iRIC Developer's manual

Release 3.0.0

iRIC organization

Jul 09, 2022

Contents

CHAPTER 1

About This Manual

This manual provides information necessary for the following people:

- Developers of solvers that run on iRIC.
- Developers of grid generating programs that run on iRIC

Developers of solvers should read *Steps of developing a solver* (page 3) first, to understand the steps of developing a solver. After that, please read *About definition files (XML)* (page 55), *iRIClib* (page 97), *Other Informations* (page 217) when you need to.

Developers of grid generating programs should read *Steps of developing a grid generating program* (page 35) first, to understand the steps of developing a grid generating program. After that, please read *About definition files (XML)* (page 55), *iRIClib* (page 97), *Other Informations* (page 217) when you need to.

CHAPTER 2

Steps of developing a solver

2.1 Preparing development environment

Prepare environment to develop iRIC solvers.

- To develop with C/C++, you can use Visual C/C++ 2013 or newer.
- To develop with FORTRAN, you can use Intel Fortran 2015 or newer.

When building solvers, you need to link against cgnslib and iriclib.

Headers and Libraries are bundled with iRIC GUI. If you've installed to standard folder, they are stored in the folder below:

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

2.2 Abstract

Solver is a program that load grid and calculation conditions, execute a river simulation, and output calculation results.

To add a solver to iRIC, it is necessary to make and deploy files shown in Table 2.1.

Solver developers have to create a new folder under "solvers" folder under iRIC install folder, and deploy files in Table 2.1 that you've prepared for your new solver.

Table 2.1: Files related to a Solver

File name	Description
definition.xml	Solver definition file
solver.exe	Executable module of the solver. Developers can select any name.
translation_ja_JP.ts etc.	Dictionary files for a solver definition file
README	File explaining the solver
LICENSE	License information file for the solver

Abstracts of each file are as follows:

2.2.1 definition.xml

File that defines the following information of solvers:

- Basic Information
- Calculation Conditions
- Grid Attributes

iRIC loads definition xml, and provides interface for creating calculation conditions and grids that can be used by the solver. Solver definition file should be written in English.

2.2.2 Solver

Executable module of a river simulation solver. It loads calculation condition and grids created using iRIC, executes river simulation, and outputs result.

Solvers use calculation data files created by iRIC, for loading and writing calculation condition, grids, and calculation results. Solvers can also use arbitrary files for data I/O that cannot be loaded from or written into calculation data files.

Solvers can be developed using FORTRAN, Python, C or C++. In this chapter, a sample solver is developed in FORTRAN.

2.2.3 translation_ja_JP.ts etc.

Dictionary files for a solver definition file. It provides translation information for texts shown on dialogs or object browser in iRIC. Dictionary files are created as separate files for each language. For example, "translation_ja_JP.ts" for Japanese, "translation_ka_KR.ts" for Korean.

2.2.4 README

README is a text file that describes about the solver. The content of README is shown in the "Description" tab in the [Select Solver] dialog.

2.2.5 LICENSE

LICENSE is a text file that describes about the license of the solver. The content of LICENSE is shown in the "License" tab in the [Select Solver] dialog.

Figure 2.1 shows the relationships of iRIC, solver and related files.

This chapter explains the steps to create the files described in this section.

2.3 Creating a folder

Create a special folder for the solver you develop under the "solvers" folder under the installation folder of iRIC. This time, please create "example" folder.

Figure 2.1: Relationships between iRIC, solvers, and related files

2.4 Creating a solver definition file

Create a solver definition file.

In solver definition file, you are going to define the information shown in Table 2.2.

Table 2.2: Informations defined in solver definition file

Item	Description	Required
Basic information	The solver name, developer name, release date, etc.	Yes
Calculation Condition	Calculation condition for solver execution	Yes
Grid Attributes	Attributes defined at nodes or cells of calculation grids	Yes
Boundary Conditions	Boundary conditions defined at nodes or cells of calculation grids	

Solver definition file is described in XML language. The basic grammer of XML language is explained in *XML files basics* (page 94).

In this section, we add definition information of a solver in the order shown in Table 2.2.

2.4.1 Defining basic information

Define basic information of a solver. Create a file with the content shown in List 2.1, and save it with name "definition.xml" under "example" folder that you created in *Creating a folder* (page 4)

List 2.1: Example solver definition file that contains basic information

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

At this point, the structure of the solver definition file is as shown in Figure 2.2.

Figure 2.2: Solver definition file structure

Now make sure the solver definition file is arranged correctly.

Launch iRIC. The [iRIC Start Page] dialog (Figure 2.3) is shown, so please click on [New Project]. The [Solver Select] dialog (Figure 2.4) will open, so make sure if there is a new item "Sample Solver" in the solver list. When you find it, select it and make sure that the basic information of the solver you wrote in solver definition file is shown.

Please note that the following attributes are not shown on this dialog:

- name
- executable
- iterationtype
- gridtype

Figure 2.3: The [iRIC Start Page] dialog

You sould take care about name attribute and version attribute, when you want to update a solver. Please refer to *Notes on solver version up* (page 93) for the detail.

2.4.2 Defining calculation conditions

Define calculation conditions. Calculation conditions are defined in "CalculationCondition" element. Add description of calculation condition to the solver definition file you created in *Defining basic information* (page 6) . Solver definition file content is now as shown in List 2.2. The added part is shown with highlight.

List 2.2: Example of solver definition file that now has calculation condition definition

```
1
2
```

(continues on next page)

Figure 2.4: The [Select Solver] dialog

(continued from previous page)

At this point, the structure of the solver definition file is as shown in Figure 2.5.

Figure 2.5: Solver definition file structure

Now make sure that solver definition file is arranged correctly.

Launch iRIC. The [iRIC Start page] dialog (Figure 2.3) will open, so please click on [Create New Project], select "Sample Solver" from the list, and click on [OK]. The Warning dialog (Figure 2.6) will be open, so click on [OK].

Figure 2.6: The [Warning] dialog

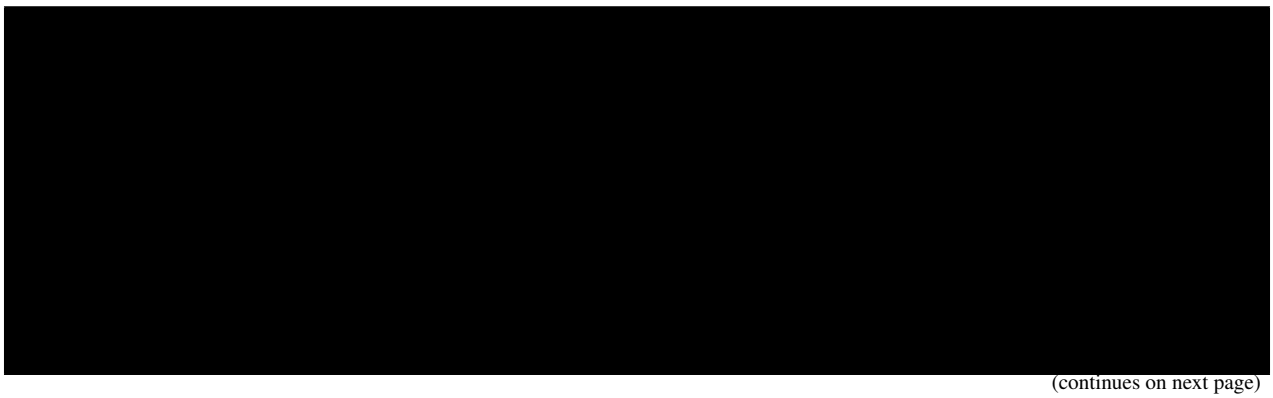
The [Pre-processing Window] will open, so perform the following:

Menu bar: –> [Calculation Condition] (C) –> [Setting] (S)

The [Calculation Condition] dialog (Figure 2.7) will open. Now you can see that the calculation condition items you defined in List 2.2 are shown.

Now add one more group and add calculation condition items. Add "Water Surface Elevation" Tab element just after "Basic Settings" Tab element. List 2.3 shows the solver definition file that has definition of "Water Surface Elevation" Tab. The added part is shown with highlight.

List 2.3: Example of solver definition file that now has calculation condition definition (abbr.)



(continues on next page)

Figure 2.7: The [Calculation Condition] dialog

(continued from previous page)

At this point, the structure of the solver definition file is as shown in Figure 2.8.

Now make sure that solver definition file is arranged correctly. Do the operation you did again, to open The [Calculation Condition] dialog (Figure 2.9).

Now you can see that the new group "Water Surface Elevation" is added in the group list. You'll also notice that the "Constant Value" item is enabled only when "Type" value is "Constant", and the "Time Dependent Value" item is enabled only when "Type" value is "Time Dependent".

What it comes down to is:

- Calculation condition group is defined with "Tab" element, and calculation condition item is defined with "Item" element.
- The Structure under "Definition" elements depends on the condition type (i. e. Integer, Real number, functional

Figure 2.8: Solver definition file structure

Figure 2.9: The [Calculation Condition] dialog

etc.). Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for examples of calculation condition items for each type.

- Dependenciy between calculation condition items can be defined with "Condition" element. In "Condition" element, define the condition when that item should be enabled. Refer to *Example of condition to activate calculation conditions etc.* (page 70) for examples of "Condition" element.
- In this example, the calculation condition dialog shows the items as a simple list, but iRIC has feature to show items with more complex layouts, like layout with group boxes. Refer to *Example of dialog layout definition* (page 70) for more complex layouts.

2.4.3 Defining Grid attributes

Define grid attributes. Grid attributes are defined with "GridRelatedCondition" element. Add definition of grid related condition to the solver definition file you created, as shown in List 2.4. The added part is shown with highlight.

List 2.4: Example of solver definition file that now has grid related condition (abbr.)

Now make sure that solver definition file is arranged correctly.

Launch iRIC, and starts a new project with solver "Sample Solver". Now you will see the [Pre-processing Window] like in Figure 2.10. When you create or import a grid, the [Pre-processing Window] will become like in Figure 2.11.

When you do not know how to create or import a grid, refer to the User Manual.

When you edit the grid attribute "Elevation" with the following procedure, the [Edit Elevation] dialog (Figure 2.12) will open, and you can check that you can input real number as "Elevation" value.

- Select [Grid] –> [Node attributes] –> [Elevation] in the [Object Browser].
- Select grid nodes with mouse clicking in the canvas area
- Show context menu with right-clicking, and click on [Edit].

When you do the same operation against attribute "Obstacle" to edit "Obstacle" value, the [Obstacle edit dialog] (Figure 2.13) will open, and you can check that you can select obstacle values from that you defined in solver definition file, in List 2.4.

What it comes down to is:

Figure 2.10: The [Pre-processing Window]

Figure 2.11: The [Pre-processing Window] after creating a grid

Figure 2.12: The [Edit Elevation] dialog

Figure 2.13: The [Obstacle edit dialog]

- Grid attribute is defined with "Item" element under "GridRelatedCondition" element.
- The structure under "Item" element is basically the same to that for calculation condition, but there are different points:
- You have to specify "position" attribute to determine whether that attribute is defined at nodes or cells.
- You can not use types "String", "Functional", "File name" and "Folder name".
- You can not define dependency.
- You can define dimension of the attribute, using "Dimension" element.

For grid attributes, iRIC defines some special names. For attributes for certain purposes, you should use those names. Refer to *Special names for grid attributes and calculation results* (page 219) for the special grid attribute names.

2.4.4 Defining Boundary Conditions

Define boundary conditions. You can define boundary conditions with "BoundaryCondition" element. Boundary conditions are not required.

Add definition of "Boundary Condition" to the solver definition file you created, as shown in List 2.5. The added part is shown with highlight.

List 2.5: Example of solver definition file that now has boundary condition (abbr.)

```
1
2
3
4
5
6
7
8
9
```

(continues on next page)

(continued from previous page)

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

Now make sure that solver definition file is arranged correctly.

Launch iRIC, and start a new project with solver "Sample Solver". When you create or import a grid, the [Pre-processing Window] will become like Figure 2.14. When you do now know how to create or imprt a grid, refer to the User Manual.

Figure 2.14: The [Pre-processing Window] after creating a grid

Click on [Add new Inflow] on the context menu on [Boundary Condition] node, and The [Boundary Condition] dialog (Figure 2.15) will open, and you can define boundary condition on this dialog.

When you have finished defining boundary condition, click on [OK]. Drag around the grid nodes to select nodes, and click on [Assign Condition] in the context menu. Figure 2.16 shows an example of a grid with boundary condition.

What it comes down to is:

- Boundary condition is defined Grid attribute is defined with "Item" element under "GridRelatedCondition" element.
- The structure under "Item" element is the same to that for calculation condition.

Figure 2.15: The [Boundary Condition] dialog

Figure 2.16: Example of a grid with boundary condition

2.5 Creating a solver

Create a solver. In this example we will develop a solver with FORTRAN.

To develop a solver that works together with iRIC, you have to make it use calculation data file that iRIC generate, for loading calculation conditions and grid and outputting calculation results.

The calculation data file that iRIC generates is a CGNS file. You can use a library called iRIClib to write code for loading and writing CGNS files.

In this section, the procedure to develop a solver is described, that load calculation data file, that iRIC generates. Table 2.3 shows the input and output processing that the solver do against the calculation data file.

