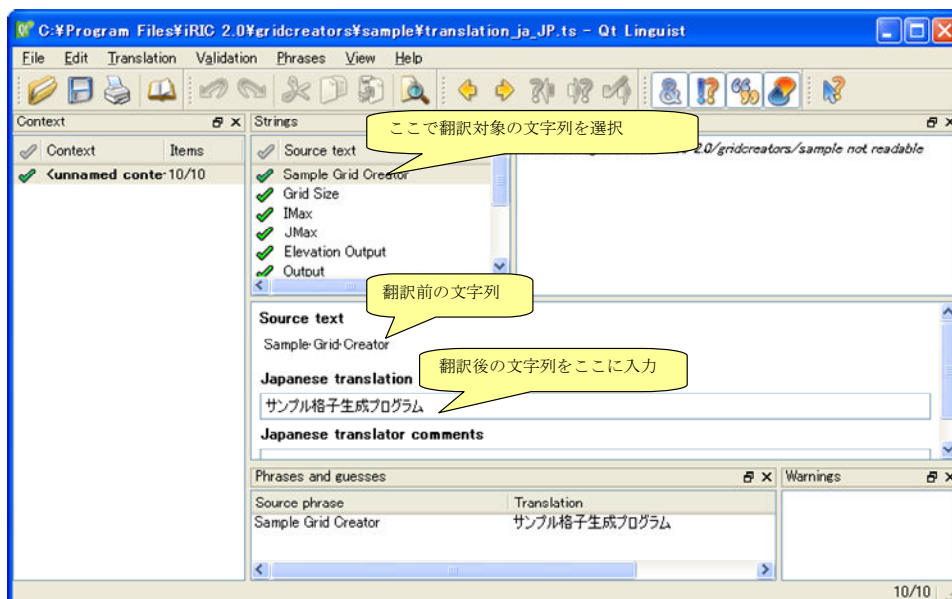


図 4-13 Qt Linguist 画面表示例

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 4-14 に示します。

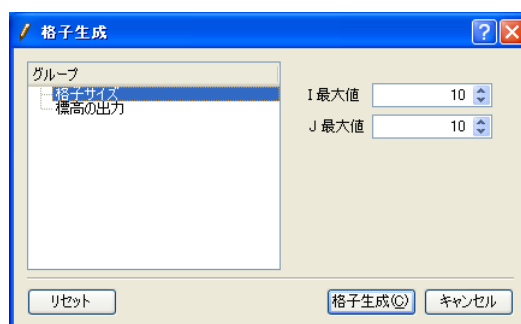


図 4-14 翻訳完了後の格子生成条件設定ダイアログ 表示例

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、4.2 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

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 4-15 に示します。

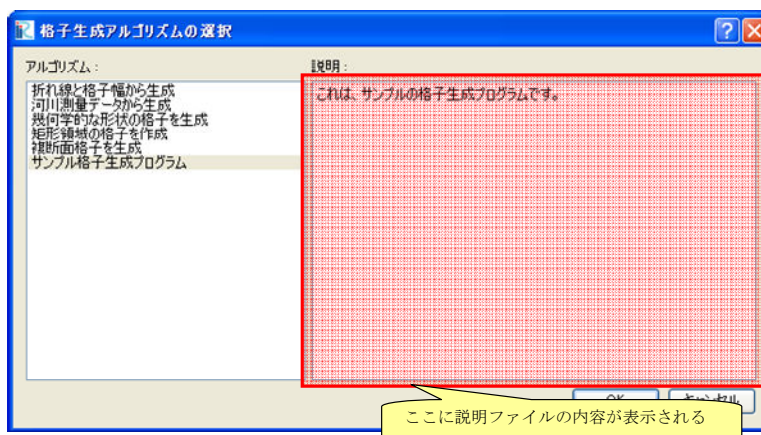


図 4-15 格子生成アルゴリズム選択ダイアログ 表示例

5. 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 表 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 表 5-2 にそれぞれ示します。

表 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		
...		
...		
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		

表 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

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	
...		

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、表 5-3 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。

表 5-3 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 5-1 に示すようなプリプロセッサが表示されます。

表 5-3 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>
```

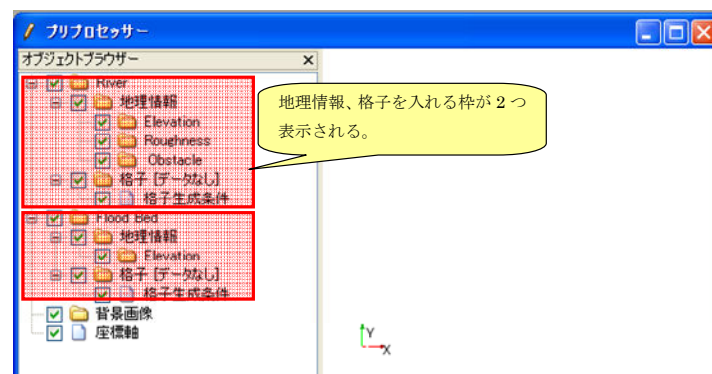


図 5-1 表 5-3 で示したソルバー定義ファイルを読み込んだ場合のプリプロセッサ 表示例

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、表 5-4 に示します。

表 5-4 格子生成プログラム定義ファイルの構造

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		
...		
...		

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 5-5 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 5.2 を参照してください。

表 5-5 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 5-6 に示します。この小節では、以下を示します。

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

ソルバー（もしくは格子生成プログラム）で値を読み込むための処理の記述例では、iRIClib の関数を使用しています。iRIClib の詳細は、6 章を参照して下さい。記述例は読み込みに関連する部分のみですので、プログラム全体の例は 2.3.4, 4.4 を参照してください。

表 5-6 計算条件、格子生成条件の項目の種類

No.	種類	説明	定義方法	ページ
1	文字列	文字列の値を入力。	valueType に “string” を指定	64
2	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	65
3	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	66
4	フォルダ名	フォルダ名を入力。	valueType に “foldername” を指定	67
5	整数	任意の整数値を入力。	valueType に “integer” を指定	68
6	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	69
7	実数	任意の実数値を入力。	valueType に “real” を指定	70
8	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	71
9	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	73

1) 文字列

表 5-7 文字列の条件の定義例

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



図 5-2 文字列の条件の表示例

表 5-8 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)
```

表 5-9 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)
```

2) ファイル名（読み込み用）

表 5-10 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>
```

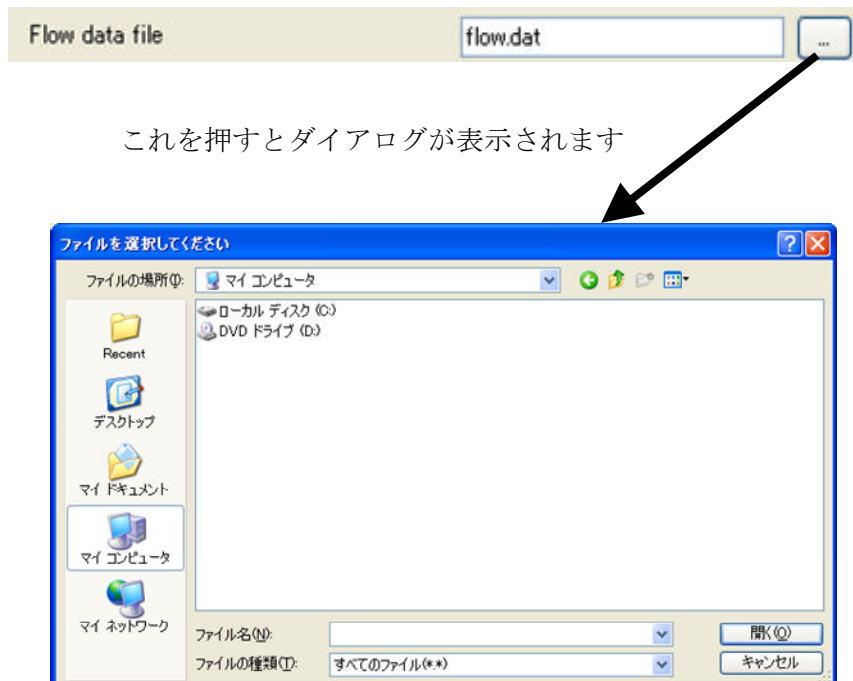


図 5-3 ファイル名（読み込み用）の条件の表示例

表 5-11 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_irc_read_string_f("flowdatafile", flowdatafile, ier)
```

表 5-12 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)

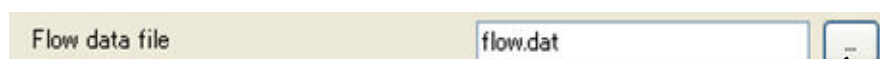
```
integer:: ier
character(200):: flowdatafile

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

3) ファイル名（書き込み用）

表 5-13 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>
```



これを押すとダイアログが表示されます

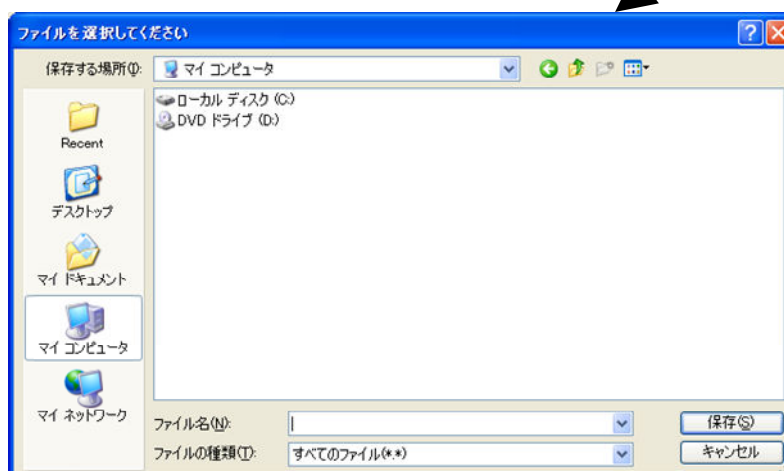


図 5-4 ファイル名（書き込み用）の条件の表示例

表 5-14 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)
```

表 5-15 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)
```

4) フォルダ名

表 5-16 フォルダ名の条件の定義例

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



これを押すとダイアログが表示されます



図 5-5 フォルダ名の条件の表示例

表 5-17 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)
```

表 5-18 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)
```


5) 整数

表 5-19 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 5-6 整数の条件の表示例

表 5-20 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)
```

表 5-21 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)
```

6) 整数（選択式）

表 5-22 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 5-7 整数（選択式）の条件の表示例

表 5-23 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)
```