Table 2.3: The I/O processing flow of solver

Processing	Required
Opens calculation data file	Yes
Initializes iRIClib	Yes
Loads calculation condition	Yes
Loads calculation grid	Yes
Outputs time (or iteration)	Yes
Outputs calculation result	Yes
Closes calculation data file	Yes

In this section, we will develop a solver in the following procedure:

1. Create a scelton
2. Adds calculation data file opening and closing codes
3. Adds codes to load calculation conditions, calculation girds, and boundary conditions
4. Adds codes to output time and calculation results

2.5.1 Creating a skelton

First, create a scelton of a solver. Create a new file with the source code in List 2.6, and save as "sample.f90". At this point, the solver does nothing.

Compile this source code. The way to compile a source code differs by the compiler. Refer to *Linking iRIClib, cgnslib using Fortran* (page 218) for the procedure to compile using Intel Fortran Compiler and gfortran.

List 2.6: Sample solver source code

When it was compiled successfully, copy the executable file to the folder you created in *Creating a folder* (page 4), and rename it into the name you specified as [executable] attribute in *Defining basic information* (page 6). This time, rename into "solver.exe". Copy the DLL files into that folder, that is needed to run the solver.

Now check whether it can be launched from iRIC successfully.

Starts a new project that uses "Example Solver", and performs the following:

Menu bar: [Simulation] (S) –> [Run] (R)

The [Solver Console] opens, and the message "Sample Program" will be shown (Figure 2.17). If the message is shown, it means that the solver was launched by iRIC successfully.

Figure 2.17: The [Solver Console]

2.5.2 Adding calculation data file opening and closing codes

Adds codes for opening and closing calculation data file.

The solver has to open calculation data file in the first step, and close it in the last step.

iRIC will handle the file name of calculation data file as a the first argument, so open that file.

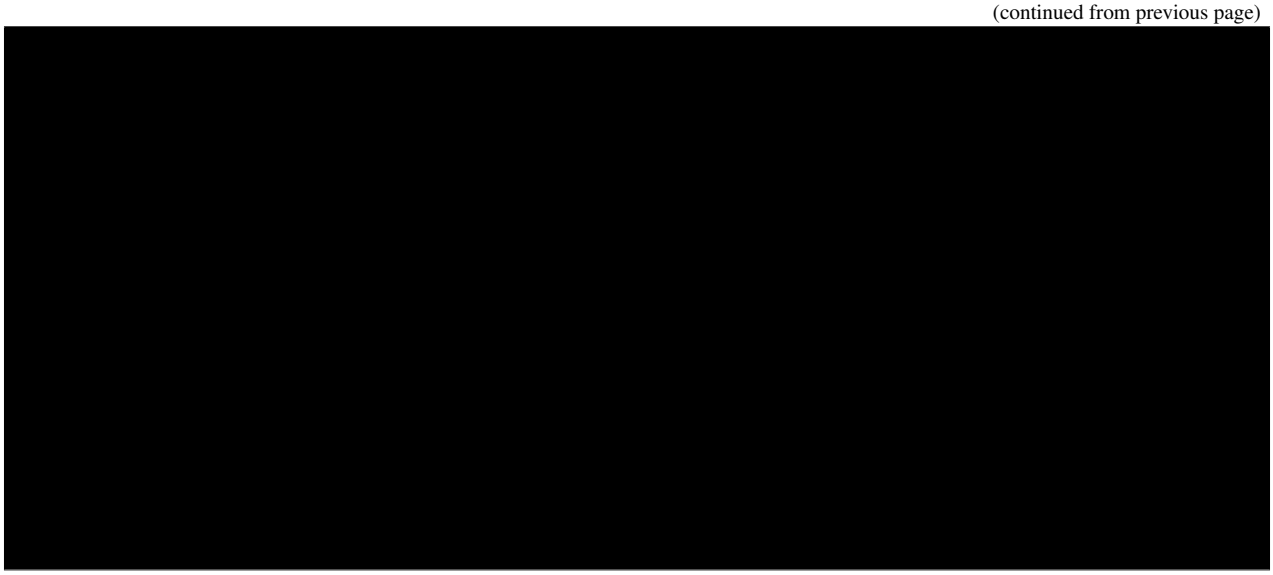
The way to handle the number of arguments and the arguments differs by compilers. Refer to *Handling command line arguments in Fortran programs* (page 217) for the way to handle them with Intel Fortran Compiler and gfortran. In this chapter we will add codes that can be compiled using Intel Fortran Compiler.

Table List 2.7 shows the source code with the lines to open and close calculation data file. The added lines are shown with highlight.

List 2.7: The source code with lines to open and close file

(continues on next page)

(continued from previous page)



Compile and deploy the executable file, just like in *Creating a skelton* (page 17).

Check whether it can be launched from iRIC successfully, just like in *Creating a skelton* (page 17).

Refer to *Opening a CGNS file* (page 99), *Initializing iRIClib* (page 99) and *Closing a CGNS file* (page 129) for the details of the subroutines added in this section.

2.5.3 Adding codes to load calculation conditions, calculation grids, and boundary conditions

Adds codes to load calculation conditions, calculation girds, and boundary conditions.

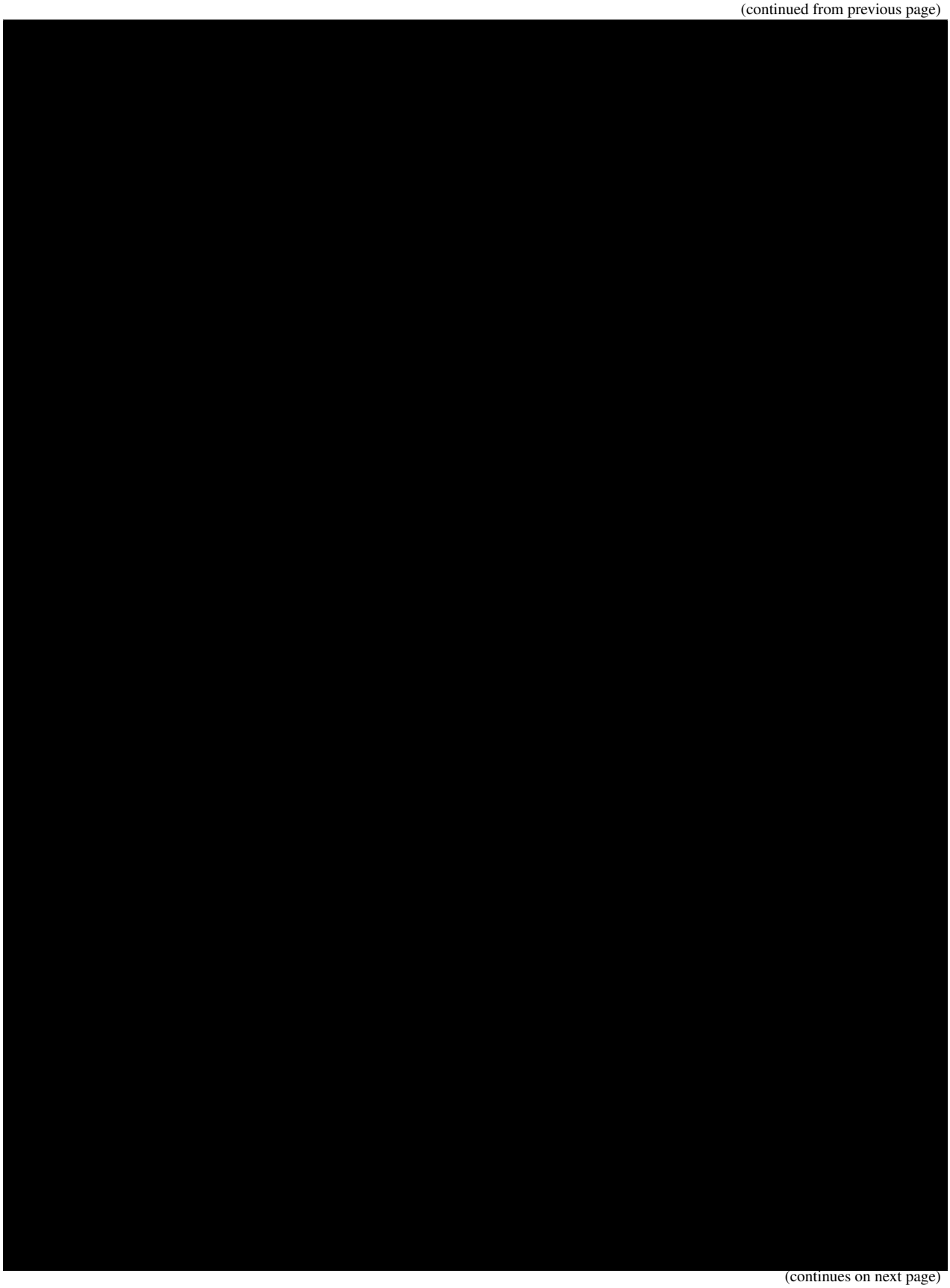
iRIC will output calculation conditions, grids, grid attributes, and boundary condition according to the solver definition file that you've created in *Creating a solver definition file* (page 6). So, the solver has to load them to coincide with the description in the solver definition file.

List 2.8 shows the source code with lines to load calculation condition, grid and boundary condition. The added lines are shown with highlight.

List 2.8: The source code with lines to load calculation condition, grid and boundary condition

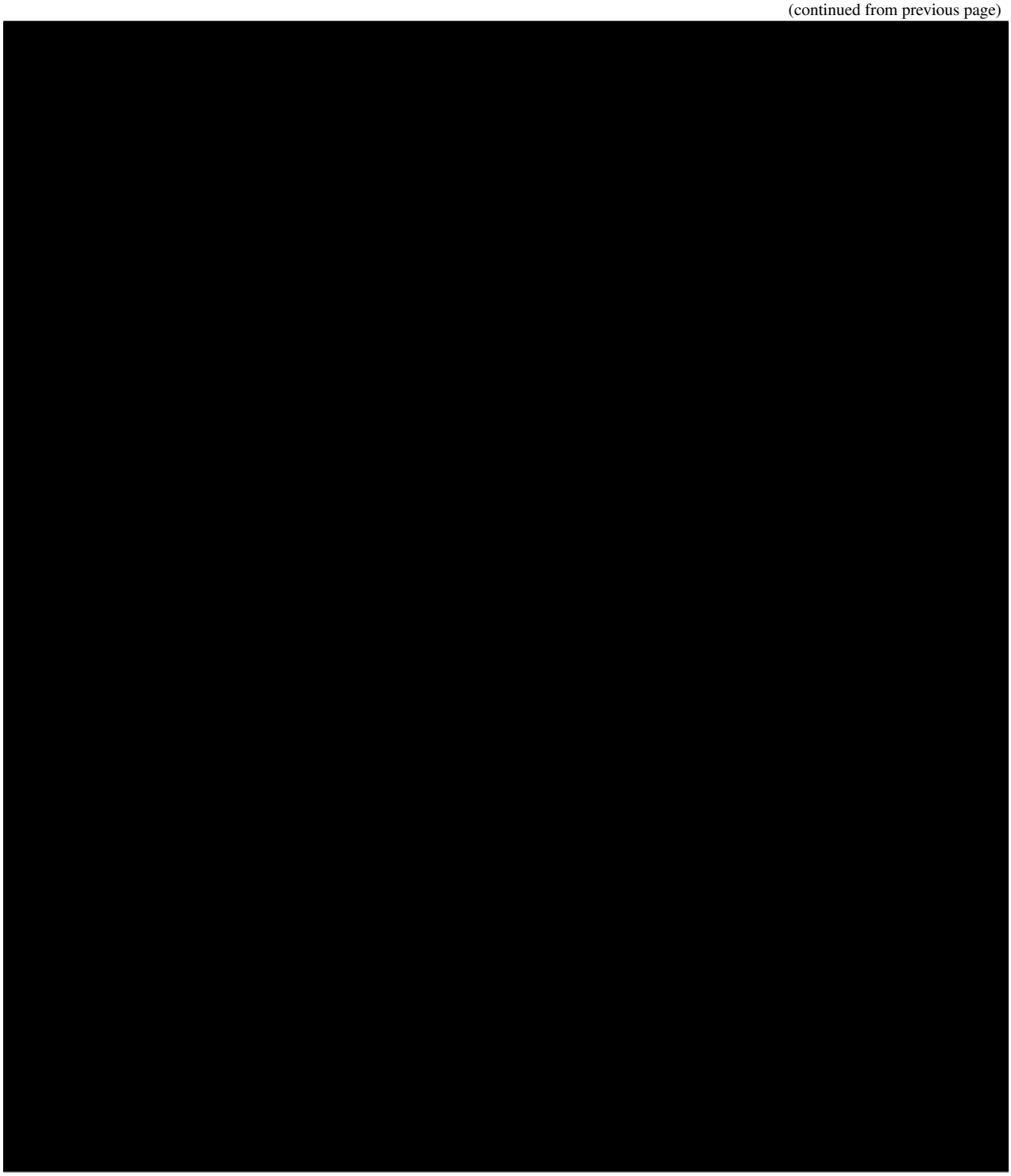
(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)



Note that the arguments passed to load calculation conditions, grid attributes and boundary conditions are the same to the [name] attributes of Items defined in *Defining calculation conditions* (page 7), *Defining Grid attributes* (page 12).

Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for the relationship between definitions of calculation condition, grid attributes, boundary conditions and the iRIClib subroutines to load them.