表 5-24 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)
```

7) 実数

表 5-25 実数の条件の定義例

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

Gravity [m/s2]	<input type="text" value="9.8"/>
----------------	----------------------------------

図 5-8 実数の条件の表示例

表 5-26 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)
```

表 5-27 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)
```

8) 関数型

表 5-28 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>
```



これを押すとダイアログが表示されます

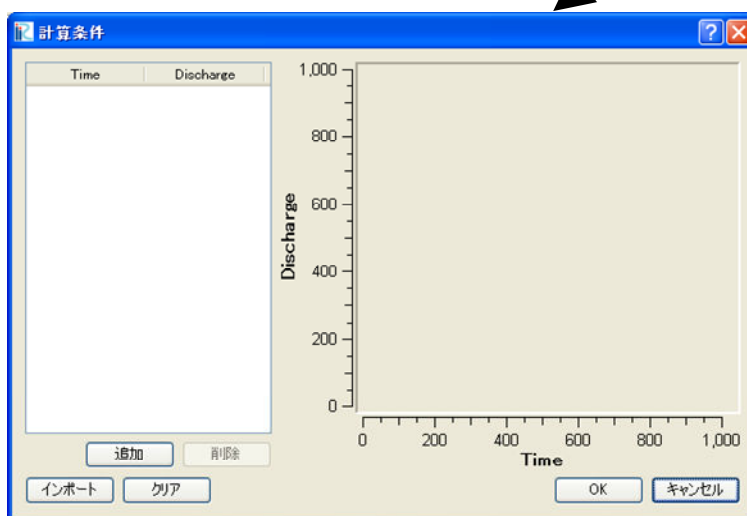


図 5-9 関数型の条件の表示例

表 5-29 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)
```

表 5-30 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)
```

9) 関数型 (複数の値)

表 5-31 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>
```



これを押すとダイアログが表示されます

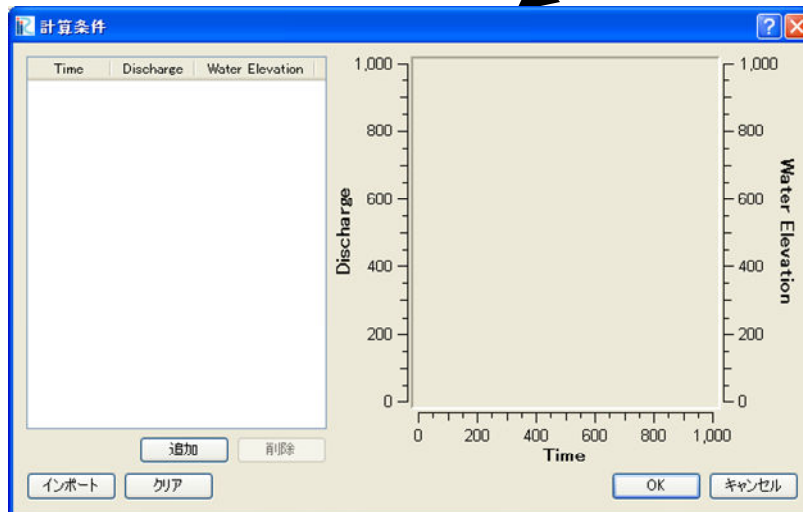


図 5-10 関数型 (複数の値) の条件の表示例

表 5-32 関数型 (複数の値) の条件を読み込むための処理の記述例
(計算条件・格子生成条件)

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

! サイズを調べる

```
call cg_iric_read_functionalsize_f("discharge", discharge_size, ier)
```

! メモリを確保

```
allocate(time_value(discharge_size))
```

```
allocate(discharge_value(discharge_size), elevation_value(discharge_size))
```

! 確保したメモリに値を読み込む

```
call cg_iric_read_functionalwithname_f("discharge", "time", time_value)
```

```
call cg_iric_read_functionalwithname_f("discharge", "discharge", discharge_value)
```

```
call cg_iric_read_functionalwithname_f("discharge", "elevation", elevation_value)
```

表 5-33 関数型（複数の値）の条件を読み込むための処理の記述例
(境界条件)

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

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

5.3.2. 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。

1) $\text{var1} = 1$

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

2) $(\text{var1} = 1) \cup (\text{var2} > 3)$

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

3) $((\text{var1} = 1) \cup (\text{var2} < 5)) \cap (\text{var3} = 100)$

```
<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

1) 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 表 5-34 に、ダイアログでの表示例を 図 5-11 にそれぞれ示します。

表 5-34 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>
  </Item>
</Tab>
```

Periodic boundary condition	Disabled
Water surface at downstream	Uniform flow
Constant value (m)	0
Slope for uniform flow	Calculated from geographic data
Slope value at downstream	0.001
Velocity at upstream	Uniform flow
Slope for uniform flow	Calculated from geographic data
Slope value at upstream	0.001

図 5-11 表 5-34 の定義に基づくダイアログの表示例

2) 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 表 5-35 に、ダイアログでの表示例を 図 5-12 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。

表 5-35 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>
    <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>
```

Periodic boundary condition Disabled

Water surface at downstream

Basic Setting Uniform flow

Constant value (m) 0

Slope for uniform flow Calculated from geographic data

Slope value at downstream 0.001

Velocity at upstream

Basic Setting Uniform flow

Slope for uniform flow Calculated from geographic data

Slope value at upstream 0.001

図 5-12 表 5-35 の定義に基づくダイアログの表示例

3) 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 表 5-36 に、ダイアログでの表示例を 図 5-13 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。

表 5-36 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>
```

Diameter of uniform bed material (mm)

Bed roughness ▼

Manning's roughness parameter

Low water channel	Flood channel	Fixed bed
<input type="text" value="0.01"/>	<input type="text" value="0.01"/>	<input type="text" value="0.01"/>

Input file for Manning's roughness

図 5-13 表 5-36 の定義に基づく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。

表 5-37 BoundaryCondition の内容

項目	名前	必須	型	値の意味
属性	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	○	選択	定義位置。以下のいずれか <ul style="list-style-type: none">• node (格子点)• cell (セル)• edge (格子の辺)
要素	Item	○	要素	要素の定義 複数の要素をもてる。

5.4.2. CalculationCondition

計算条件の情報を保持します。

表 5-38 CalculationCondition の内容

項目	名前	必須	型	値の意味
要素	Tab		タブ要素	計算条件入力ダイアログの各ページの情報を持つ要素 複数の要素をもてる。

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。

表 5-39 Condition の内容

項目	名前	必須	型	値の意味
属性	conditionType	○	選択	以下のいずれか <ul style="list-style-type: none">• 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• or• not
	target	△	文字列	比較対象の計算条件の名前。 conditionType が以下以外の場合に指定する。 <ul style="list-style-type: none">• and• or• not
	value	△	文字列	比較対象の値。 conditionType が以下以外の場合に指定する。 <ul style="list-style-type: none">• and• or• not
要素	Condition		要素	AND, OR, NOT の演算子を適用する対象の条件。 conditionType が以下の場合のみ指定する。 <ul style="list-style-type: none">• and• or• not

Condition 要素の定義例は、5.3.2 を参照してください。

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

計算条件、もしくは格子生成条件の定義情報を保持します。

表 5-40 Definition の内容

項目	名前	必須	型	値の意味
属性	valueType	○	選択	以下のいずれか <ul style="list-style-type: none"> integer (整数) real (実数) string (文字列) filename (ファイル名) filename_all (ファイル名。存在しないファイルも選択可能) foldername (フォルダ名) functional (関数型)
	default		文字列	valueType で指定した型として認識できる任意の文字列。 例えば、 valueType="integer" の場合、“0”、“2”などと指定する。
要素	Enumeration		要素	値を、Enumeration で指定する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		要素	この計算条件（もしくは格子生成条件）が、有効になる時の条件を指定する。常に有効な場合は不要。

Definition 要素の定義例は、5.3.1 を参照してください。

5.4.5. Definition (格子属性の定義)

計算格子に定義する属性の定義情報を保持します。

表 5-41 Definition の内容

項目	名前	必須	型	値の意味
属性	valueType	○	選択	以下のいずれか <ul style="list-style-type: none">integer (整数)real (実数)complex (複合型)
	position	○	選択	属性の定義位置。以下のいずれか <ul style="list-style-type: none">node (格子点)cell (セル)
	default		文字列	valueType で指定した型として認識できる任意の文字列。 例えば、valueType="integer" の場合、“0”, “2” などと指定する。 “min”, “max” と指定すると、地理情報が存在しない領域では、それぞれ読み込まれた地理情報の最小値、最大値が利用される。
要素	Dimension		要素	次元 (例: 時刻) を追加したい場合に指定する。複数定義することができる。
	Enumeration		要素	値を、Enumeration で指定する選択肢からのみ選べるようにしたい場合に指定する。
	Item		要素	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。

表 5-42 Dimension の内容

項目	名前	必須	型	値の意味
属性	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	○	選択	以下のいずれか <ul style="list-style-type: none">integer (整数)real (実数)

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。

表 5-43 Enumeration の内容

項目	名前	必須	型	値の意味
要素	value	○	任意	caption に対応する値を表す文字列。例えば Definition の valueType が “integer” の場合、“0”, “1” などと指定する。
	caption	○	文字列	表示する文字列

Enumeration の定義例は、5.3.1.6) を参照してください。

5.4.8. ErrorCodes

エラーコードのリストを保持します。

表 5-44 ErrorCodes の内容

項目	名前	必須	型	値の意味
要素	ErrorCode		要素	エラーコード。

5.4.9. ErrorCode

エラーコードの定義情報を保持します。

表 5-45 ErrorCode の内容

項目	名前	必須	型	値の意味
属性	caption	○	文字列	表示する文字列
	value	○	整数	エラーコード

5.4.10. 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。

表 5-46 GroupBox の内容

項目	名前	必須	型	値の意味
属性	caption	○	文字列	表示する文字列
要素	VBoxLayout など		要素	

GroupBox 要素の定義例は 5.3.3.2) を参照してください。

5.4.11. GridGeneratingCondition

格子生成条件の情報を保持します。

表 5-47 4.4.8. GridGeneratingCondition の内容

項目	名前	必須	型	値の意味
要素	Tab		タブ要素	格子生成条件入力ダイアログの各ページの情報を持つ要素 複数の要素をもてる。

5.4.12. GridGeneratorDefinition

格子生成プログラムの定義情報を保持します。

表 5-48 4.4.10. GridGeneratorDefinition 要素の内容

項目	名前	必須	型	値の意味
属性	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（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	○	選択	生成する講師の種類。以下を指定する。 <ul style="list-style-type: none"> structured2d (2 次元構造格子) unstructured2d (2 次元非構造格子)
要素	GridGeneratingCondition	○	格子生成条件	格子生成条件
	ErrorCodes		エラーコードの リスト	

5.4.13. 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。

表 5-49 GridLayout の内容

項目	名前	必須	型	値の意味
要素	VBoxLayout など		要素	

GridLayout の定義例は 5.3.3.3) を参照してください。

5.4.14. GridTypes

入力格子の定義情報のリストを保持します。

表 5-50 GridTypes の内容

項目	名前	必須	型	値の意味
要素	GridType		要素	格子の種類。

5.4.15. GridType

入力格子の定義情報を保持します。

表 5-51 GridType の内容

項目	名前	必須	型	値の意味
属性	gridtype	○	選択	以下のいずれかを指定する。 <ul style="list-style-type: none">1d (1 次元格子)1.5d (1.5 次元格子)1.5d_withcrosssection (横断面情報を持つ 1.5 次元格子)structured2d (2 次元構造格子)unstructured2d (2 次元非構造格子)
	multiple	○	真偽	以下のいずれか <ul style="list-style-type: none">true (複数の格子を扱える)false (1 つの格子のみ扱える)
要素	GridRelatedCondition	○	要素	入力格子に定義する属性の情報

5.4.16.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。

表 5-52 HBoxLayout の内容

項目	名前	必須	型	値の意味
要素	VBoxLayout など		要素	

HBoxLayout 要素は、その子要素に指定した要素を水平方向に並べます。子要素には Label, Item, GroupBox, HBoxLayout, VBoxLayout, GridLayout を指定できます。

5.4.17. 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。

表 5-53 Item の内容

項目	名前	必須	型	値の意味
属性	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）。
要素	Definition	○	要素	要素の定義

Item 要素の定義例は、5.3.1 を参照してください。

5.4.18. 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。

表 5-54 Label の内容

項目	名前	必須	型	値の意味
属性	caption	○	文字列	表示する文字列

Label の定義例は、5.3.3.3) を参照してください。

5.4.19. Param

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

表 5-55 Param の内容

項目	名前	必須	型	値の意味
属性	caption	○	文字列	表示する文字列
	valueType	○	選択	以下のいずれかを指定する。 <ul style="list-style-type: none">integer (整数)real (実数)
	axislog		真偽	以下のいずれか <ul style="list-style-type: none">true (横軸を対数軸にする)false (横軸を通常の軸にする)

Param の定義例は、5.3.1.8)を参照してください。

5.4.20. SolverDefinition

ソルバーの定義情報を保持します。

表 5-56 SolverDefinition 要素の内容

項目	名前	必須	型	値の意味
属性	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: Solver.exe）
	iterationtype	○	選択	以下のいずれかを指定 <ul style="list-style-type: none"> 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。収束計算をするソルバーなどで利用）
要素	gridtype	○	選択	1 種類の入力格子を利用する場合のみ指定する。 以下のいずれかを指定する。 <ul style="list-style-type: none"> 1d（1 次元格子） 1.5d（1.5 次元格子） 1.5d_withcrosssection（横断面情報を持つ 1.5 次元格子） structured2d（2 次元構造格子） unstructured2d（2 次元非構造格子）
	multiple	○	真偽	1 種類の入力格子を利用する場合のみ指定する。 以下のいずれかを指定する。 <ul style="list-style-type: none"> true（複数の格子を扱える） false（1 つの格子のみ扱える）
	CalculationCondition	○	計算条件要素	計算条件
	GridRelatedCondition		格子属性	1 種類の入力格子を利用する場合のみ定義する。
	GridTypes		格子種類	2 種類以上の入力格子を利用する場合のみ定義する。

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、5.5 を参照してください。

5.4.21. 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。

表 5-57 Tab の内容

項目	名前	必須	型	値の意味・備考
属性	name	○	文字列	識別名（英数字のみ）
	caption	○	文字列	名前（任意の文字を利用可能）
要素	Item, GroupBox など	○	要素	このページに表示する計算条件（もしくは格子生成条件）

Tab 要素の定義例は 5.3.3 を参照してください。

5.4.22. 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。

表 5-58 Value の内容

項目	名前	必須	型	値の意味
属性	caption	○	文字列	表示する文字列
	valueType	○	選択	以下のいずれかを指定する。 <ul style="list-style-type: none">integer (整数)real (実数)
	name		文字列	識別名（英数字のみ）。複数の値を持つ関数型の条件の場合のみ指定する。
	axis		選択	設定ダイアログのグラフでの Y 軸の表示方法を指定。以下のいずれかを指定する。デフォルトでは、1 つ目の値は left, 2 つめ以降の値は right となる。 <ul style="list-style-type: none">left (左)right (右)
	axislog		真偽	以下のいずれか <ul style="list-style-type: none">true (横軸を対数軸にする)false (横軸を通常の軸にする)
	axisreverse		真偽	以下のいずれか <ul style="list-style-type: none">true (Y 軸を上下逆転する)false (Y 軸を通常の軸にする)

	step		真偽	以下のいずれか <ul style="list-style-type: none"> • true (棒グラフとして表示する) • false (折れ線グラフとして表示する)
	hide		真偽	設定ダイアログのグラフで隠すかどうかを指定。 true を指定すると、グラフ上に描画しない。

Value の定義例は、5.3.1.8), 5.3.1.9) を参照してください。

5.4.23. VBoxLayout

計算条件（もしくは格子生成条件）設定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。

表 5-59 VBoxLayout の内容

項目	名前	必須	型	値の意味
要素	VBoxLayout など		要素	

VBoxLayout 要素は、その子要素に指定した要素を水平方向に並べます。子要素には Label, Item, GroupBox, HBoxLayout, VBoxLayout, GridLayout を指定できます。

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 5-60 のポリシーに従って変更して下さい。なお、バージョン番号の構成要素については 図 5-14 に示します。

表 5-60 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は 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

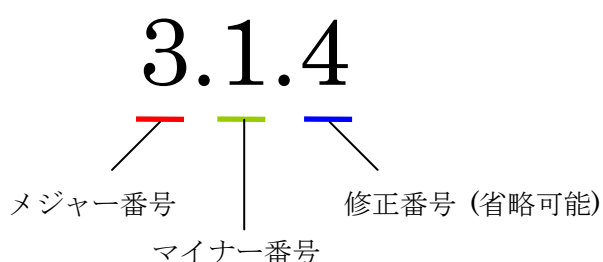


図 5-14 バージョン番号の構成要素

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 表 5-61 に示します。

表 5-61 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

		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	×	×	○	○

バージョン番号の増やし方の基本方針は 表 5-60 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。**Sample Solver** というソルバーのバージョン 1.1、2.0 を混在させる場合のフォルダ構成の例を 表 5-62 に示します。

表 5-62 複数のバージョンのソルバーを混在させる場合のフォルダ構成例

ファイル名、フォルダ名	説明	参照
iRIC 2.0	iRIC 2.0 のインストールフォルダ (例: C:\Program Files\iRIC 2.0)	
solvers	ソルバーを格納するフォルダ	
sample_11	Sample Solver 1.1 の関連ファイルを格納するフォルダ	
definition.xml	Sample Solver 1.1 の定義ファイル。SolverDefinition 要素の name 属性に “sample”、version 属性に “1.1” を指定する。	
(他のファイルは省略)		
sample_20	Sample Solver 2.0 の関連ファイルを格納するフォルダ	
definition.xml	Sample Solver 2.0 の定義ファイル。SolverDefinition 要素の name 属性に “sample”、version 属性に “2.0” を指定する。	
(他のファイルは省略)		

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 表 5-63 に示します。

表 5-63 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 表 5-64 に示します。

表 5-64 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、表 5-65, 表 5-66 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。

表 5-65 子要素を持たない要素の記述例

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

表 5-66 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。

表 5-67, 表 5-68, 表 5-69 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。

表 5-67 要素の記述例

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

表 5-68 要素の記述例

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

表 5-69 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。表 5-70 にコメントの記述例を示します。

表 5-70 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>
```

6. 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 (7.4 節参照)。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. この章の読み方

6.3 節で、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、6.3 節を読んで 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 6.4 節を参照してください。

6.3. 概要

この節では、iRIClib の概要について説明します。

6.3.1. 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。各処理に必要な関数とその使用例は、表 6-1、表 6-2 に示したページを参照して下さい。

表 6-1 ソルバーの入出力処理

処理の内容	ページ
CGNS ファイルを開く	101
内部変数の初期化	101
計算条件の読み込み	102
計算格子の読み込み	104
境界条件の読み込み	107
地形データの読み込み（必要な場合のみ）	109
計算格子の出力（格子の生成、再分割を行う場合のみ）	109
時刻（もしくはループ回数）の出力	115
計算格子の出力（移動格子の場合のみ）	116
計算結果の出力	118
CGNS ファイルを閉じる	122

} 複数回
繰り返す

表 6-2 格子生成プログラムの入出力処理

処理の内容	ページ
CGNS ファイルを開く	101
内部変数の初期化	101
格子生成条件の読み込み	102
エラーコードの出力	122
格子の出力	109
CGNS ファイルを閉じる	122

6.3.2. CGNS ファイルを開く

【説明】

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。

【利用する関数】

関数	備考
<code>cg_open_f</code>	CGNS ファイルを開く。

6.3.3. 内部変数の初期化

【説明】

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。

【利用する関数】

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

6.3.4. 計算条件（もしくは格子生成条件）の読み込み

【説明】

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。

【利用する関数】

関数	備考
cg_irc_read_integer_f	整数の条件を読み込む
cg_irc_read_real_f	倍精度実数の条件を読み込む
cg_irc_read_realsingle_f	単精度実数の条件を読み込む
cg_irc_read_string_f	文字列の条件を読み込む
cg_irc_read_functionalsize_f	関数型の条件のサイズを調べる
cg_irc_read_functional_f	倍精度実数の関数型の条件を読み込む
cg_irc_read_functional_realsingle_f	単精度実数の関数型の条件を読み込む
cg_irc_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表 6-3 に示します。