Refer to *Reading calculation conditions* (page 100), *Reading calculation grid* (page 102) and *Reading boundary*

conditions (page 104) for the detail of subroutines to load calculation condition, grids, and boundary conditions.

2.5.4 Adding codes to output time and calculation results

Adds codes to output time and calculation results.

When you develop a solver that is used for time-dependent flow, you have to repeat outputting time and calculation results for the number of time steps.

Before starting outputting calculation results, the solver should check whether user canceled calculation. If canceled, the solver should stop.

In solver definition files, no definition is written about the calculation results the solver output. So, you do not have to take care about the correspondence relation between solver definition file and the solver code about them.

List 2.9 shows the source code with lines to output time and calculations. The added lines are shown with highlight.

List 2.9: Source code with lines to output time and calculation results

(continues on next page)

(continued from previous page)

```
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
```

Refer to *Outputting time (or iteration count)* (page 112) and *Outputting calculation results* (page 114) for the details of the subroutines to output time and calculation results. Refer to *Outputting calculation grids (only in the case of a moving grid)* (page 113) for the details of the subroutines to output the grid coordinates in case of moving grid.

For the calculation results, some special names is named in iRIC. You should use that name for calculation results used for a certain purpose. Refer to *Calculation results* (page 219) for the special names.

2.6 Creating a solver definition dictionary file

Create a solver definition dictionary file that is used to translate the strings used in solver definition files, and shown on dialogs etc.

First, launch iRIC and perform the following:

Menu bar: [Option] (O) –> [Create/Update Translation Files] (C)

The [Definition File Translation Update Wizard] (Figure 2.18 to Figure 2.20) will open. Following the wizard, the dictionary files are created or updated.

The dictionary files are created in the folder that you created in *Creating a folder* (page 4). The files created only include the texts before translation (i. e. English strings). The dictionary files are text files, so you can use text editors to edit it. Save the dictionary files with UTF-8 encoding.

List 2.10 and List 2.11 show the example of editing a dictionary file. As the example shows, you have to add translated texts in "translation" element.

List 2.10: The Dictionary file of solver definition file (before editing)

```
1
2
3
4
```

List 2.11: The Dictionary file of solver definition file (after editing)

```
1
2
3
4
```

Figure 2.18: The [Definition File Translation Update Wizard] (Page 1)

Figure 2.19: The [Definition File Translation Update Wizard] (Page 2)

Figure 2.20: The [Definition File Translation Update Wizard] (Page 3)

You can use [Qt Linguist] for translating the dictionary file. [Qt Linguist] is bundled in Qt, and it provides GUI for editing the dictionary file. Figure 2.21 shows the [Qt Linguist]. Qt can be downloaded from the following URL:

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

When the translation is finished, switch the iRIC language from Preferences dialog, restart iRIC, and check whether the translation is complete. Figure 2.22 and Figure 2.23 shows examples of [Pre-processing Window] and [Calculation Condition] dialog after completing transtaion of dictionary.

2.7 Creating a README file

Creates a text file that explains the abstract of the solver.

Creates a text file with name "README" in the folder you created in *Creating a folder* (page 4). Save the file with UTF-8 encoding.

You should create the README file with the file names like below. When the language-specific README file does not exists, "README" file (in English) will be used.

- English: "README"
- Japanese: "README_ja_JP"

The postfix (ex. "ja_JP") is the same to that for dictionary files created in *Creating a solver definition dictionary file* (page 23).

The content of "README" will be shown in "Description" area on the [Select Solver] dialog. When you created "README", opens the [Select Solver] dialog by starting a new project, and check whether the content is shown on

Figure 2.21: The [Qt Linguist]

Figure 2.22: [Pre-processor Window] after completing translation of dictionary (Japanese mode)

Figure 2.23: The [Calculation Condition] dialog after completing translation of dictionary (Japanese mode)

that dialog.

2.8 Creating a LICENSE file

Creates a text file that explains the license information of the solver.

Creates a text file with name "LICENSE" in the folder you created in *Creating a folder* (page 4). Save the file with UTF-8 encoding.

You should create the LICENSE file with the file names like below. When the language-specific LICENSE file does not exists, "LICENSE" file (in English) will be used.

- English: "LICENSE"
- Japanese: "LICENSE_ja_JP"

The postfix (ex. "ja_JP") is the same to that for dictionary files created in *Creating a solver definition dictionary file* (page 23).

The content of "LICENSE" will be shown in "License" area on the [Select Solver] dialog. When you created "LICENSE", opens the [Select Solver] dialog by starting a new project, and check whether the content is shown on that dialog.

Figure 2.25 shows an example of the [Select Solver] dialog.

Figure 2.24: The [Select Solver] dialog

Figure 2.25: The [Select Solver] dialog

CHAPTER 3

Steps of developing a calculation result analysis program

3.1 Abstract

Calculation result analysis program is a program that reads calculation result of a soler from a CGNS file, execute analysis or modify calculation result. Analysis result or modified calculation results can be output to another CGNS file.

The steps of developing a calculation result analysis program is basically the same to that of a solver (See *Steps of developing a solver* (page 3)). The difference is that it handles multiple CGNS files.

To handle multiple CGNS files at the same time, you should use different functions than those used in *Steps of developing a solver* (page 3) (See *List of subroutines* (page 130)). The names of functions for handling multiple CGNS files ends with "_mul_f", and the first argument is the file ID. You should call "cg_iric_initread_f" instead of "cg_iric_init_f" when initializing the CGNS file to be used by iRIClib.

List 3.1 shows the source code to handle multiple CGNS files.

List 3.1: Source code that handles multiple CGNS files (abstract)

(continues on next page)

(continued from previous page)

```
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

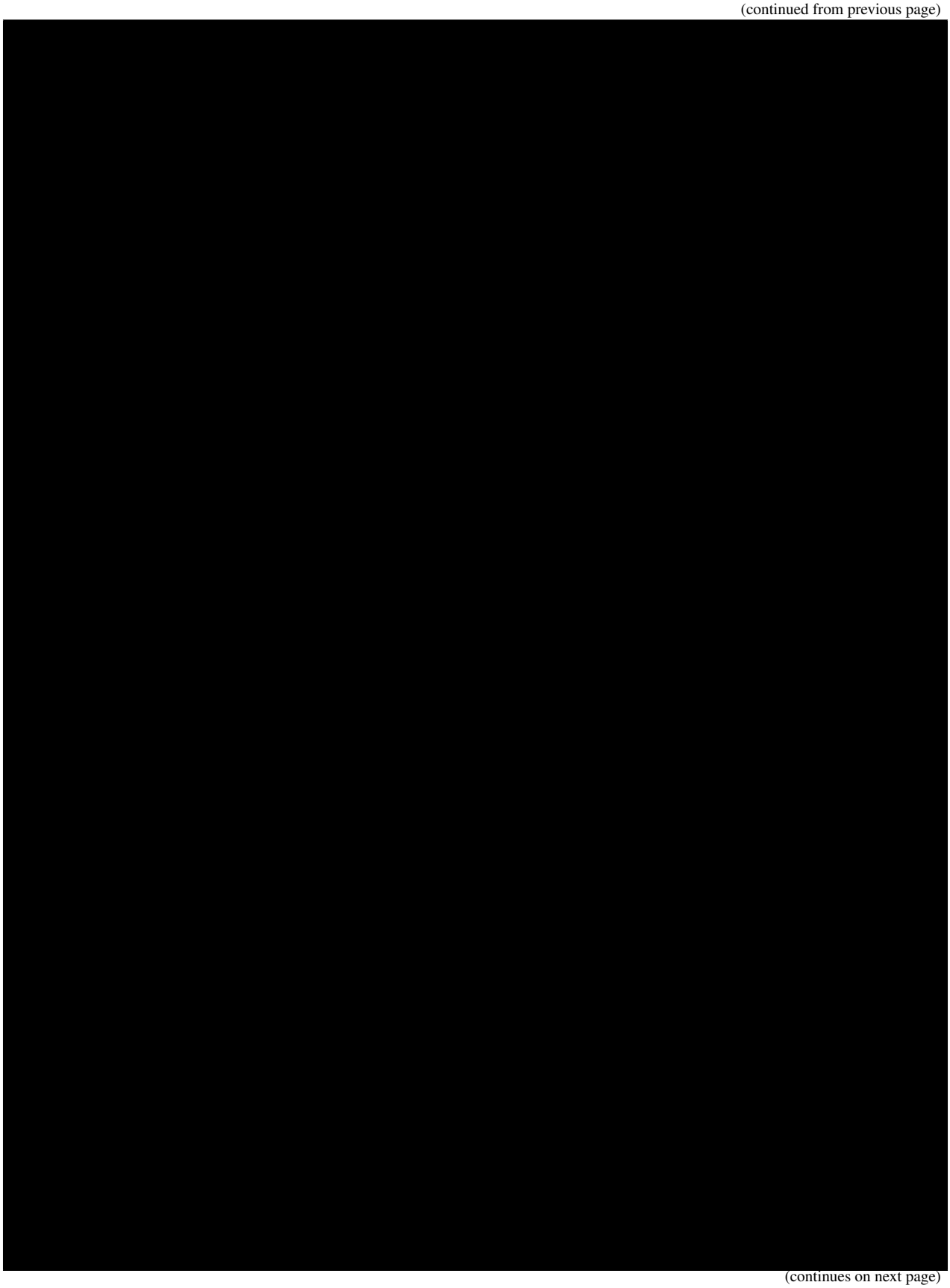
List 3.2 shows the source code the analysis program that reads calculation result from CGNS file, and executes fish habitat analysis.

List 3.2: Source code that reads calculation result from CGNS file and output analysis result

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)

CHAPTER 4

Steps of developing a grid generating program

4.1 Abstract

Grid generating program is a program that load grid creating conditions and generate a grid. The program can be used seamlessly from iRIC as one of the grid generating algorithms.

To add a grid generating program that can be used from iRIC, it is necessary to make and deploy files shown in Table 4.1.

Grid generating program developers have to create a new folder under "gridcreators" folder, and deploy files related to the new grid generating program under that.

Table 4.1: Files related to grid generating programs

File name	Description
definition.xml	Grid generating program definition file.
generator.exe	Executable module of the grid generating program. Developers can select any name.
translation_ja_JP.ts etc.	Dictionary files for a grid generating program definition file.
README	File that explains the grid generating program

Abstracts of each file are as follows:

4.1.1 definition.xml

File that defines the following information of grid generating programs:

- Basic Information
- Grid generating condition

iRIC loads definition.xml, and provides interface for creating grid generating conditions that can be used by the grid generating program. iRIC make the grid generating program available only when the solver supports the grid type that the grid generating program generate.

definition.xml should be written in English.

4.1.2 Grid Generating program

Executable module of a grid generating program. It loads grid generating condition, generate a grid, and outputs it.

Grid generating programs use grid generating data file created by iRIC, for loading and writing grid generating condition and grids.

Grid generating programs can be developed using FORTRAN, Python, C, or C++. In this chapter, a sample grid generating program is developed in FORTRAN.

4.1.3 translation_ja_JP.ts etc.

Dictionary files for a grid generating program definition file. It provides translation information for strings shown on dialogs in iRIC. Dictionary files are created one file for each language. For example, "translation_ja_JP.ts" for Japanese, "translation_ka_KR.ts" for Korean.

4.1.4 README

README is a text file that describes about the grid generating program. The content of README is shown in the "Description" area on [Select Grid Creating Algorithm] dialog].

Figure 4.1 shows the relationship between iRIC, grid generating program and related files.

Figure 4.1: The relationships between iRIC, grid generating programs, and related files

This chapter explains the steps to create the files described in this section.

4.2 Creating a folder

Create a special folder for the grid generating program you develop under "gridcreators" folder under the installation folder of iRIC. This time, please create "example" folder.

4.3 Creating a grid generating program definition file

Create a grid generating program definition file.

In grid generating program definition file, you are going to define the information shown in Table 4.2.

Table 4.2: Information defined in grid generating program definition file

Item	Description	Required
Basic Information	The grid generator name, developer name,release date etc.	Yes
Grid Generating Condition	Grid generating condition required for the argorithmn.	Yes
Error Codes	Error codes and message that correspond to the code.	

Grid generating program definition file is described in XML language. The basic grammer of XML language is explained in *XML files basics* (page 94).

In this section, we add definition information of a grid generating program in the order shown in Table 4.2.

4.3.1 Defining basic information

Define basic information of a grid generating program. Create a file with the content shown in List 4.1, and save it with name "definition.xml" under "example" folder that you created in *Creating a folder* (page 37).

List 4.1: Example grid generating program definition file that contains basic information

```
1
2
3
4
5
6
7
8
9
10
11
12
```

At this point, the structure of the grid generating program definition file is as shown in Figure 4.2.

Figure 4.2: Grid generating program definition file structure

Now make sure the grid generating file definition file is arranged correctly.

Launch iRIC. The [iRIC Start Page] dialog (Figure 4.3) is shown, so click on [New Project]. Now the [Solver Select] dialog (Figure 4.4) will open, so select "Nays2DH" in the solver list, and click on [OK]. The new project will start.

Open the [Select Grid Creating Algorithm] dialog (Figure 4.5) by processing the following action.

Menu bar: [Grid] (G) -> [Select Algorithm to Create Grid] (S)

Check that the "Sample Grid Creator" is added in the list. When you finish checking, close the dialog by clicking on [Cancel].

Figure 4.3: The [iRIC Start Page] dialog

4.3.2 Defining grid generating conditions

Define grid generating conditions. Grid generating conditions are defined in "GridGeneratingCondition" element in a grid generating program definition file. Add description of grid generating condition to the grid generating program definition file you created in *Defining basic information* (page 37), and overwrite it. Grid generating program definition file content is now as shown in List 4.2. The added part is shown with highlight.

List 4.2: Example of grid generating program definition file that now has grid generating condition definition

```
1
2
3
4
5
```

(continues on next page)

Figure 4.4: The [Select Solver] dialog

Figure 4.5: The [Select Grid Creating Algorithm] dialog

(continued from previous page)

```
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

At this point, the structure of the grid generating program definition file is as shown in Figure 4.6.