表 6-3 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_irc_init_f(fin, ier)
  if (ier /=0) STOP "*** Initialize error of CGNS file ***"

  call cg_irc_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表 6-4 に示します。

表 6-4 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、5.3.1 を参照してください。

6.3.5. 計算格子の読み込み

【説明】

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。

【利用する関数】

関数	備考
cg_iris_gotogridcoord2d_f	2次元構造格子を読み込む準備をする
cg_iris_getgridcoord2d_f	2次元構造格子を読み込む
cg_iris_gotogridcoord3d_f	3次元構造格子を読み込む準備をする
cg_iris_getgridcoord3d_f	3次元構造格子を読み込む
cg_iris_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_iris_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_iris_read_grid_integer_cell_f	セルで定義された整数の属性を読み込む
cg_iris_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_iris_read_complex_count_f	複合型の属性のグループの数を読み込む
cg_iris_read_complex_integer_f	複合型の属性の整数の条件を読み込む
cg_iris_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_iris_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_iris_read_complex_string_f	複合型の属性の文字列の条件を読み込む
cg_iris_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_iris_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_iris_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_iris_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_iris_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_iris_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_iris_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_iris_read_grid_functionaltime_f	次元「時刻」(Time)の値を読み込む
cg_iris_read_grid_functionaldimensionsize_f	次元の数を調べる
cg_iris_read_grid_functionaldimension_integer_f	整数の次元の値を読み込む
cg_iris_read_grid_functionaldimension_real_f	倍精度実数の次元の値を読み込む
cg_iris_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_irc_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_irc_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_irc_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_irc_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 表 6-5 に示します。

表 6-5 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_irc_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_irc_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_irc_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_irc_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)
    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_irc_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_irc_read_grid_functionaltimesize_f('Rain', rain_timesize, ier);
! Rain の時刻を読み込むメモリを確保。
allocate(rain_time(rain_timesize))
! Rain の時刻を読み込み
call cg_irc_read_grid_functionaltime_f('Rain', rain_time, ier);

! 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_irc_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。

6.3.6. 境界条件の読み込み

【説明】

CGNS ファイルから、境界条件を読み込みます。

【利用する関数】

関数	備考
cg_irc_read_bc_count_f	境界条件の数を取得する
cg_irc_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_irc_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_irc_read_bc_integer_f	整数型境界条件の値を取得する
cg_irc_read_bc_real_f	実数(倍精度)境界条件の値を取得する
cg_irc_read_bc_realsingle_f	実数(単精度)境界条件の値を取得する
cg_irc_read_bc_string_f	文字列型境界条件の値を取得する
cg_irc_read_bc_functionalsize_f	関数型境界条件のサイズを取得する
cg_irc_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_irc_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 表 6-5 に示します。この例では、流入口（Inflow）の数を `cg_irc_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_irc_read_bc_string_f` で読み込めます。

表 6-6 境界条件を読み込む処理の記述例

```
program Sample8
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, isize, jsize, ksize, i, j, k, aret
  integer:: condid, indexid
  integer:: condcount, indexlenmax, funcsizemax
  integer:: tmpflen
  integer, dimension(:), allocatable:: condindexlen
  integer, dimension(:, :, :), allocatable:: condindices
  integer, dimension(:), allocatable:: intparam
  double precision, dimension(:), allocatable:: realparam
  character(len=200), dimension(:), allocatable:: stringparam
  character(len=200):: tmpstr
  integer, dimension(:), allocatable:: func_size
  double precision, dimension(:, :, :), allocatable:: func_param;
  double precision, dimension(:, :, :), allocatable:: func_value;
```



```

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

! 流入口の数取得する
call cg_irc_read_bc_count_f('inflow', condcount)
! 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。
allocate(condindexlen(condcount), intparam(condcount), realparam(condcount))
allocate(stringparam(condcount), func_size(condcount))
print *, 'condcount ', condcount

! 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
indexlenmax = 0
funcsizemax = 0
do condid = 1, condcount
  call cg_irc_read_bc_indicessize_f('inflow', condid, condindexlen(condid), ier)
  if (indexlenmax < condindexlen(condid)) then
    indexlenmax = condindexlen(condid)
  end if
  call cg_irc_read_bc_functionalsize_f('inflow', condid, 'funcparam', func_size(condid), ier);
  if (funcsizemax < func_size(condid)) then
    funcsizemax = func_size(condid)
  end if
end do

! 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
allocate(condindices(condcount, 2, indexlenmax))
allocate(func_param(condcount, funcsizemax), func_value(condcount, funcsizemax))
! インデックスと、境界条件 を読み込み
do condid = 1, condcount
  call cg_irc_read_bc_indices_f('inflow', condid, condindices(condid:condid,:,:), ier)
  call cg_irc_read_bc_integer_f('inflow', condid, 'intparam', intparam(condid:condid), ier)
  call cg_irc_read_bc_real_f('inflow', condid, 'realparam', realparam(condid:condid), ier)
  call cg_irc_read_bc_string_f('inflow', condid, 'stringparam', tmpstr, ier)
  stringparam(condid) = tmpstr
  call cg_irc_read_bc_functional_f('inflow', condid, 'funcparam', func_param(condid:condid,:),
func_value(condid:condid,:), ier)
end do

! 読み込まれた境界条件を表示
do condid = 1, condcount
  do indexid = 1, condindexlen(condid)
    print *, 'condindices ', condindices(condid:condid,:,indexid:indexid)
  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

```

6.3.7. 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。

【利用する関数】

関数	備考
cg_irc_read_geo_count_f	地形データの数を返す
cg_irc_read_geo_filename_f	地形データのファイル名と種類を返す
irc_geo_polygon_open_f	ポリゴンファイルを開く
irc_geo_polygon_read_integervalue_f	ポリゴンの値を整数で返す
irc_geo_polygon_read_realvalue_f	ポリゴンの値を実数で返す
irc_geo_polygon_read_pointcount_f	ポリゴンの頂点の数を返す
irc_geo_polygon_read_points_f	ポリゴンの頂点の座標を返す
irc_geo_polygon_read_holecount_f	ポリゴンに開いた穴の数を返す
irc_geo_polygon_read_holepointcount_f	ポリゴンの穴の頂点の数を返す
irc_geo_polygon_read_holepoints_f	ポリゴンの穴の頂点の座標を返す
irc_geo_polygon_close_f	ポリゴンファイルを閉じる
irc_geo_riversurvey_open_f	河川測量データを開く
irc_geo_riversurvey_read_count_f	河川横断線の数を返す
irc_geo_riversurvey_read_position_f	横断線の中心点の座標を返す
irc_geo_riversurvey_read_direction_f	横断線の向きを返す
irc_geo_riversurvey_read_name_f	横断線の名前を文字列として返す
irc_geo_riversurvey_read_realname_f	横断線の名前を実数値として返す
irc_geo_riversurvey_read_leftshift_f	横断線の標高データのシフト量を返す
irc_geo_riversurvey_read_altitudecount_f	横断線の標高データの数を返す
irc_geo_riversurvey_read_altitudes_f	横断線の標高データを返す
irc_geo_riversurvey_read_fixedpointl_f	横断線の左岸延長線のデータを返す
irc_geo_riversurvey_read_fixedpointtr_f	横断線の右岸延長線のデータを返す
irc_geo_riversurvey_read_watersurfaceelevation_f	横断線での水面標高のデータを返す
irc_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表 6-7 に、河川測量データを読み込む処理の記述例を表 6-8 にそれぞれ示します。

表 6-7 ポリゴンを読み込む処理の記述例

```

program TestPolygon
  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:: 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))
        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

```

表 6-8 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 ****"

! 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_RIVERSURVEY) then
        call iric_geo_riversurvey_open_f(filename, rsid, ier)
        call iric_geo_riversurvey_read_count_f(rsid, xsec_count, ier)
        do xsec_id = 1, xsec_count
            call iric_geo_riversurvey_read_name_f(rsid, xsec_id, xsec_name, ier)
            print *, 'xsec ', xsec_name
            call iric_geo_riversurvey_read_position_f(rsid, xsec_id, xsec_x, xsec_y, ier)
            print *, 'position: ', xsec_x, xsec_y
            call iric_geo_riversurvey_read_direction_f(rsid, xsec_id, xsec_x, xsec_y, ier)
            print *, 'direction: ', xsec_x, xsec_y

```

```

call iric_geo_riversurvey_read_leftshift_f(rsid, xsec_id, xsec_leftshift, ier)
print *, 'leftshift: ', xsec_leftshift
call iric_geo_riversurvey_read_altitudecount_f(rsid, xsec_id, xsec_altcount, ier)
print *, 'altitude count: ', xsec_altcount
allocate(xsec_altpos(xsec_altcount))
allocate(xsec_altheight(xsec_altcount))
allocate(xsec_altactive(xsec_altcount))
call iric_geo_riversurvey_read_altitudes_f( &
    rsid, xsec_id, xsec_altpos, xsec_altheight, xsec_altactive, ier)
do xsec_altid = 1, xsec_altcount
    print *, 'Altitude ', xsec_altid, ': ', &
        xsec_altpos(xsec_altid:xsec_altid), ', ', &
        xsec_altheight(xsec_altid:xsec_altid), ', ', &
        xsec_altactive(xsec_altid:xsec_altid)
end do
deallocate(xsec_altpos, xsec_altheight, xsec_altactive)
call iric_geo_riversurvey_read_fixedpointl_f( &
    rsid, xsec_id, xsec_set, xsec_x, xsec_y, xsec_index, ier)
print *, 'FixedPointL: ', xsec_set, xsec_x, xsec_y, xsec_index
call iric_geo_riversurvey_read_fixedpointR_f( &
    rsid, xsec_id, xsec_set, xsec_x, xsec_y, xsec_index, ier)
print *, 'FixedPointR: ', xsec_set, xsec_x, xsec_y, xsec_index
call iric_geo_riversurvey_read_watersurfaceelevation_f( &
    rsid, xsec_id, xsec_set, xsec_wse, ier)
print *, 'WaterSurfaceElevation: ', xsec_set, xsec_wse
end do
call iric_geo_riversurvey_close_f(rsid, 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 TestRiverSurvey

```

6.3.8. 計算格子の出力

【説明】

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は、6.3.10 に示す関数を使用して下さい。

【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_irc_write_grid_real_cell_f	セルで定義された整数の属性を出力する
cg_irc_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 表 6-9 に示します。

表 6-9 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
  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_irc_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_irc_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_irc_getgridcoord2d_f(grid_x, grid_y, ier)
  call cg_irc_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_irc_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_irc_write_sol_time_f(time, ier)
! 格子を出力
call cg_irc_write_sol_gridcoord3d_f(grid3d_x, grid3d_y, grid3d_z, ier)
! 計算結果を出力
call cg_irc_write_sol_real_f('Velocity', velocity, ier)
call cg_irc_write_sol_real_f('Density', density, ier)
call cg_irc_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_irc_write_sol_time_f(time, ier)
  ! 格子を出力
  call cg_irc_write_sol_gridcoord3d_f(grid3d_x, grid3d_y, grid3d_z, ier)
  ! 計算結果を出力
  call cg_irc_write_sol_real_f('Velocity', velocity, ier)
  call cg_irc_write_sol_real_f('Density', density, ier)
  call cg_irc_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