Figure 4.6: Grid generating program definition file structure

Now make sure that grid generating program definition file is arranged correctly.

Launch iRIC, and opens the [Select Grid Generating Algorithm] dialog with the same procedure in *Defining basic information* (page 37). Select "Sample Grid Creator" in the list, and click on [OK].

The [Grid Creation] dialog (Figure 4.7) will open. Now you can see that the grid generating condition items you defined are shown. When you checked, click on [Cancel] to close the dialog.

Now add one more group and add grid generating condition items. Add "Elevation Output" Tab element just under "Grid Size" Tab element. The added part is shown with highlight.

List 4.3: Example of grid generating program definition file that now has grid generating condition definition

```
1
2
3
4
5
6
7
8
9
10
11
```

(continues on next page)

Figure 4.7: The [Grid Creation] dialog

(continued from previous page)

```
12
13
14
15
16
17
```

At this Point, the structure of grid generating program definition file is as shown in Figure 4.8.

Figure 4.8: Grid generating program definition file structure

Now make sure that grid generating program definition file is arranged correctly. Do the operation you did again, to show the [Grid Creation] dialog (Figure 4.9).

Now you'll see that the new group "Elevation Output" in the group list. You'll also notice that "Value" item is enabled only when "Output" value is "Enabled".

What it comes down to is:

Figure 4.9: The [Grid Creation] dialog

- Grid generating condition group is defined with "Tab" element, and grid generating condition item is defined with "Item" element.
- The Structure under "Definition" elements depends on the condition type (i. e. Integer, Real number, functional etc.). Refer to Section *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for examples of grid generating condition items for each type.
- Dependenciy between grid generating condition items can be defined with "Condition" element. In "Condition" element, define the condition when that item should be enabled. Refer to *Example of condition to activate calculation conditions etc.* (page 70) for examples of "Condition" element.
- In this example, the calculation condition dialog shows the items as a simple list, but iRIC has feature to show items with more complex layouts, like layout with group boxes. Refer to *Example of dialog layout definition* (page 70) for more complex calculation condition page layouts.

4.3.3 Defining error codes

Define error codes of errors that occurs in grid generating program, and the messages that correspond to them. Error codes can be defined with ErrorCode elements in grid generating program definition file. Add definitions to the definition file you created, as shown in List 4.4. The added part is shown with highlight.

List 4.4: Example of grid generating program definition file that now has error codes

```
1
2
3
4
5
6
7
8
```

At this Point, the structure of grid generating program definition file is as shown in Figure 4.10. The ErrorCodes element is not required.

Figure 4.10: The grid generating program definition file structure

You can not check whether ErrorCode element is properly defined until you create a grid generating program. You are going to check it in *Adding error handling codes* (page 49).

4.4 Creating a grid generating program

Create a grid generating program. In this example we will develop a grid generating program with FORTRAN.

To develop a grid generating program that works together with iRIC, you have to make it use grid generating data file that iRIC generate, for loading grid generation conditions and outputting a grid.

The grid generating data file that iRIC generates is a CGNS file. You can use a library called iRIClib to write code for loading and writing CGNS files.

In this section, We'll explain the procedure to develop a grid generating program that load calculation data file, that iRIC generates.

Creating a grid generating program definition file (page 37) shows the input and output processing that the grid generating program do against the grid generating data file.

Table 4.3: The I/O processing flow of grid generating program

Processing
Opens grid generating data file
Initializes iRIClib
Loads grid generating condition
Outputs grid
Closes grid generating data file

In this section, we will develop a grid generating program in the following procedure:

1. Create a scelton
2. Adds grid generating data file opening and closing codes
3. Adds codes to output grid
4. Adds codes to load grid generating conditions
5. Adds codes for error handling

4.4.1 Creating a scelton

First, create a scelton of a grid generating program. Create a new file with the source code in List 4.5, and save as "sample.f90". At this point, the grid generating program does nothing.

Compile this source code. The way to compile a source code differs by the compiler. Refer to *Intel Fortran Compiler (Windows)* (page 218) for the procedure to compile using Intel Fortran Compiler and gfortran.

List 4.5: Sample grid generating program source code

Make sure that the compilation succeeds.

4.4.2 Adding grid generating data file opening and closing codes

Adds codes for opening and closing grid generating data file.

The grid generating program has to open calculation data file in the first step, and close it in the last step.

iRIC will handle the file name of grid generating data file as the first argument, so open that file.

The way to handle the number of arguments and the arguments differs by compilers. Refer to List 4.6 for the way to handle them with Intel Fortran Compiler and gfortran. In this chapter we will add codes that can be compiled using Intel Fortran Compiler.

List 4.6 shows the source code with the lines to open and close grid generating data file. The added lines are shown with highlight.

List 4.6: The source code with lines to open and close file

Compile the executable file, just like in *Creating a scelton* (page 43).

Check that the source code can be compiled successfully.

Refer to *Opening a CGNS file* (page 99), *Initializing iRIClib* (page 99) and *Closing a CGNS file* (page 129) for the details of the subroutines added in this section.

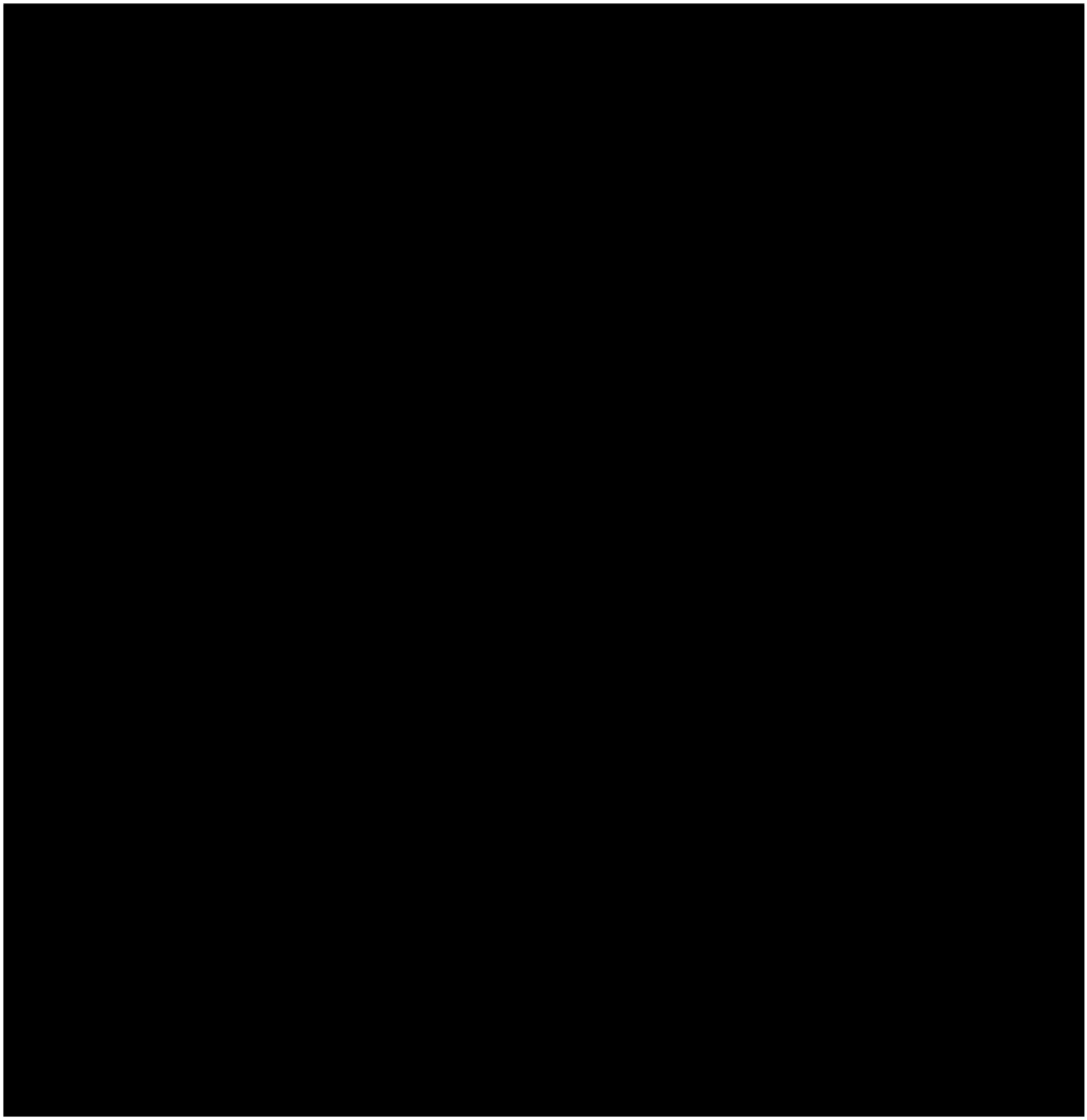
4.4.3 Adding codes to output a grid

Adds codes to output grid.

First, add codes to output a very simple grid, to check whether the program works together with iRIC successfully.

List 4.7 shows the source code with lines to output grid. The added lines are shown with highlight.

List 4.7: The source code with lines to output grid



When it was compiled successfully, copy the executable file to the folder you created in *Creating a folder* (page 37), and rename it into the name you specified as [executable] attribute in *Defining basic information* (page 37). This time, rename into "generator.exe". Copy the DLL files into that folder, that is need to run the grid generating program.

Now check whether the grid generating program can be launched from iRIC successfully.

Starts a new project with solver "Nays2DH", and select "Sample Grid Creator" as the grid generating algorithm like in *Defining basic information* (page 37). The [Grid Creation] dialog (Figure 4.11) will open.

Figure 4.11: The [Grid Creation] dialog

Click on [Create Grid], and a 10 x 10 grid will be created and loaded on the pre-processing window (Figure 4.12).

Refer to *Outputting calculation grids* (page 110) for the detail of subroutines to output grids. Note that in *Outputting calculation grids* (page 110) the subroutines to output three-dimensional grids are listed, but they can not be used in grid generating programs. In grid generating programs, only subroutines to output two-dimensional grids can be used.

4.4.4 Adding codes to load grid generating condition

Adds codes to load grid generating conditions.

iRIC will output grid generating conditions according to the grid generating program definition file. So, the grid generating program have to load them to coincide with the description in the grid generating program definition file.

List 4.8 shows the source code with lines to load grid generating condition. The added lines are shown with highlight. Note that the arguments passed to load grid generating conditions are the same to the [name] attributes of Items defined in *Defining grid generating conditions* (page 38).

When it is compiled successfully, create a grid from iRIC in the procedure same to *Adding codes to output a grid* (page 45), and the grid will be created with the condition you specified on [Grid Creation] dialog.

Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for the relation between definitions of grid generating condition and the iRIClib subroutines to load them. Refer to *Reading calculation conditions* (page 100) for the detail of subroutines to load grid generating conditions.

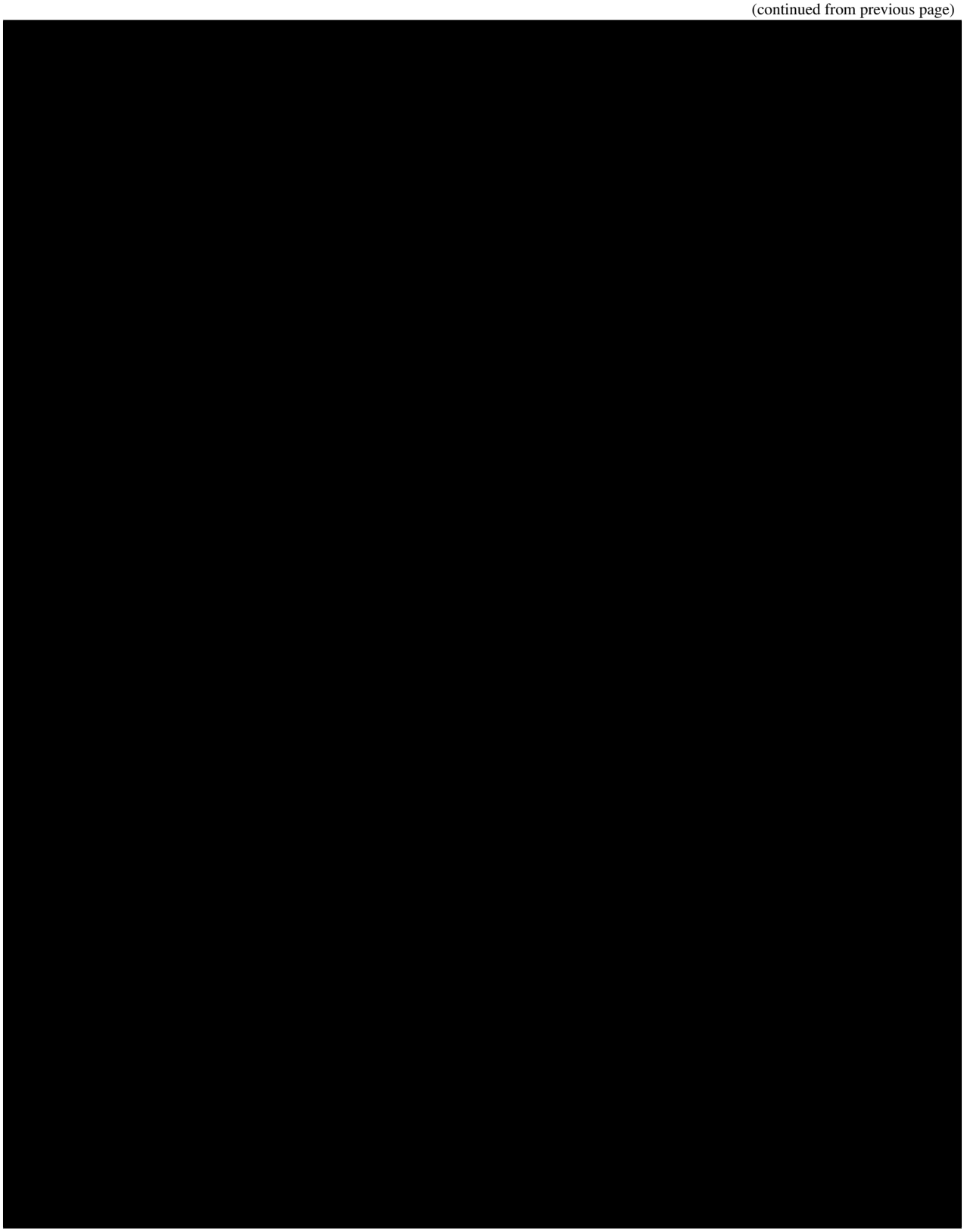
List 4.8: Source codewith lines to load grid generating conditions

```
1
2
3
```

(continues on next page)

Figure 4.12: The pre-processing window after creating grid

(continued from previous page)



4.4.5 Adding error handling codes

Adds error handling code, to support cases that grid generating conditions have some problems.

Table:numref:*gridgenerator_with_error_handling* shows the source code with lines to handle errors. The added lines are shown with highlight. With the lines added, the grid generating program will return error when the number of grid nodes exceeds 100000.

When it is compiled successfully, create a grid with the algorithm in tha same way to *Adding codes to output a grid* (page 45). Check that when you specify big imax and jmax values, the [Error] dialog (Figure 4.13) will open.

Refer to *Outputting Error code* (page 129) for the subroutines to output error codes.

List 4.9: Source code with lines to handle errors

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

Figure 4.13: The [Error] dialog

4.5 Creating a grid generating program definition dictionary file

Create a grid generating program definition dictionary file that is used to translate the strings used in grid generating program definition files, and shown on dialogs etc.

First, launch iRIC and perform the following:

Menu bar: [Option] (O) –> [Create/Update Translation Files] (C)

The [Definition File Translation Update Wizard] (Figure 4.14 to Figure 4.16) will open. Following the wizard, the dictionary files are created or updated.

Figure 4.14: The [Definition File Translation Update Wizard] (Page 1)

The dictionary files are created in the folder that you created in *Creating a folder* (page 37). The files created only include the strings before the translation (i. e. English strings). The dictionary files are text files, so you can use text editors to edit it. Save the dictionary files with UTF-8 encoding.

List 4.10 and List 4.11 show the example of editing a dictionary file. As the example shows, add translated string in "translation" element.

List 4.10: The Dictionary file of grid generating program definition file (before editing)

```
1
2
3
4
```

List 4.11: The Dictionary file of grid generating program definition file (after editing)

```
1
2
3
4
```

You can use [Qt Linguist] for translating the dictionary file. [Qt Linguist] is bundled in Qt, and it provides GUI for editing the dictionary file. Figure 4.17 shows the [Qt Linguist]. Qt can be downloaded from the following URL:

Figure 4.15: The [Definition File Translation Update Wizard] (Page 2)

Figure 4.16: The [Definition File Translation Update Wizard] (Page 3)

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

Figure 4.17: The [Qt Linguist]

When the translation is finished, switch the iRIC language from Preferences dialog, restart iRIC, and check whether the translation is complete. Figure 4.18 shows an example of [Grid Creation] dialog after completing transtaion of dictionary.

4.6 Creating a README file

Creates a text file that explains the abstract of the grid generating program.

Creates a text file with name "README" in the folder you created in *Creating a folder* (page 37). Save the file with UTF-8 encoding.

You should create the README file with the file names like below. When the language-specific README file does not exists, "README" file (in English) will be used.

- English: "README"
- Japanese: "README_ja_JP"

The postfix (ex. "ja_JP") is the same to that for dictionary files created in *Creating a grid generating program definition dictionary file* (page 49).

The content of "README" will be shown in "Description" area on the [Select Grid Creating Algorithm] dialog. When you created "README", opens the [Select Grid Creating Algorithm] dialog, and check whether the content is shown on that dialog.

Figure 4.19 shows an example of the [Select Grid Creating Algorithm] dialog.

Figure 4.18: The [Grid Creation] dialog

Figure 4.19: The [Select Grid Creating Algorithm] dialog

CHAPTER 5

About definition files (XML)

5.1 Abstract

iRIC loads definition files (solver definition files and grid generating program definition files), and provides graphic interface to create input data for the corresponding programs (solvers and grid generating programs).

5.2 Structure

Structures of solver definition files, grid generating program definition files are described in this section.

5.2.1 Solver definition file

Structure of solver definition files for a solver that uses only one calculation grids is shown in Figure 5.1, and that for a solver that uses multiple types of calculation grids is shown in Figure 5.2, respectively.

When the solver uses multiple types of grids, Solver developers should add multiple GridType elements, and defines grid structure, grid attributes, and boundary conditions under each GridType element.

An example of solver definition file for a solver that uses multiple grid types, is shown in List 5.1. In this example, boundary condition definition is dropped, because that is not required. Please pay attention that the following point is different:

- Grid structure (gridtype attribute) is not definied in SolverDefinition elemenet, but in GridType elements.

When a user creates a new project and selects a solver that bundles the solver definition shown in List 5.1, a new pre-processor in Figure 5.3 is shown.

Figure 5.1: Structure of solver definition file

Figure 5.2: Structure of solver definition files for a solver that uses multiple grid types

List 5.1: An example of solver definition file for a solver that uses multiple types of grids

5.2.2 Grid generating program definition file

Structure of grid generating program definition file is shown in Figure 5.4

5.3 Examples

5.3.1 Examples of calculation conditions, boundary conditions, and grid generating condition

Example of definitions of calculating conditions in solver definition files, grid generating condition if grid generating program definition file is shown in this section. The position to describe the definition differs like shown in Table 5.1, but the grammers are the same. Refer to *Structure* (page 55) for the whole structure of each file.

Figure 5.3: Pre-processor image after loading the solver definition file with multiple grid type definitions

Figure 5.4: Structure of grid generating program definition file

Table 5.1: Position to define definition elements

Item	Target file	Definition position
Calculation condition	Solver definition file	Under CalculationCondition element
Grid generating condition	Grid generating program definition file	Under GridGeneratingCondition element

The types of items available, are shown in Table 5.2. In this subsection, the followings are described fore each type:

- Definition example
- Example of the corresponding widget shown on calculation condition edit dialog in iRIC
- Code example to load the values in solvers (or grid generating program).

In code examples to load the values, subroutines in iRIClib are used. Please refer to *iRIClib* (page 97) to know more about iRIClib.

The examples only show the sample codes for loading, so please refer to *Creating a solver* (page 17), *Creating a grid generating program* (page 43) to see examples of whole programs.

Table 5.2: Types of calculation conditions and grid generating conditions

Type	Description	Definition
String	Input string value	Specify "string" for valueType
File name (For reading)	Input file name for reading. Users can select only files that already exist.	Specify "filename" for valueType
File name (For writing)	Input file name for writing. Users can select any file name, including those does not exists.	Specify "filename_all" for valueType
Folder name	Input folder name.	Specify "foldername" for valueType
Integer	Input arbitrary integer value.	Specify "integer" for valueType
Integer (Choice)	Select integer value from choices.	Specify "Integer" for valueType, and define choises with Enumeration elements
Real number	Input arbitrary real number value.	Specify "real" for valueType
Functional	Input pairs of (X, Y) values.	Specify "functional" for valueType and define variable and value with a Parameter element and a Value element.
Functional (with multiple values)	Input trinity of (X, Y1, Y2) values.	Specify "functional" for valueType and define one Parameter element and two Value elements.
CGNS file name	Input CGNS file name for reading. Users can select only files that already e3xists.	Specify "cgns_filename" for valueType
Calculation result in CGNS file	Select the name of calculatioin result	Specify ""result_gridNodeReal"" etc. for valueType"

String

List 5.2: Example of a string type condition definition

```
1
2
3
```

Figure 5.5: Widget example of a string type condition

List 5.3: Code example to load a string type condition (for calculation conditions and grid generating conditions)

```
1
2
3
4
```

List 5.4: Code example to load a string type condition (for boundary conditions)

```
1
2
3
4
```

File name (for reading)

List 5.5: Example of a file name type (for reading) condition definition

```
1
2
3
```

List 5.6: Code example to load a file name (for reading) type condition (for calculation conditions and grid generating conditions)

```
1
2
3
4
```

List 5.7: Code example to load a file name (for reading) type condition (for boundary conditions)

```
1
2
3
4
```

File name (for writing)

List 5.8: Example of a file name (for writing) type condition definition

```
1
2
```

(continues on next page)

Figure 5.6: Widget example of a file name (for reading) type condition

(continued from previous page)

```
3
```

List 5.9: Code example to load a file name (for writing) type condition
(for calculation conditions and grid generating conditions)

```
1
2
3
4
```

List 5.10: Code example to load a file name (for writing) type condition
(for boundary conditions)

```
1
2
3
4
```

Folder name

List 5.11: Example of a folder name type condition definition

```
1
2
3
```

Figure 5.7: Widget example of a file name type (for writing) condition

Figure 5.8: Widget example of a folder name type condition

List 5.12: Code example to load a folder name type condition (for calculation conditions and grid generating conditions)

```
1
2
3
4
```

List 5.13: Code example to load a folder name type condition (for boundary conditions)

```
1
2
3
4
```

Integer

List 5.14: Example of a integer type condition definition

```
1
2
3
```

Figure 5.9: Widget example of a integer type condition

List 5.15: Code example to load a integer type condition (for calculation conditions and grid generating conditions)

```
1
2
3
```

List 5.16: Code example to load a integer type condition (for boundary conditions)

```
1
2
3
```

Integer (Choice)

List 5.17: Example of a integer (choise) type condition definition

```
1
2
3
4
```

(continues on next page)

(continued from previous page)

Figure 5.10: Widget example of a integer (choice) type condition

List 5.18: Code example to load a integer (choise) type condition (for calculation conditions and grid generating conditions)

List 5.19: Code example to load a integer (choise) type condition (for boundary conditions)

Real number

List 5.20: Example of a real number type condition definition

Figure 5.11: Widget example of a real number type condition

List 5.21: Code example to load a real number type condition (for calculation conditions and grid generating conditions)

List 5.22: Code example to load a real number type condition (for boundary conditions)

Functional

List 5.23: Example of a functional type condition definition

```
1
2
3
4
5
6
```

Figure 5.12: Widget example of a functional type condition

List 5.24: Code example to functional type condition (for calculation conditions and grid generating conditions)

```
1
2
3
4
5
```

(continues on next page)

(continued from previous page)

```
6
7
8
9
10
```

List 5.25: Code example to functional type condition (for boundary conditions)

```
1
2
3
4
5
6
7
8
9
10
```

Functional (with multiple values)

List 5.26: Example of a functional (with multiple values) type condition definition

```
1
2
3
4
5
6
7
```

List 5.27: Code example to load a functional (with multiple values) type condition (for calculation conditions and grid generating conditions)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Figure 5.13: Widget example of a functional (with multiple values) type condition

List 5.28: Code example to load a functional (with multiple values) type condition (for boundary condition)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

CGNS file name etc.

"CGNS file name" and "Calculation result in CGNS file" is used together.

Widget to select CGNS file name can be created with valueType attribute "cgns_filename".

Widget to select calculation result in CGNS file can be created with valueType attribute "result_gridNodeReal" etc., and cgnsFile attribute that refers the name of "CGNS file name" widget.

List 5.29: Example of a CGNS file name and Calculation result in CGNS

```
1
2
3
4
5
6
```

Figure 5.14: Widget example of CGNS file name and Calculation result in CGNS

List 5.30: Code example to load CGNS file name and Calculation result in CGNS (for calculation conditions and grid generating conditions)

```
1
2
3
4
5
```

List 5.31: Code example to load CGNS file name and Calculation result in CGNS (for boundary condition)

```
1
2
3
4
5
```

5.3.2 Example of condition to activate calculation conditions etc.

Examples of conditions to activate calculation conditions, grid generating conditions, and boundary conditions are shown in this subsection. As these examples show, complex conditions can be defined using conditions with types "and" and "or".

var1 = 1

```
1
```

var1 = 1 and var2 > 3

```
1
2
3
4
```

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

```
1
2
3
4
5
6
7
```

5.3.3 Example of dialog layout definition

Simple layout

Simple layout (that only uses Item elements) example definition is shown in List 5.32, and the corresponding dialog is shown in Figure 5.15.