```

6.3.9. 時刻（もしくはループ回数）の出力

【説明】

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。

【利用する関数】

関数	備考
cg_irc_write_sol_time_f	時刻を出力する
cg_irc_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 表 6-10 に示します。

表 6-10 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_irc_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_irc_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_irc_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
```


6.3.10. 計算格子の出力（計算開始後の格子）

【説明】

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 6.3.9 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、6.3.7 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

【利用する関数】

関数	備考
cg_irc_write_sol_gridcoord2d_f	2次元構造格子を出力する
cg_irc_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 表 6-11 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

```
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_irc_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_irc_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_irc_getgridcoord2d_f(grid_x, grid_y, ier)

  ! 初期状態の情報を出力
  time = 0

  call cg_irc_write_sol_time_f(time, ier)
  ! 格子を出力
  call cg_irc_write_sol_gridcoord2d_f (grid_x, grid_y, ier)
```

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

6.3.11. 計算結果の出力

【説明】

CGNS ファイルに、計算結果を出力します。

特定の時間での計算結果を出力する前に、必ず 6.3.9 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。

iRIClib で出力できる計算結果は、大きく以下に分類されます。

- 格子点に関係なく、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

【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

【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

[Subroutines to use for outputting particles as calculation result for each time step]

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)

計算結果を出力する処理の例を 表 6-12 に示します。

表 6-12 計算結果を出力する処理の記述例

```
program Sample6
  implicit none
  include 'cgnslib_f.h'

  integer:: fin, ier, isize, jsize
  double precision:: time
  double precision:: convergence
  double precision, dimension(:,:), allocatable:: grid_x, grid_y
  real, dimension(:,:), allocatable:: velocity_x, velocity_y, depth
  integer, dimension(:,:), allocatable:: wetflag
  double precision, dimension(:,:), allocatable:: velocity_x, velocity_y, depth

  ! 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_irc_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_irc_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(velocity_x(isize,jsize), velocity_y(isize,jsize), depth(isize, jsize), wetflag(isize,jsize))
allocate(particlex(10), particley(10))
! 格子を読み込む
call cg_irc_getgridcoord2d_f(grid_x, grid_y, ier)

! 初期状態の情報を出力
time = 0
convergence = 0.1
call cg_irc_write_sol_time_f(time, ier)
! 格子を出力
call cg_irc_write_sol_gridcoord2d_f(grid_x, grid_y, ier)
! 計算結果を出力
call cg_irc_write_sol_real_f('VelocityX', velocity_x, ier)
call cg_irc_write_sol_real_f('VelocityY', velocity_y, ier)
call cg_irc_write_sol_real_f('Depth', depth, ier)
call cg_irc_write_sol_integer_f('Wet', wetflag, ier)
call cg_irc_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_irc_write_sol_time_f(time, ier)
  ! 格子を出力
  call cg_irc_write_sol_gridcoord2d_f(grid_x, grid_y, ier)
  ! 計算結果を出力
  call cg_irc_write_sol_real_f('VelocityX', velocity_x, ier)
  call cg_irc_write_sol_real_f('VelocityY', velocity_y, ier)
  call cg_irc_write_sol_real_f('Depth', depth, ier)
  call cg_irc_write_sol_integer_f('Wet', wetflag, ier)
  call cg_irc_write_sol_baseiterative_real_f('Convergence', convergence, ier)
  call cg_irc_write_sol_particle_pos2d_f(10, particlex, particley, 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 Sample6

```

なお、iRIClib では、ベクトル量の計算結果とスカラー量の計算結果では、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、上記で示したように 'VelocityX', 'VelocityY' などの名前で各成分を出力してください。

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

6.3.12. 既存の計算結果の読み込み

【説明】

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。

【利用する関数】

関数	備考
cg_irc_read_sol_count_f	計算結果の数を取得する
cg_irc_read_sol_time_f	計算結果の時刻の値を取得する
cg_irc_read_sol_iteration_f	計算結果のループ回数の値を取得する
cg_irc_read_sol_baseiterative_integer_f	整数の計算結果の値を取得する
cg_irc_read_sol_baseiterative_real_f	倍精度実数の計算結果の値を取得する
cg_irc_read_sol_gridcoord2d_f	計算結果の 2 次元構造格子を取得する
cg_irc_read_sol_gridcoord3d_f	計算結果の 3 次元構造格子を取得する
cg_irc_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_irc_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

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 表 6-12 に示します。

表 6-13 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_irc_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_irc_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_irc_read_sol_count_f(solcount, ier)
  do solid = 1, solcount
```

```

call cg_irc_read_sol_iteration_f(solid, iter, ier)
call cg_irc_read_sol_gridcoord2d_f(solid, grid_x, grid_y, ier)
call cg_irc_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（3 章参照）。

6.3.13. エラーコードの出力

【説明】

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。

【利用する関数】

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

6.3.14. CGNS ファイルを閉じる

【説明】

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。

【利用する関数】

関数	備考
cg_close_f	CGNS ファイルを閉じる。

6.4. リファレンス

6.4.1. サブルーチン一覧

サブルーチンとその分類の一覧を 表 6-14 に示します。

表 6-14 iRIClib サブルーチン一覧

分類	No.	名前	機能	複数版	頁
CGNS ファイルを開く	1	cg_open_f	CGNS ファイルを開く	×	130
内部変数の初期化	2	cg_irc_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	×	130
	3	cg_irc_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	×	131
計算条件、格子生成条件の読み込み	4	cg_irc_read_integer_f	整数型変数の値を取得する	○	131
	5	cg_irc_read_real_f	実数(倍精度)変数の値を取得する	○	131
	6	cg_irc_read_realsingle_f	実数(単精度)変数の値を取得する	○	132
	7	cg_irc_read_string_f	文字列型変数の値を取得する	○	132
	8	cg_irc_read_functionalsize_f	関数型変数のサイズを取得する	○	132
	9	cg_irc_read_functional_f	倍精度実数の関数型変数の値を取得する	○	133
	10	cg_irc_read_functional_realsingle_f	単精度実数の関数型変数の値を取得する	○	133
	11	cg_irc_read_functionalwithname_f	複数の値を持つ倍精度実数の関数型変数の値を取得する	○	134
計算格子の読み込み	12	cg_irc_gotogridcoord2d_f	格子を読み込む準備をする	○	134
	13	cg_irc_gotogridcoord3d_f	格子を読み込む準備をする	○	134
	14	cg_irc_getgridcoord2d_f	格子の X, Y 座標を読み込む	○	135
	15	cg_irc_getgridcoord3d_f	格子の X, Y, Z 座標を読み込む	○	135
	16	cg_irc_read_grid_integer_node_f	格子点で定義された整数の属性を読み込む	○	136
	17	cg_irc_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	○	136
	18	cg_irc_read_grid_integer_cell_f	セルで定義された整数の属性を読み込む	○	136
	19	cg_irc_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	○	137
	20	cg_irc_read_complex_count_f	複合型の属性のグループの数を読み込む	○	137

21	cg_irc_read_complex_integer_f	複合型の属性の整数の条件を読み込む	○	137
22	cg_irc_read_complex_real_f	複合型の属性の倍精度実数の条件を読み込む	○	138
23	cg_irc_read_complex_realsingle_f	複合型の属性の単精度実数の条件を読み込む	○	138
24	cg_irc_read_complex_string_f	複合型の属性の文字列の条件を読み込む	○	139
25	cg_irc_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	○	139
26	cg_irc_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	○	140
27	cg_irc_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	○	140
28	cg_irc_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	○	141
29	cg_irc_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	○	141
30	cg_irc_read_grid_complex_cell_f	セルで定義された複合型の属性を読み込む	○	142
31	cg_irc_read_grid_functional_timesize_f	次元「時刻」(Time)を持つ格子属性の、時刻の数を調べる	○	142
32	cg_irc_read_grid_functional_time_f	次元「時刻」(Time)の値を読み込む	○	142
33	cg_irc_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	○	144
34	cg_irc_read_grid_functional_dimensionsize_f	次元の数を調べる	○	143
35	cg_irc_read_grid_functional_dimension_integer_f	整数の次元の値を読み込む	○	143
36	cg_irc_read_grid_functional_dimension_real_f	倍精度実数の次元の値を読み込む	○	144
37	cg_irc_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	○	145
38	cg_irc_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	○	145
39	cg_irc_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	○	146

境界条件の読み込み	40	cg_irc_read_bc_count_f	境界条件の数を取得する	○	146
	41	cg_irc_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	○	147
	42	cg_irc_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	○	148
	43	cg_irc_read_bc_integer_f	整数型境界条件の値を取得する	○	149
	44	cg_irc_read_bc_real_f	実数(倍精度)境界条件の値を取得する	○	149
	45	cg_irc_read_bc_realsingle_f	実数(単精度)境界条件の値を取得する	○	150
	46	cg_irc_read_bc_string_f	文字列型境界条件の値を取得する	○	150
	47	cg_irc_read_bc_functionalsize_f	関数型境界条件のサイズを取得する	○	151
	48	cg_irc_read_bc_functional_f	倍精度実数の関数型境界条件の値を取得する	○	151
	49	cg_irc_read_bc_functional_realsingle_f	単精度実数の関数型境界条件の値を取得する	○	152
	50	cg_irc_read_bc_functionalwithname_f	複数の値を持つ倍精度実数の関数型境界条件の値を取得する	○	152
地形データの読み込み	51	cg_irc_read_geo_count_f	地形データの数を返す	○	153
	52	cg_irc_read_geo_filename_f	地形データのファイル名と種類を返す	○	153
	53	irc_geo_polygon_open_f	ポリゴンファイルを開く	×	154
	54	irc_geo_polygon_read_integervalue_f	ポリゴンの値を整数で返す	×	154
	55	irc_geo_polygon_read_realvalue_f	ポリゴンの値を実数で返す	×	154
	56	irc_geo_polygon_read_pointcount_f	ポリゴンの頂点の数を返す	×	155
	57	irc_geo_polygon_read_pointints_f	ポリゴンの頂点の座標を返す	×	155
	58	irc_geo_polygon_read_holecount_f	ポリゴンに開いた穴の数を返す	×	155
	59	irc_geo_polygon_read_holepointcount_f	ポリゴンの穴の頂点の数を返す	×	156
	60	irc_geo_polygon_read_holepoints_f	ポリゴンの穴の頂点の座標を返す	×	156
	61	irc_geo_polygon_close_f	ポリゴンファイルを閉じる	×	157
	62	irc_geo_riversurvey_open_f	河川測量データを開く	×	157
	63	irc_geo_riversurvey_read_count_f	河川横断線の数を返す	×	157

	64	iric_geo_riversurvey_read_position_f	横断線の中心点の座標を返す	×	158
	65	iric_geo_riversurvey_read_direction_f	横断線の向きを返す	×	158
	66	iric_geo_riversurvey_read_name_f	横断線の名前を文字列として返す	×	159
	67	iric_geo_riversurvey_read_realname_f	横断線の名前を実数値として返す	×	159
	68	iric_geo_riversurvey_read_leftshift_f	横断線の標高データのシフト量を返す	×	159
	69	iric_geo_riversurvey_read_altitudecount_f	横断線の標高データの数を返す	×	160
	70	iric_geo_riversurvey_read_altitudes_f	横断線の標高データを返す	×	160
	71	iric_geo_riversurvey_read_fixedpointl_f	横断線の左岸延長線のデータを返す	×	161
	72	iric_geo_riversurvey_read_fixedpointtr_f	横断線の右岸延長線のデータを返す	×	161
	73	iric_geo_riversurvey_read_watersurfaceelevation_f	横断線での水面標高のデータを返す	×	162
	74	iric_geo_riversurvey_close_f	河川測量データを閉じる	×	162
計算格子の出力	75	cg_iric_writegridcoord1d_f	1次元構造格子を出力する	○	162
	76	cg_iric_writegridcoord2d_f	2次元構造格子を出力する	○	163
	77	cg_iric_writegridcoord3d_f	3次元構造格子を出力する	○	163
	78	cg_iric_write_grid_integer_node_f	格子点で定義された整数の属性を出力する	○	164
	79	cg_iric_write_grid_real_node_f	格子点で定義された倍精度実数の属性を出力する	○	164
	80	cg_iric_write_grid_integer_cell_f	セルで定義された整数の属性を出力する	○	164
	81	cg_iric_write_grid_real_cell_f	セルで定義された倍精度実数の属性を出力する	○	165
時刻（ループ回数）の出力	82	cg_iric_write_sol_time_f	時刻を出力する	○	165
	83	cg_iric_write_sol_iteration_f	ループ回数を出力する	○	165
計算結果の出力	84	cg_iric_write_sol_gridcoord2d_f	2次元構造格子を出力する	○	166
	85	cg_iric_write_sol_gridcoord3d_f	3次元構造格子を出力する	○	166

	86	cg_irc_write_sol_baseiter ative_integer_f	整数の計算結果を出力する	○	167
	87	cg_irc_write_sol_baseiter ative_real_f	倍精度実数の計算結果を出力する	○	167
	88	cg_irc_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	○	167
	89	cg_irc_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	○	168
計算結果の出力 (粒子)	90	cg_irc_write_sol_particle _pos2d_f	粒子の位置を出力する (2 次元)	○	168
	91	cg_irc_write_sol_particle _pos3d_f	粒子の位置を出力する (3 次元)	○	169
既存の計算結果 の読み込み	92	cg_irc_read_sol_count_f	計算結果の数を取得する	○	169
	93	cg_irc_read_sol_time_f	計算結果の時刻の値を取得する	○	169
	94	cg_irc_read_sol_iteration _f	計算結果のループ回数の値を取得する	○	170
	95	cg_irc_read_sol_baseiter ative_integer_f	整数の計算結果の値を取得する	○	170
	96	cg_irc_read_sol_baseiter ative_real_f	倍精度実数の計算結果の値を取得する	○	170
	97	cg_irc_read_sol_gridcoor d2d_f	計算結果の 2 次元構造格子を取得する	○	171
	98	cg_irc_read_sol_gridcoor d3d_f	計算結果の 3 次元構造格子を取得する	○	171
	99	cg_irc_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	○	172
	100	cg_irc_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	○	172
エラーコードの 出力	101	cg_irc_write_errorcode_f	エラーコードを出力する	○	172
CGNS ファイル を閉じる	102	cg_close_f	CGNS ファイルを閉じる	×	173

なお、「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 (6.4.5 参照) 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_irc_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 6-15 に示します。

表 6-15 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	6.4.5 以降参照	第一引数：ファイル ID (integer 型)
操作対象ファイル	最後に <code>cg_irc_init_f</code> または <code>cg_irc_initread_f</code> で 指定したファイル	第一引数で指定したファイル

6.4.2. cg_open_f

- CGNS ファイルを開く。

【形式】

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

【引数】

型	変数名	I/O	内容
character(*)	filename	I	ファイル名
integer	mode	I	オープンモード CG_MODE_MODIFY : 読み書き可 CG_MODE_READ : 読み込みのみ CG_MODE_WRITE : 書き込みのみ CG_MODE_CLOSE : 閉じる
integer	fid	O	ファイル ID
integer	ier	O	エラーコード。0 なら成功

6.4.3. cg_irc_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】

call **cg_irc_init_f**(fid, ier)

【引数】

型	変数名	I/O	内容
integer	fid	I	ファイル ID
integer	ier	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。

6.4.4. cg_irc_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】

call **cg_irc_initread_f** (fid, ier)

【引数】

型	変数名	I/O	内容
integer	fid	I	ファイル ID
integer	ier	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。

6.4.5. cg_irc_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】

call **cg_irc_read_integer_f**(label, intvalue, ier)

【引数】

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

6.4.6. cg_irc_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】

call **cg_irc_read_real_f**(label, realvalue, ier)

【引数】

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

6.4.7. cg_irc_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】

call **cg_irc_read_realsingle_f**(label, realvalue, ier)

【引数】

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

6.4.8. cg_irc_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】

call **cg_irc_read_string_f**(label, strvalue, ier)

【引数】

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

6.4.9. cg_irc_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】

call **cg_irc_read_functionalsize_f**(label, size, ier)

【引数】

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

6.4.10. cg_irc_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】

call **cg_irc_read_functional_f** (label, x, y, ier)

【引数】

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

6.4.11. cg_irc_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】

call **cg_irc_read_functional_realsingle_f** (label, x, y, ier)

【引数】

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

6.4.12. cg_irc_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】

call **cg_irc_read_functionalwithname_f** (label, name, data, ier)

【引数】

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

6.4.13. cg_irc_gotogridcoord2d_f

- 二次元構造格子を読み込む準備を行う。

【形式】

call **cg_irc_gotogridcoord2d_f**(nx, ny, ier)

【引数】

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

6.4.14. cg_irc_gotogridcoord3d_f

- 三次元構造格子を読み込む準備を行う。

【形式】

call **cg_irc_gotogridcoord3d_f**(nx, ny, nz, ier)

【引数】

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

6.4.15. cg_irc_getgridcoord2d_f

- 二次元構造格子を読み込む。

【形式】

call **cg_irc_getgridcoord2d_f** (x, y, ier)

【引数】

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

6.4.16. cg_irc_getgridcoord3d_f

- 三次元構造格子を読み込む。

【形式】

call **cg_irc_getgridcoord3d_f** (x, y, z, ier)

【引数】

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

6.4.17. cg_irc_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】

call **cg_irc_read_grid_integer_node_f** (label, values, ier)

【引数】

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

6.4.18. cg_irc_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】

call **cg_irc_read_grid_real_node_f** (label, values, ier)

【引数】

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

6.4.19. cg_irc_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】

call **cg_irc_read_grid_integer_cell_f** (label, values, ier)

【引数】

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

6.4.20. cg_irc_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】

call **cg_irc_read_grid_real_cell_f** (label, values, ier)

【引数】

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

6.4.21. cg_irc_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】

call **cg_irc_read_complex_count_f** (type, num, ier)

【引数】

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

6.4.22. cg_irc_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】

call **cg_irc_read_complex_integer_f** (type, num, name, value, ier)

【引数】

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

6.4.23. cg_irc_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】

call **cg_irc_read_complex_real_f** (type, num, name, value, ier)

【引数】

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

6.4.24. cg_irc_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】

call **cg_irc_read_complex_realsingle_f** (type, num, name, value, ier)

【引数】

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

6.4.25. cg_irc_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】

call **cg_irc_read_complex_string_f** (type, num, name, value, ier)

【引数】

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

name には、ソルバ定義ファイルの Item 要素で指定した name 属性の値を指定します。
iRIC 上で「名前」に指定した値を読み込むには、name に "_caption" を指定します。

6.4.26. cg_irc_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】

call **cg_irc_read_complex_functionalsize_f** (type, num, name, size, ier)

【引数】

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

6.4.27. cg_irc_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】

call **cg_irc_read_complex_functional_f** (type, num, name, x, y, ier)

【引数】

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

6.4.28. cg_irc_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】

call **cg_irc_read_complex_functional_realsingle_f** (type, num, name, x, y, ier)

【引数】

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

6.4.29. `cg_irc_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】

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

【引数】

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

6.4.30. `cg_irc_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】

call `cg_irc_read_grid_complex_node_f` (label, values, ier)

【引数】

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

6.4.31. cg_irc_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】