List 5.32: Simple layout definition example

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
```

Layout that uses Group boxes

Layout example that uses group boxes is shown in List 5.33, and the corresponsing dialog is shown in Figure 5.16.

GroupBox elements can be used to define groups of items.

Figure 5.15: Dialog for simple layout definition example

List 5.33: Layout definition example that uses group boxes

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
```

(continues on next page)

(continued from previous page)

Figure 5.16: Dialog for layout definition example that uses group boxes

Free layout

Free layout example, that uses GridLayout element, is shown in List 5.34, and the corresponsing dialog is shown in Figure 5.17.

GridLayout, HBoxLayout, VBoxLayout can be used to layout widgets freely. When using these elements for layouting, caption attributes are not used to show labels, but Label elements are used to show labels instead.

GridLayout, HBoxLayout, VBoxLayout elements can be used recursively. GroupBox element can be used inside these elements freely.

List 5.34: Free layout definition example

(continues on next page)

(continued from previous page)

Figure 5.17: Dialog for free layout definition example

5.4 Elements reference

5.4.1 BoundaryCondition

BoundaryCondition element contains boundary condition information.

Example

List 5.35: Example of BoundaryCondition definition

Attributes

Table 5.3: Attributes of BoundaryCondition

Name	Value	Required	Description
name	String	Yes	Name of element
caption	String	Yes	String to be displayed on dialog
position	Refer the table below	Yes	Definition position

Table 5.4: position values

Value	Meaning
node	Grid nodes
cell	Grid cells
edge	Grid edges

Child elements

Table 5.5: Child elements of BoundaryCondition

Name	Required	Description
Item		Definition of element

5.4.2 CalculationCondition

CalculationCondition element contains calculation condition information.

Example

List 5.36: Example of CalculationCondition definition

(continues on next page)

(continued from previous page)

```
10
11
12
```

Attributes

No attribute can be defined.

Child elements

Table 5.6: Child elements of CalculationCondition

Name	Required	Description
Tab		An element that contains information on a page (or tab) of the calculation condition input dialog

5.4.3 Condition

Condition element contains information of a condition that must be met when a certain input item of calculation conditions become active.

Example

List 5.37: Example of Condition definition 1

```
1
```

List 5.38: Example of Condition definition 2

Refer to ??? for other examples.

Attributes

Table 5.7: Attributes of Condition

Name	Value	Required	Description
conditionType	Refer to the table below	Yes	Type of condition
target	String	Depends	Name of the calculation condition to be compared with. Needs to be sconditionType Needs to be specified only if the conditionType value is none of "and", "or", "not".
value	String	Depends	Value to be compared with. Needs to be sconditionType Needs to be specified only if the conditionType value is none of "and", "or", "not".

Table 5.8: conditionType values

Value	Meaning
isEqual	is equal to
isGreaterEqual	is greater than or equal to
isGreaterThan	is greater than
isLessEqual	is less than or equal to
isLessThan	is less than
and	and
or	or
not	not

Child elements

Table 5.9: Child elements of Condition

Name	Required	Description
Condition	Depends	The condition to which the AND, OR or NOT operator is applied. Need to be specified only if the conditionType value is one of the following:

5.4.4 Definition (when used under CalculationCondition element or BoundaryCondition element)

Definition element contains definition information of calculation conditions or grid attributes or boundary conditions.

Example

List 5.39: Example of Definition definition 1

```
1
```

List 5.40: Example of Definition definition 2

```
1
2
3
4
```

Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for examples of Condition element definition.

Attributes

Table 5.10: Attributes of Definition

Name	Value	Required	Description	
valueType	Refer to the table below	Yes	Value type	
default	String		Default value	
noSort	true or false		When value "true" is specified	the table on the dialog is not sorted by Param value.

Table 5.11: valueType values

Value	Meaning
integer	Integer
real	Real number
string	String
functional	Functional type
filename	File name
filename_all	filename; even a file that currently does not exist can be specified
foldername	Folder name
result_gridNodeInteger	Select integer value calculation result defined at grid nodes
result_gridNodeReal	Select real value calculation result defined at grid nodes
result_gridCellInteger	Select integer value calculation result defined at grid cells
result_gridCellReal	Select real value calculation result defined at grid cells
result_gridEdgeIInteger	Select integer value calculation result defined at I-direction edges
result_gridEdgeIReal	Select real value calculation result defined at I-direction edges
result_gridEdgeJInteger	Select integer value calculation result defined at J-direction edges
result_gridEdgeJReal	Select real value calculation result defined at J-direction edges
result_baseIterativeInteger	Select integer value calculation result defined globally
result_baseIterativeReal	Select real value calculation result defined globally

Child elements

Table 5.12: Child elements of Definition

Name	Required	Description
Enumeration		It should be specified when solver developers wants to limit the selection of the value. It can be specified only when valueType is integer or real.
Condition		The condition that must be met when the element become active.
Param		value for horizontal axis. It can be specified only when valueType is functional.
Value		value for vertical axis. It can be specified only when valueType is functional.

5.4.5 Definition (when used under the GridRelatedCondition element)

This element contains definition information of the attributes to be defined for an input grid.

Example

List 5.41: Example of Definition definition

```
1
```

Attributes

Table 5.13: Attributes of Definition

Name	Value	Required	Description
valueType	Refer to the table below	Yes	Value type
position	Refer to the table below	Yes	Definition position
default	String		Default value.If "min" or "max" is specified, the minimum value or the maximum value, respectively, of the input geographical information will be used for an area devoid of geographical information.

Table 5.14: valueType values

Value	Meaning
integer	Integer
real	Real number
complex	Complex type

Table 5.15: position values

Value	Meaning
node	Grid nodes
cell	Grid cells

Child elements

Table 5.16: Child elements of Definition

Name	Required	Description
Dimension		It should be specified when solver developers want to define a dimension (ex. Time).
Enumeration	Depends	Specify this when you want to limit choice for value
Item		Required only if the valueType attribute value is "complex". The structure of Item element is the same to that of Item element under Boundary Condition element.

5.4.6 Dimension

Dimension element contains information that defines a dimension of an attributes to be defined for an input grid.

Example

List 5.42: Example of Dimension definition

```
1
```

Attributes

Table 5.17: Attributes of Dimension

Name	Value	Required	Description
name	String	Yes	Name of element
caption	String	Yes	String to be displayed in the [Pre-processor]
valueType	Refer to table below	Yes	Value type

Table 5.18: valueType vaslues

Value	Meaning
integer	Integer
real	Real number

Child elements

No child element can be defined.

5.4.7 Enumeration

Enumeration element contains information that defines an input option for the input item of calculation conditions or grid generating condition.

Example

List 5.43: Example of Enumeration definition

```
1
```

Refer to *Integer (Choice)* (page 64) for examples of Enumeration element definitions.

Attributes

Table 5.19: Attributes of Enumeration

Name	Value	Required	Description
value	String	Yes	A string representing a value that corresponds to caption
caption	String	Yes	String to be displayed.

Child elements

No child elements can be defined.

5.4.8 ErrorCode

ErrorCodes element contains a list of error codes.

Examples

List 5.44: Example of ErrorCode definition

```
1
```

Attributes

Table 5.20: Attributes of ErrorCode

Name	Value	Required	Description
value	Integer	Yes	Error code
caption	String	Yes	String to be displayed

Child elements

No child elements can be specified

5.4.9 ErrorCodes

ErrorCodes element contains a list of error codes.

Example

List 5.45: Example of ErrorCodes definition

Attributes

No attributes can be defined.

Child elements

Table 5.21: Child elements of ErrorCodes

Name	Required	Description
ErrorCode		Error code

5.4.10 GridGeneratingCondition

GridGeneratingCondition element contains information that defines a grid generating condition.

Example

List 5.46: Example of GridGeneratingCondition definition

Attributes

No attributes can be defined.

Child elements

Table 5.22: Child elements of GridGeneratingCondition

Name	Required	Description
Tab		An element that contains information on a page of the grid generationg condition input dialog

5.4.11 GridGeneratorDefinition

GridGeneratorDefinition element contains definition information of the grid generating program.

Example

List 5.47: Example of GridGeneratorDefinition

Attributes

Table 5.23: Attributes of GridGeneratorDefinition

Name	Value	Required	Description
name	String	Yes	Identification name of the grid generator (in alphanumeric characters only)
caption	String	Yes	Name of the grid generator (any characters can be used)
version	String	Yes	Version number, in a form such as "1.0" or "1.3.2"
copyright	String	Yes	Name of copyright owner; basically in English
release	String	Yes	Date of release, in a form such as "2010.01.01"
homepage	String	Yes	URL of the web page that provides information on the grid generator
executable	String	Yes	Filename of the executable program. (ex. GridGen.exe)
gridtype	Refer to table below	Yes	Grid type

Table 5.24: gridType values

Value	Meaning
structured2d	two-dimensional structured grid
unstructured2d	two-dimensional unstructured grid

Child elements

Table 5.25: Child elements of GridGeneratingCondition

Name	Required	Description
GridGeneratingCondition	Yes	Grid creating condition
ErrorCodes		List of error codes

5.4.12 GridLayout

GridLayout element contains information that defines the group box to be displayed in the calculation conditions input dialog or grid generating condition input dialog.

Example

Refer to *Free layout* (page 73) for an example of GridLayout element definition.

Attributes

No attributes can be defined.

Child elements

Table 5.26: Child elements of GridLayout

Name	Required	Description
Item, VBoxLayout etc.		Definitions of elements of child layouts

5.4.13 GridRelatedCondition

GridRelatedCondition element contains information that defines the list of grid attributes.

Example

List 5.48: Example of GridRelatedCondition definition

Attributes

No attributes can be defined.

Child elements

Table 5.27: Child elements of GridRelatedCondition

Name	Required	Description
Item	Yes	Grid attribute definition

5.4.14 GridType

GridType element contains a list of definition information of input grids.

Example

Please refer to List 5.1.

Attributes

Table 5.28: Attributes of GridType

Name	Value	Required	Description
gridtype	Refer to the table below	Yes	Grid type
multiple	true or false		true if two or more grids can be used

Table 5.29: gridtype values

Value	Meaning
1d	one-dimensional grid
1.5d	one-and-half dimensional grid
1.5d_withcrosssection	one-and-half dimensional grid having cross-sectional info
structured2d	two-dimensional structured grid
unstructured2d	two-dimensional unstructured grid

Child elements

Table 5.30: Child elements of GridType

Name	Required	Description
GridRelatedCondition	Yes	Grid attribute definition list

5.4.15 GridTypes

GridTypes element contains a list of definition information of input grids types.

Example

Please refer to List 5.1.

Attributes

No attributes can be defined.

Child elements

Table 5.31: Child elements of GridTypes

Name	Required	Description
GridType	Yes	Grid type

5.4.16 GroupBox

GroupBox element contains information that defines a group box to be displayed in the calculation condition input dialog or grid generating condition input dialog.

Example

List 5.49: Example of GroupBox definition

```
1
2
3
4
5
6
7
8
```

Refer to *Layout that uses Group boxes* (page 71) for an example of GroupBox element definition.

Attributes

Table 5.32: Attributes of GroupBox

Name	Value	Required	Description
caption	String	Yes	String to be displayed

Child elements

Table 5.33: Child elements of GroupBox

Name	Required	Description
Item, VBoxLayout etc.		Definitions of elements of child layouts

5.4.17 HBoxLayout

HBoxLayout element contains information that defines layout to arrange elements horizontally in the calculation condition input dialog or grid generating condition input dialog.

Example

List 5.50: Example of HBoxLayout definition

Attributes

No attributes can be defined.

Child elements

Table 5.34: Child elements of HBoxLayout

Name	Required	Description
Item, VBoxLayout etc.		Definitions of elements of child layouts

5.4.18 Item

Item element contains information that defines an input item of calculation conditions, grid generating condtions, attributes of the input grid, or .boundary conditions.

Example

List 5.51: Example of Item definition

Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for examples of Item element definitions.

Attributes

Table 5.35: Attributes of Item

Name	Value	Required	Description
name	String	Yes	Name of element
caption	String		String to be displayed on the dialog

Child elements

Table 5.36: Child elements of Item

Name	Required	Description
Definition	Yes	Definition of elements

5.4.19 Label

Label element contains information that defines a label to be displayed in the calculation condition input dialog or grid creating condition input dialog.

Example

List 5.52: Example of Label definition

```
1
```

Refer to *Free layout* (page 73) for Label element definition example.

Attributes

Table 5.37: Attributes of Label

Name	Value	Required	Description
caption	String		String to be displayed on dialog

Child elements

No child elements can be defined.

5.4.20 Param

Param element contains information that defines an argument of functional type calculation conditions or grid creating conditions.

Example

List 5.53: Example of Param definition

```
1
```

Refer to *Functional* (page 66) for Param element definition example.

Attributes

Table 5.38: Attributes of Param

Name	Value	Required	Description
caption	String	Yes	String to be displayed on dialog
valueType	Refer to table below	Yes	Data type
axislog	true or false		true when you want to make the horizontal axis log

Child elements

No child elements can be defined.

5.4.21 SolverDefinition

SolverDefinition element contains definition information of the solver.

Examples

List 5.54: Example of SolverDefinition definition

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

Attributes

Table 5.39: Attributes of SolverDefinition

Name	Value	Required	Description	
name	String	Yes	Identification name of the solver (in alphanumeric characters only)	
caption	String	Yes	Name of the solver (any characters can be used)	
version	String	Yes	Version number, in a form such as "1.0", "1.3.2"	
copyright	String	Yes	Name of copyright owner; basically in English	
release	String	Yes	Date of release, in a form such as "2010.01.01"	
homepage	String	Yes	URL of the web page that provides information on the solver	
executable	String	Yes	Filename of the executable program. (e.g.	Solver.exe)
iterationtype	Refer to table below	Yes	Unit of outputting result	
gridtype	Refer to table below	Yes	Grid type	
multiple	true or false		true if two or more grids can be used	

Table 5.40: iterationtype value

Value	Meaning
time	Results are output by time
iteration	Results are output by iteration

Table 5.41: gridtype value

Value	Meaning
1d	one-dimensional grid
1.5d	one-and-half dimensional grid
1.5d_withcrosssection	one-and-half dimensional grid having cross-sectional info
structured2d	two-dimensional structured grid
unstructured2d	two-dimensional unstructured grid

When solver developers want to update solvers, version attribute should be changed. Refer to *Notes on solver version up* (page 93) for notes on solver version up.

Child elements

Table 5.42: Child elements of SolverDefinition

Name	Required	Description
CalculationCondition	Yes	Calculation condition
GridRelatedCondition		This should be defined only when a single type of input grid is used.
GridTypes		This should be defined only when two or more types of input grids are used.

5.4.22 Tab

Tab element contains the information that defines a page of the calculation condition input dialog.

Example

List 5.55: Example of Tab definition

Attributes

Table 5.43: Attributes of Tab

Name	Value	Required	Description
caption	String	Yes	Name (Any characters can be used.)

Child elements

Table 5.44: Child elements of Tab

Name	Required	Description
Item, VBoxLayout etc.	Yes	Definitions of elements of child layouts

5.4.23 Value

Value element contains information that defines a value of functional type calculation conditions or grid creating conditions.

Example

List 5.56: Example of Value definition

Refer to *Functional* (page 66), for Value element definition example.

Attributes

Table 5.45: Attributes of Value

Name	Value	Required	Description
caption	String	Yes	String to be displayed on dialog
valueType	Refer to table below	Yes	Data type
name	String		Identification name (in alphanumeric characters only). It is required only when the condition has multiple values.
axis	Refer to table below		Specify on which side to show Y-axis.
axislog	true or false		true when you want to make the vertical axis log
axisreverse	true or false		true when you want to reverse the vertical axis
span	true or false		true when you want to define values at spans
step	true or false		true when you want to show the chart as bar chart
hide	true or false		true when you want to hide the data from chart

Table 5.46: valueType value

Value	Meaning
integer	Integer
real	Real number

Table 5.47: axis value

Value	Meaning
left	Use Y-axis on left side
right	Use Y-axis on right side

Child elements

No child elements can be defined.

5.4.24 VBoxLayout

VBoxLayout element contains information that defines layout to arrange elements vertically in the calculation condition input dialog or grid generating condition input dialog.

Example

List 5.57: Example of VBoxLayout definition

```
1
2
3
4
5
```

(continues on next page)

(continued from previous page)

Attributes

No attributes can be defined.

Child elements

Table 5.48: Child elements of VBoxLayout

Name	Required	Description
Item, VBoxLayout etc.		Definitions of elements of child layouts

5.5 Notes on solver version up

When you update the solver you developed, you have to modify not only solver source code but also solver definition file. When you modify solver definition files you have to note the followings:

- You **must not** edit "name" attribute of SolverDefinition element. When the "name" attribute is changed, iRIC regard the solver as a completely different solver from the older version, and any project files that are created for the older version become impossible to open with the new solver.
- You **should** modify the "caption" attribute of SolverDefinition element. "caption" element is an arbitrary string that is used to display the solver name and version information, so you should input "Sample Solver 1.0", "Sample Solver 3.2 beta", "Sample Solver 3.0 RC1" as caption value for example. The caption value can be set independent from "version" attribute.
- You **must** modify the "version" attribute following the policy in Table 5.49.

Version number consists of several numbers joined with ".". The numbers are called "Major number", "Minor number", and "Fix number" for each. Fix number can be omitted.

Table 5.49: Elements of version number to increment

Element to increment	Condition to increment	Exmaple
Major number	When you changed a big modification so that the grid, calculation condition you created with older version will not be compatible to the new solver.	2.1 –> 3.0
Minor number	When you changed a small modification to calculation condition and grid. When a old project file that was created for an older solver is loaded, the default values are used for the new conditions, and that will cause no problem.	2.1 –> 2.2
Fix number	When you fixed bugs or changed inner algorithm. No modification is made to the interface (i. e. grid and calculation condition) is made.	2.1 –> 2.1.1

In iRIC, project files compatibility is like the following:

- Project files with different major number are not compatible.
- Project files with same major number and smaller minor number are compatible.
- Project files with same major number, same minor number and different fix number are compatible.

Figure 5.18 shows the examples of compatibility with different solver version numbers.

Figure 5.18: Examples of compatibility of project files with various version numbers

The basic policy is shown in Table 5.49, but in the last, solver developers should judge which number to increment, taking account of compatibility.

When you deploy multiple versions of a same solver in one environment, create multiple folders under "solvers" folder with different names, and deploy files related to each version under them. Folder names can be selected independent of solver names.

5.6 XML files basics

In this section, the basics of XML file format are described. XML file format is adopted as file format for solver definition file and grid generating program definitioin file.

5.6.1 Defining Elements

Element start tag is described with "<" and ">".

Element end tag is described with "</" and ">".

List 5.58 shows an example of Item element definition.

List 5.58: Example of Item element

An element can have the followings:

- Child element
- Attributes

An element can have multiple child elements that have the same name. On the other hand, an element can have only one attribute for each name. List 5.59 shows an example of a definition of Item element with two "Subitem" child elements and "name" attribute.

List 5.59: Example of Item element

An element that do not have a child element can be delimited with "<" and "/>". For example, List 5.60 and List 5.61 are processed as the same data by XML parsers.

List 5.60: Example of item without a child element

List 5.61: Example of item without a child element

5.6.2 About tabs, spaces, and line breaks

In XML files, tabs, spaces, and line breaks are ignored, so you can add them freely to make XML files easier to read. Please note that spaces in attribute values are not ignored. Elements in List 5.62, List 5.63, List 5.64 are processed as the same data by XML parsers.

List 5.62: Example of element

List 5.63: Example of element

List 5.64: Example of element

5.6.3 Comments

In XML files, strings between "<!–" and "–>" are treated as comments. List 5.65 shows an example of a comment.

List 5.65: Example of comment

CHAPTER 6

iRIClib

6.1 What is iRIClib?

iRIClib is a library for interfacing a river simulation solver with iRIC.

iRIC uses a CGNS file for input/output to/from solvers and grid generating programs. Input-output subroutines for CGNS files are published as an open-source library called cgnslib (see *Information on CGNS file and CGNS library* (page 220)). However, describing the necessary input-output directly using cgnslib would require complicated processing description.

Therefore, the iRIC project offers iRIClib as a library of wrapper subroutines that makes possible the abbreviated description of input-output processing which is frequently used by solvers that work together with iRIC. With these subroutines, input-output processing of a solver that performs calculation using a single structured grid can be described easily.

Note that iRIClib does not offer subroutines necessary for a solver that uses multiple grids or an unstructured grid. In case of such solvers, it is necessary to use cgnslib subroutines directly.

This chapter describes the groups of subroutines included in iRIClib, and examples of using them, along with compilation procedures.

6.2 Languages supported by iRIClib

iRIClib is supported for the following languages:

- FORTRAN
- C/C++
- Python

In this manual examples are described with FORTRAN mainly.

In this section, how to use iRIClib from FORTRAN, C/C++, and Python are briefly explained.

Please note that arguments and returned values depends on the language. Please refer to the subsections for each functions in *Reference* (page 129) about the format to use the functions from each languages.

6.2.1 FORTRAN

Call iRIClib functions after including the header, like shown in List 6.1.

List 6.1: Example of using iRIClib from FORTRAN

```
1
2
3
```

6.2.2 C/C++

Call iRIClib functions after including the header, like shown in List 6.2.

List 6.2: Example of using iRIClib from C/C++

```
1
2
3
4
```

6.2.3 Python

Call iRIClib functions after importing iric module, like shown in List 6.3.

List 6.3: Example of using iRIClib from Python

```
1
2
3
```

6.3 How to read this chapter

In this chapter, first *Overview* (page 98) explains what kinds of information input/output iRIC assumes a solver to perform, and what subroutines are available for each kind of processing. First, read Section to understand the general concept of iRIClib.

Since *Overview* (page 98) gives only an outline of subroutines, see *Reference* (page 129) for detailed information, such as lists of arguments for those subroutines.

6.4 Overview

This section provides an overview of iRIClib.

6.4.1 Processes of the program and iRIClib subroutines

The I/O processings in solvers and grid generating programs are shown in Table 6.1 and Table 6.2 .

Table 6.1: I/O processings of solvers

Process
Opens a CGNS file
Initializes iRIClib
Sets up options
Reads calculation conditions
Reads grids
Reads boundary conditions
Reads geographic data (only when needed)
Outputs grids (only in cases when grid creation or re-division is performed)
Outputs time (or iteration count)
Outputs grids (only in cases when grid moves)
Outputs calculation results
Closes a CGNS file

Table 6.2: I/O processings of a grid generating program

Process
Opens a CGNS file
Initializes iRIClib
Reads grid generating condition
Outputs error code
Outputs grid
Closes CGNS File

6.4.2 Opening a CGNS file

Open a CGNS file, read it in and make it into a writable state. The subroutine is defined in cgnslib.

Table 6.3: Subroutine to use

Subroutine	Remarks
cg_open_f	Opens a CGNS file

6.4.3 Initializing iRIClib

Prepares the CGNS file that has been opened for use by iRIClib. After the CGNS file is opened, this should be executed.

When you add calculation result to the CGNS file, open the CGNS file with CG_MODE_MODIFY mode, and initialize using "cg_iric_init_f".

When you just read grid and calculation result from CGNS file, open the CGNS file with CG_MODE_READ mode, and initialize using "cg_iric_initread_f".

Table 6.4: Subroutine to use

Subroutine	Remarks
cg_iric_init_f	Initialize the internal variables that are used for reading and modifying the opened CGNS file.
cg_iric_initread_f	Initialize the internal variables that are used for reading the opened CGNS file.

6.4.4 Set up options

Sets up options about the solver.

The options that can be set up currently are as follows:

- IRIC_OPTION_CANCEL: The solver detects cancel request by calling iric_check_cancel_f.

Table 6.5: Subroutines to use

Subroutine	Remarks
iric_initoption_f	Set up solver option

6.4.5 Reading calculation conditions

Reads calculation conditions from the CGNS file.

Table 6.6: Subroutines to use

Subroutine	Remarks
cg_iric_read_integer_f	Reads an integer calculation-condition value
cg_iric_read_real_f	Reads a double-precision real calculation-condition value
cg_iric_read_realsingle_f	Reads a single-precision real calculation-condition value
cg_iric_read_string_f	Reads a string calculation-condition value
cg_iric_read_functionalsize_f	Checks the size of a functional-type calculation condition
cg_iric_read_functional_f	Reads functional calculation condition data in double-precision real type
cg_iric_read_functional_realsingle_f	Reads functional calculation condition data in single-precision real type
cg_iric_read_functionalwithname_f	Reads functional calculation condition data (with multiple values)

For reading calculation condition data other than in functional type, a subroutine reads a single calculation condition. An example of reading an integer calculation condition value is shown in List 6.4.

List 6.4: Example of source code to read calculation conditions

(continues on next page)

(continued from previous page)

```
15
16
17
18
19
20
21
```

In contrast, for getting functional-type calculation conditions, it is necessary to use two subroutines: cg_iric_read_functionalsize_f and cg_iric_read_functional_f. An example of getting functional-type calculation condition data is shown in List 6.5.

List 6.5: Example of source code to read functional-type calculation conditions

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

Refer to *Examples of calculation conditions, boundary conditions, and grid generating condition* (page 58) for examples of codes to load calculation conditions (or grid generating conditions).

6.4.6 Reading calculation grid

Reads a calculation grid from the CGNS file. iRIClib offers subroutines for reading structured grids only.