call **cg_irc_read_grid_complex_cell_f** (label, values, ier)

【引数】

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

6.4.32. cg_irc_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】

call **cg_irc_read_grid_functionaltimesize_f** (label, count, ier)

【引数】

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

6.4.33. cg_irc_read_grid_functionaltime_f

- 次元「時刻」(Time)の値を読み込む。

【形式】

call **cg_irc_read_grid_functionaltime_f** (label, values, ier)

【引数】

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

6.4.34. cg_irc_read_grid_functionaldimensionsize_f

- 次元の数を調べる。

【形式】

call **cg_irc_read_grid_functionaltime_f** (label, dimname, count, ier)

【引数】

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

6.4.35. cg_irc_read_grid_functionaldimension_integer_f

- 整数の次元の値を読み込む

【形式】

call **cg_irc_read_grid_functionaldimension_integer_f** (label, dimname, values, ier)

【引数】

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

6.4.36. cg_irc_read_grid_functionaldimension_real_f

- 実数の次元の値を読み込む

【形式】

call **cg_irc_read_grid_functionaldimension_real_f** (label, dimname, values, ier)

【引数】

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

6.4.37. cg_irc_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】

call **cg_irc_read_grid_functional_integer_node_f** (label, dimid, values, ier)

【引数】

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

6.4.38. cg_irc_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】

call **cg_irc_read_grid_functional_real_node_f** (label, dimid, values, ier)

【引数】

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

6.4.39. cg_irc_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】

call **cg_irc_read_grid_functional_integer_cell_f** (label, dimid, values, ier)

【引数】

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

6.4.40. cg_irc_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】

call **cg_irc_read_grid_functional_real_cell_f** (label, dimid, values, ier)

【引数】

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

6.4.41. cg_irc_read_bc_count_f

- 境界条件の数を取得する。

【形式】

call **cg_irc_read_bc_count_f** (type, num)

【引数】

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

6.4.42. cg_irc_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】

call **cg_irc_read_bc_indicessize_f** (type, num, size, ier)

【引数】

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

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 6-16 に示すように異なります。

表 6-16 境界条件を設定された位置と size に返される値の関係

No.	境界条件を設定された位置	size に返される値
1	格子点 (node)	格子点の数
2	セル (cell)	セルの数
3	辺 (edge)	辺の数×2

6.4.43. cg_irc_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）のインデックスの配列を取得する。

【形式】

call **cg_irc_read_bc_indices_f** (type, num, indices, ier)

【引数】

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

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 6-17 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 6-17 境界条件を設定された位置と indices に返される値の関係

No.	境界条件を設定された位置	indices に返される値
1	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)
2	セル (cell)	(セル 1 の I), (セル 1 の J) ... (セル N の I), (セル N の J)
3	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)

6.4.44. cg_irc_read_bc_integer_f

- 整数型の境界条件の値を読み込む。

【形式】

call **cg_irc_read_bc_integer_f** (type, num, name, value, ier)

【引数】

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

6.4.45. cg_irc_read_bc_real_f

- 実数(倍精度)型の境界条件の値を読み込む。

【形式】

call **cg_irc_read_bc_real_f** (type, num, name, value, ier)

【引数】

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

6.4.46. cg_irc_read_bc_realsingle_f

- 実数(単精度)型の境界条件の値を読み込む。

【形式】

call **cg_irc_read_bc_realsingle_f** (type, num, name, value, ier)

【引数】

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

6.4.47. cg_irc_read_bc_string_f

- 文字列型の境界条件の値を読み込む。

【形式】

call **cg_irc_read_bc_string_f** (type, num, name, value, ier)

【引数】

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

name には、ソルバ定義ファイルの **Item** 要素で指定した **name** 属性の値を指定します。
iRIC 上で「名前」に指定した値を読み込むには、**name** に "_caption" を指定します。

6.4.48. cg_irc_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】

call **cg_irc_read_bc_functionalsize_f** (type, num, name, size, ier)

【引数】

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

6.4.49. cg_irc_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】

call **cg_irc_read_bc_functional_f** (type, num, name, x, y, ier)

【引数】

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

6.4.50. cg_irc_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】

call **cg_irc_read_bc_functional_realsingle_f** (type, num, name, x, y, ier)

【引数】

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

6.4.51. cg_irc_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】

call **cg_irc_read_bc_functionalwithname_f** (type, num, name, paramname, data, ier)

【引数】

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

6.4.52. cg_irc_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】

call **cg_irc_read_geo_count_f** (name, geocount, ier)

【引数】

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

6.4.53. cg_irc_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】

call **cg_irc_read_geo_filename_f** (name, geoid, geofilename, geotype, ier)

【引数】

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

なお、geotype で読み込まれる地形データの種類は、本作業で新たに作成した iriclib_f.h で定義された、表 6-18 に示す値のいずれかです。

表 6-18 iriclib_f.h で定義された地形データ種類を表す定数

No.	定数名	値	備考
1	IRIC_GEO_POLYGON	1	ポリゴン
2	IRIC_GEO_RIVERSURVEY	2	河川測量データ

6.4.54. iric_geo_polygon_open_f

- ポリゴンファイルを開く。

【形式】

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

【引数】

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

6.4.55. iric_geo_polygon_read_integervalue_f

- ポリゴンの値を整数で返す。

【形式】

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

【引数】

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

6.4.56. iric_geo_polygon_read_realvalue_f

- ポリゴンの値を実数で返す。

【形式】

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

【引数】

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

6.4.57. iric_geo_polygon_read_pointcount_f

- ポリゴンの頂点の数を返す。

【形式】

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

【引数】

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

6.4.58. iric_geo_polygon_read_points_f

- ポリゴンの頂点の座標を返す。

【形式】

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

【引数】

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

6.4.59. iric_geo_polygon_read_holecount_f

- ポリゴンに開いた穴の数を返す。

【形式】

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

【引数】

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

6.4.60. iric_geo_polygon_read_holepointcount_f

- ポリゴンの穴の頂点の数を返す。

【形式】

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

【引数】

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

6.4.61. iric_geo_polygon_read_holepoints_f

- ポリゴンの穴の頂点の座標を返す。

【形式】

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

【引数】

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

6.4.62. iric_geo_polygon_close_f

- ポリゴンファイルを閉じる。

【形式】

call **iric_geo_polygon_close_f**(pid, ier)

【引数】

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

6.4.63. iric_geo_riversurvey_open_f

- 河川測量データを開く。

【形式】

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

【引数】

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

6.4.64. iric_geo_riversurvey_read_count_f

- 河川横断線の数を返す。

【形式】

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

【引数】

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

6.4.65. iric_geo_riversurvey_read_position_f

- 横断線の中心点の座標を返す。

【形式】

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

【引数】

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

6.4.66. iric_geo_riversurvey_read_direction_f

- 横断線の向きを返す。

【形式】

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

【引数】

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

6.4.67. iric_geo_riversurvey_read_name_f

- 横断線の名前を文字列として返す。

【形式】

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

【引数】

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

6.4.68. iric_geo_riversurvey_read_realname_f

- 横断線の名前を実数値として返す。

【形式】

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

【引数】

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

6.4.69. 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】

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

【引数】

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

6.4.70. iric_geo_riversurvey_read_altitudecount_f

- 横断線の標高データの数を返す。

【形式】

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

【引数】

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

6.4.71. iric_geo_riversurvey_read_altitudes_f

- 横断線の中心点の座標を返す。

【形式】

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

【引数】

型	変数名	I/O	内容
integer	rid	I	河川測量データの ID
pointid	pointid	I	横断線の ID
double precision , dimension(:), allocatable	position	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)
double precision , dimension(:), allocatable	height	O	標高データの高さ
integer, dimension(:), allocatable	active	O	標高データの有効/無効 (1: 有効, 0: 無効)
integer	ier	O	エラーコード。0 なら成功

6.4.72. iric_geo_riversurvey_read_fixedpointl_f

- 横断線の左岸延長線のデータを返す。

【形式】

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

【引数】

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

6.4.73. iric_geo_riversurvey_read_fixedpointtr_f

- 横断線の右岸延長線のデータを返す。

【形式】

call **iric_geo_riversurvey_read_fixedpointtr_f**(rid, pointid, set, directionx, directiony, index, ier)

【引数】

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

6.4.74. iric_geo_riversurvey_read_watersurfaceelevation_f

- 横断線での水面標高のデータを返す。

【形式】

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

【引数】

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

6.4.75. iric_geo_riversurvey_close_f

- 河川測量データファイルを閉じる。

【形式】

call **iric_geo_riversurvey_close_f**(pid, ier)

【引数】

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

6.4.76. cg_iric_writegridcoord1d_f

- 1 次元構造格子を出力する。

【形式】

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

【引数】

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

6.4.77. cg_irc_writegridcoord2d_f

- 2次元構造格子を出力する。

【形式】

call **cg_irc_writegridcoord2d_f** (nx, ny, x, y, ier)

【引数】

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

6.4.78. cg_irc_writegridcoord3d_f

- 3次元構造格子を出力する。

【形式】

call **cg_irc_writegridcoord3d_f** (nx, ny, nz, x, y, z, ier)

【引数】

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

6.4.79. cg_irc_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】

call **cg_irc_write_grid_integer_node_f** (label, values, ier)

【引数】

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

6.4.80. cg_irc_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】

call **cg_irc_write_grid_real_node_f** (label, values, ier)

【引数】

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

6.4.81. cg_irc_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】

call **cg_irc_write_grid_integer_cell_f** (label, values, ier)

【引数】

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

6.4.82. cg_irc_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】

call **cg_irc_read_grid_real_cell_f** (label, values, ier)

【引数】

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

6.4.83. cg_irc_write_sol_time_f

- 時刻を出力する。

【形式】

call **cg_irc_write_sol_time_f** (time, ier)

【引数】

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

6.4.84. cg_irc_write_sol_iteration_f

- ループ回数を出力する。

【形式】

call **cg_irc_write_sol_iteration_f** (iteration, ier)

【引数】

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

6.4.85. `cg_irc_write_sol_gridcoord2d_f`

- 2次元構造格子を出力する。

【形式】

call `cg_irc_write_sol_gridcoord2d_f`(x, y, ier)

【引数】

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

6.4.86. `cg_irc_write_sol_gridcoord3d_f`

- 3次元構造格子を出力する。

【形式】

call `cg_irc_write_sol_gridcoord3d_f`(x, y, z, ier)

【引数】

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

6.4.87. cg_irc_write_sol_baseiterative_integer_f

- 整数の計算結果を出力する。

【形式】

call **cg_irc_write_sol_baseiterative_integer_f** (label, val, ier)

【引数】

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

6.4.88. cg_irc_write_sol_baseiterative_real_f

- 倍精度実数の計算結果を出力する。

【形式】

call **cg_irc_write_sol_baseiterative_real_f** (label, val, ier)

【引数】

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

6.4.89. cg_irc_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】

call **cg_irc_write_sol_integer_f** (label, val, ier)

【引数】

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

6.4.90. 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】

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

【引数】

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

6.4.91. cg_iric_write_sol_particle_pos2d_f

- 粒子の位置を出力する。(2次元)

【形式】

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

【引数】

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

6.4.92. cg_irc_write_sol_particle_pos3d_f

- 粒子の位置を出力する。(3次元)

【形式】

call **cg_irc_write_sol_particle_pos3d_f** (count, x, y, ier)

【引数】

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

6.4.93. cg_irc_read_sol_count_f

- 計算結果の数を取得する。

【形式】

call **cg_irc_read_sol_count_f** (count, ier)

【引数】

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

6.4.94. cg_irc_read_sol_time_f

- 計算結果の時刻の値を取得する。

【形式】

call **cg_irc_read_sol_time_f** (step, time, ier)

【引数】

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

6.4.95. cg_irc_read_sol_iteration_f

- 計算結果のループ回数の値を取得する。

【形式】

call **cg_irc_read_sol_iteration_f** (step, iteration, ier)

【引数】

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

6.4.96. cg_irc_read_sol_baseiterative_integer_f

- 整数の計算結果の値を取得する。

【形式】

call **cg_irc_read_sol_baseiterative_integer_f** (step, label, val, ier)

【引数】

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

6.4.97. cg_irc_read_sol_baseiterative_real_f

- 倍精度実数の計算結果の値を取得する。

【形式】

call **cg_irc_read_sol_baseiterative_real_f** (step, label, val, ier)

【引数】

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

6.4.98. cg_irc_read_sol_gridcoord2d_f

- 計算結果の 2 次元構造格子を取得する。

【形式】

call **cg_irc_read_sol_gridcoord2d_f** (step, x, y, ier)

【引数】

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

6.4.99. cg_irc_read_sol_gridcoord3d_f

- 計算結果の 3 次元構造格子を取得する。

【形式】

call **cg_irc_read_sol_gridcoord3d_f** (step, x, y, z, ier)

【引数】

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

6.4.100.cg_irc_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】

call **cg_irc_read_sol_integer_f** (step, label, val, ier)

【引数】

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

6.4.101.cg_irc_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】

call **cg_irc_read_sol_real_f** (step, label, val, ier)

【引数】

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

6.4.102.cg_irc_write_errorcode_f

- エラーコードを出力する。

【形式】

call **cg_irc_write_errorcode_f** (code, ier)

【引数】

型	変数名	I/O	内容
integer	code	I	格子生成プログラムが返すエラーコード
integer	ier	O	エラーコード。0 なら成功

6.4.103.cg_close_f

- CGNS ファイルを閉じる。

【形式】

call **cg_close_f**(fid, ier)

【引数】

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

7. 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とき、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、condFile へ代入します。

```
icount = nargs()      ! コマンド名が個数に含まれるので、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、condFile へ代入します。

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。

```
icount = iargc() ! コマンド名は個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.lib	cgnslib.lib
GNU Fortran(gfortran)	iriclib.a	libcgns.a

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, cgnslib.lib, libiric.lib, msvcprt.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 libcgns.lib iriclib.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, cgnslib.a, libiric.a, 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 cgnslib.a libiric.a
```

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	定義例
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。	表 7-3 参照

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。

表 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 表 7-4 に示します。

表 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_irc_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 7-5 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。また、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 表 7-6 に示します。

表 7-5 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。	

表 7-6 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 ADFviewer を用いて閲覧する方法を説明します。ADFviewer は、CGNS ライブラリの開発元によってフリーソフトとして公開されているソフトウェアです。

1) CGNSTools のインストール

まず、ADFviewer を含む、CGNSTools をインストールします。CGNSTools 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。



<http://sourceforge.net/projects/cgns/>

SourceForge.net > [Find Software](#) > [CFD General Notation System \(CGNS\)](#)


CFD General Notation System (CGNS) Beta
 by [brucewedan](#), [rumsey](#), [thomas_hauser](#)
[Summary](#) [Files](#) [Support](#) [Develop](#)

[Share](#) [f](#) [t](#) [g](#) [+](#) [More](#)

The CFD General Notation System (CGNS) provides a standard for recording and recovering computer data associated with the numerical solution of fluid dynamics equations. Current source is available by SVN, although only the downloads are supported.



Download Now!
First beta for cgns 3.0 (3.6 MB)  OR [View all files](#) 

<http://www.cgns.org>

[Show project details](#)

Rate and Review

Would you recommend this project?


 OR 

Related Projects



[AXopenCFD](#)
dev lang=C++; Calculate=CUDA
;GUI=GTK; 3DLib=OpenGL
AXopenCFD is a free and open source CFD code ...

[RapidDownloader](#)
RapidDownloader is application for automatized downloading from file hosting services like

Ratings and Reviews

Show: [Everything](#) 

100% of 4 users recommend this project


Thumbs up:  4
Thumbs down:  0

Be the first to post a text review of CFD General Notation System (CGNS). Rate and review a project by

CGNS のホームページから、上記丸で囲って示した「View all files」のリンクをクリックします。すると、以下で示すように CGNS に関連したさまざまなプログラムがダウンロードできる画面に移動します。ここで、丸で囲って示した「win-install-2-5-2.zip」をクリックしてダウンロードしてください。



これが、CGNSTools のインストーラです。解凍してインストーラを実行することで、CGNSTools がインストールされます。

SourceForge.net > [Find Software](#) > [CFD General Notation System \(CGNS\)](#) > [Browse Files](#)
































CFD General Notation System (CGNS) Beta
 by [brucewedan](#), [rumsey](#), [thomas_hauser](#)
[Summary](#) [Files](#) [Support](#) [Develop](#)

[Share](#) [f](#) [t](#) [g](#) [+](#) [More](#)

The CFD General Notation System (CGNS) provides a standard for recording and recovering computer data associated with the numerical solution of fluid dynamics equations. Current source is available by SVN, although only the downloads are supported.

Download Now!
First beta for cgns 3.0 (3.6 MB)  OR [View all files](#) 

Browse Files for CFD General Notation System (CGNS)

File/Folder Name	Platform	Size	Date	Downloads	Notes/Subscribe
Newest Files					
 pcgns-0.2.0-Source.tar.gz		40.8 kB	2010-05-25	104	
All Files					
▼  Parallel_cgns		40.8 kB	2010-05-25	104	 
 pcgns-0.2.0-Source.tar.gz		40.8 kB	2010-05-25	104	
▶  cgns_3.0_beta Beta releases of cgns 3.0		3.6 MB	2010-05-24	468	 
▶  cgnslib_2.5		1.2 MB	2009-09-01	10,659	 
▼  cgnstools		28.5 MB	2009-09-01	8,690	 
▶  Version 2.5, Release 4		1.3 MB	2009-09-01	2,118	 
▼  Version 2.5, Release 2		5.9 MB	2007-09-07	3,248	 
 win-install-2-5-2.zip		4.5 MB	2007-09-07	1,625	
 cgnslib-2-5-2.tar.gz		1.4 MB	2007-09-07	1,623	
▶  Version 2.4, Rev 1		8.1 MB	2005-08-23	1,808	 
▶  Version 2.3, Release		7.2 MB	2004-10-02	864	 

2) ADFviewer を利用した CGNS ファイルの閲覧

ADFviewer 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 ADFviewer 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

CGNS ファイルを開いた後の ADFviewer の画面表示例を 図 7-1 に示します。

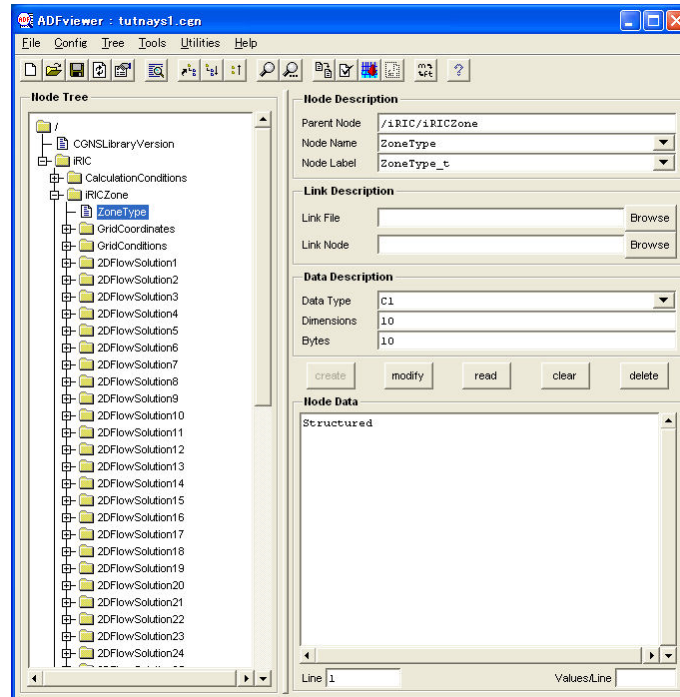


図 7-1 ADFviewer 表示例

画面の左側の Node Tree には、CGNS ファイル内のツリー構造が表示されます。Node Tree で、閲覧したい項目を選択すると、画面右側に、選択した項目の名前や種類、内部のデータなどが表示されます。

なお、大きな配列が格納された項目（例: 計算格子の X 座標）などを選択した時は、すぐにはデータが表示されません。大きな配列については、項目を選択した上で 図 7-2 に示した「read」ボタンを押すと、データが読み込まれて表示されます。

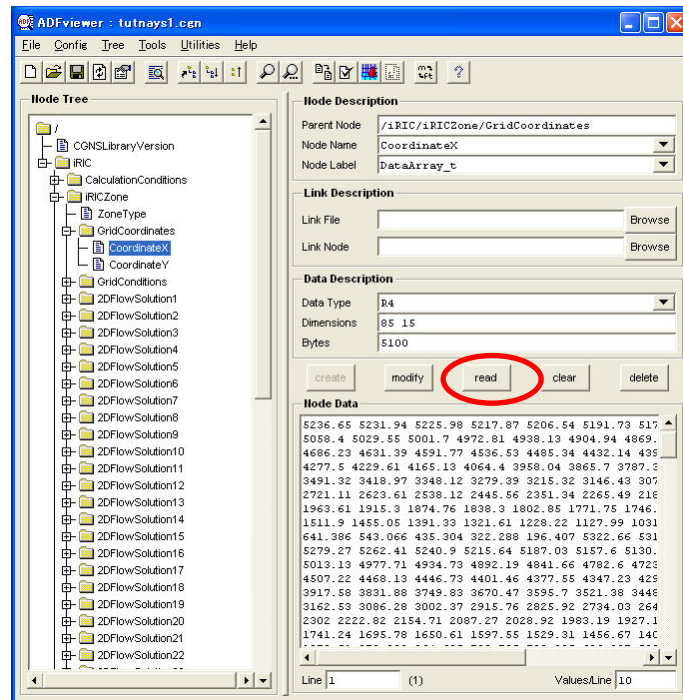


図 7-2 ADFviewer での計算格子の X 座標 表示例

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 7-7 を参照してください。

表 7-7 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://www.grc.nasa.gov/WWW/cgns/midlevel/index.html
CGNS ファイルの内部構造	http://www.grc.nasa.gov/WWW/cgns/sids/index.html
CGNS ライブラリの利用プログラムの記述例集	http://sourceforge.net/projects/cgns/files/UserGuideCode/Release%203/UserGuideCodeV3.zip/download

【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。

iRIC Software Developer's Manual

監修	一般財団法人 北海道河川財団	全体
----	----------------	----

編集・執筆者	みずほ情報総研株式会社	全体
--------	-------------	----