Table 6.7: Subroutine to use

Subroutine	Remarks
cg_iric_gotogridcoord2d_f	Makes preparations for reading a 2D structured grid
cg_iric_getgridcoord2d_f	Reads a 2D structured grid
cg_iric_gotogridcoord3d_f	Makes preparations for reading a 3D structured grid
cg_iric_getgridcoord3d_f	Reads a 3D structured grid
cg_iric_read_grid_integer_node_f	Reads the integer attribute values defined for grid nodes
cg_iric_read_grid_real_node_f	Reads the double-precision attribute values defined for grid nodes
cg_iric_read_grid_integer_cell_f	Reads the integer attribute values defined for cells
cg_iric_read_grid_real_cell_f	Reads the double-precision attribute values defined for cells
cg_iric_read_complex_count_f	Reads the number of groups of complex type grid attribute
cg_iric_read_complex_integer_f	Reads the integer attribute values of complex type grid attribute
cg_iric_read_complex_real_f	Reads the double precision attribute values of complex type grid attribute
cg_iric_read_complex_realsingle_f	Reads the single precision attribute values of complex type grid attribute
cg_iric_read_complex_string_f	Reads the string attribute values of complex type grid attribute
cg_iric_read_complex_functionalsize_f	Checks the size of a functional-type attribute of complex type grid attribute
cg_iric_read_complex_functional_f	Reads functional attribute data of complex type grid attribute
cg_iric_read_complex_functionalwithname_f	Reads functional attribute of complex type grid attribute (with multiple values)
cg_iric_read_complex_functional_realsingle_f	Reads functional attribute data of complex type grid attribute
cg_iric_read_grid_complex_node_f	Reads the complex attribute values defined at grid nodes
cg_iric_read_grid_complex_cell_f	Reads the complex attribute values defined at grid cells
cg_iric_read_grid_functionaltimesize_f	Reads the number of values of dimension "Time" for functional grid attribute
cg_iric_read_grid_functionaltime_f	Reads the values of dimension "Time" for functional grid attribute
cg_iric_read_grid_functionaldimensionsize_f	Reads the number of values of dimension for functional grid attribute
cg_iric_read_grid_functionaldimensioninteger_f	Reads the values of integer dimension for functional grid attribute
cg_iric_read_grid_functionaldimensionreal_f	Reads the values of double-precision dimension for functional grid attribute
cg_iric_read_grid_functional_integer_node_f	Reads the values of functional integer grid attribute with dimension "Time" definied at grid nodes.
cg_iric_read_grid_functional_real_node_f	Reads the values of functional double-precision grid attribute with dimension "Time" definied at grid nodes.
cg_iric_read_grid_functional_integer_cell_f	Reads the values of functional integer grid attribute with dimension "Time" definied at grid cells.
cg_iric_read_grid_functional_real_cell_f	Reads the values of functional double-precision grid attribute with dimension "Time" definied at grid cells.

The same subroutines for getting attributes such as cg_iric_read_grid_integer_node_f can be used both for two-dimensional structured grids and three-dimensional structured grids.

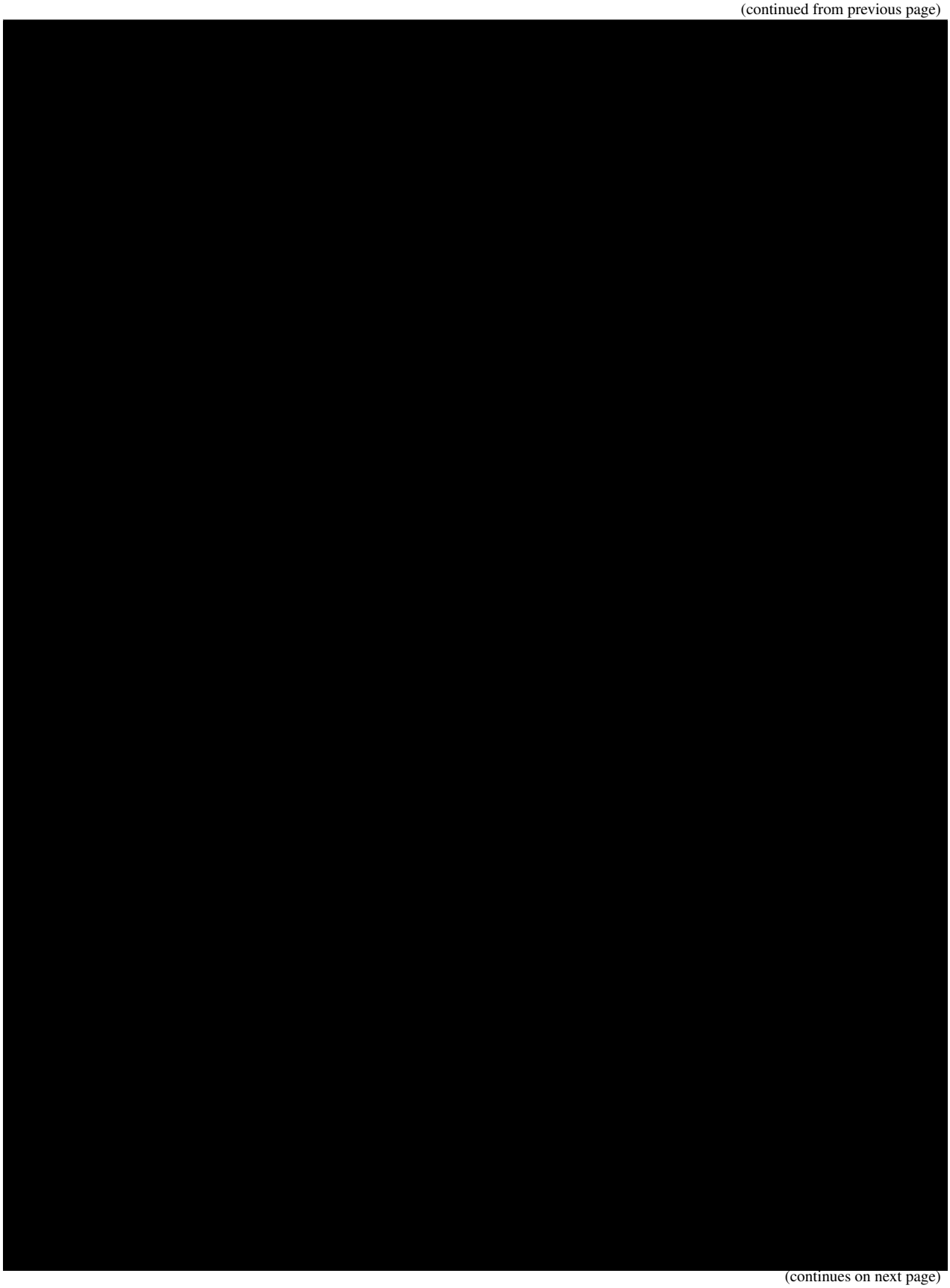
An example description for reading a two-dimensional structured grid is shown in List 6.6.

List 6.6: Example of source code to read a grid

```
1
2
3
4
5
6
```

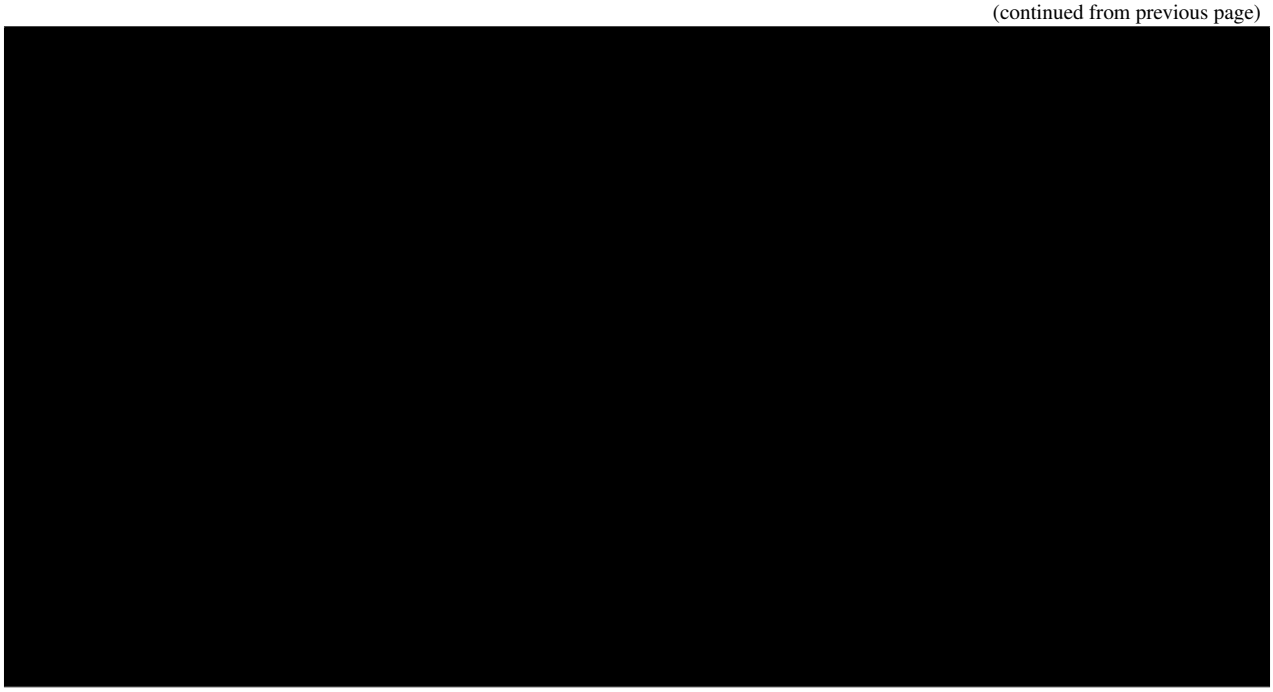
(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)



Processing for a three-dimensional grid can be described in the same manner.

6.4.7 Reading boundary conditions

Reads boundary conditions from CGNS file.

Table 6.8: Subroutine to use

Subroutine	Remarks
cg_iric_read_bc_count_f	Reads the number of boundary condition
cg_iric_read_bc_indicessize_f	Reads the number of nodes (or cells) where boundary condition is assigned.
cg_iric_read_bc_indices_f	Reads the indices of nodes (or cells) where boundary condition is assigned.
cg_iric_read_bc_integer_f	Reads a integer boundary condition value
cg_iric_read_bc_real_f	Reads a double-precision real boundary condition value
cg_iric_read_bc_realsingle_f	Reads a single-precision real boundary condition value
cg_iric_read_bc_string_f	Reads a string-type boundary condition value
cg_iric_read_bc_functionalsize_f	Reads a functional-type boundary condition value
cg_iric_read_bc_functional_f	Reads a functional-type boundary condition value
cg_iric_read_bc_functionalwithname_f	Reads a functional-type boundary condition value (with multiple values)

You can define multiple boundary conditions with the same type, to one grid. For example, you can define multiple inflows to a grid, and set discharge value for them independently.

List 6.7 shows an example to read boundary conditions. In this example the number of inflows is read by cg_iric_read_bc_count_f first, memories are allocated, and at last, the values are loaded.

The name of boundary condition user specifys on iRIC GUI can be loaded using cg_iric_read_bc_string_f. Please refer to 6.4.48 for detail.

List 6.7: Example of source code to read boundary conditions

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
↪ier)
```

(continues on next page)

(continued from previous page)

6.4.8 Reading geographic data

Reads geographic data that was imported into project and used for grid generation.

This function is used when you want to read river survey data or polygon data in solvers directly. The procedure of reading geographic data is as follows:

1. Reads the number of geographic data, the file name of geographic data etc. from CGNS file.
2. Open geographic data file and read data from that.

Table 6.9: Subroutine to use

Subroutine	Remarks
cg_iric_read_geo_count_f	Reads the number of geographic data
cg_iric_read_geo_filename_f	Reads the file name and data type of geographic data
iric_geo_polygon_open_f	Opens the geographic data file that contains polygon data
iric_geo_polygon_read_integervalue_f	Reads the value of polygon data as integer
iric_geo_polygon_read_realvalue_f	Reads the value of polygon datas double precision real
iric_geo_polygon_read_pointcount_f	Reads the number of polygon vertices
iric_geo_polygon_read_points_f	Reads the coorinates of polygon vertices
iric_geo_polygon_read_holecount_f	Reads the number of holes in the polygon
iric_geo_polygon_read_holepointcount_f	Reads the number of vertices of hole polygon
iric_geo_polygon_read_holepoints_f	Reads the coordinates of hole polygon vertices
iric_geo_polygon_close_f	Closes the geographic data file
iric_geo_riversurvey_open_f	Opens the geographic data file that contains river survey data
iric_geo_riversurvey_read_count_f	Reads the number of the crosssections in river survey data
iric_geo_riversurvey_read_position_f	Reads the coordinates of the crosssection center point
iric_geo_riversurvey_read_direction_f	Reads the direction of the crosssection as normalized vector
iric_geo_riversurvey_read_name_f	Reads the name of the crosssection as string
iric_geo_riversurvey_read_realname_f	Reads the name of the crosssection as real number
iric_geo_riversurvey_read_leftshift_f	Reads the shift offset value of the crosssection
iric_geo_riversurvey_read_altitudecount_f	Reads the number of altitude data of the crosssection
iric_geo_riversurvey_read_altitudes_f	Reads the altitude data of the crosssection
iric_geo_riversurvey_read_fixedpointl_f	Reads the data of left bank extension line of the crosssection
iric_geo_riversurvey_read_fixedpointr_f	Reads the data of right bank extension line of the crosssection
iric_geo_riversurvey_read_watersurfaceelevation_f	Reads the water elevation at the crosssection
iric_geo_riversurvey_close_f	Closes the geographic data file

List 6.8 shows an example of reading polygon. List 6.9 shows an example of reading river survey data.

List 6.8: Example source code of reading polygon

(continues on next page)

(continued from previous page)

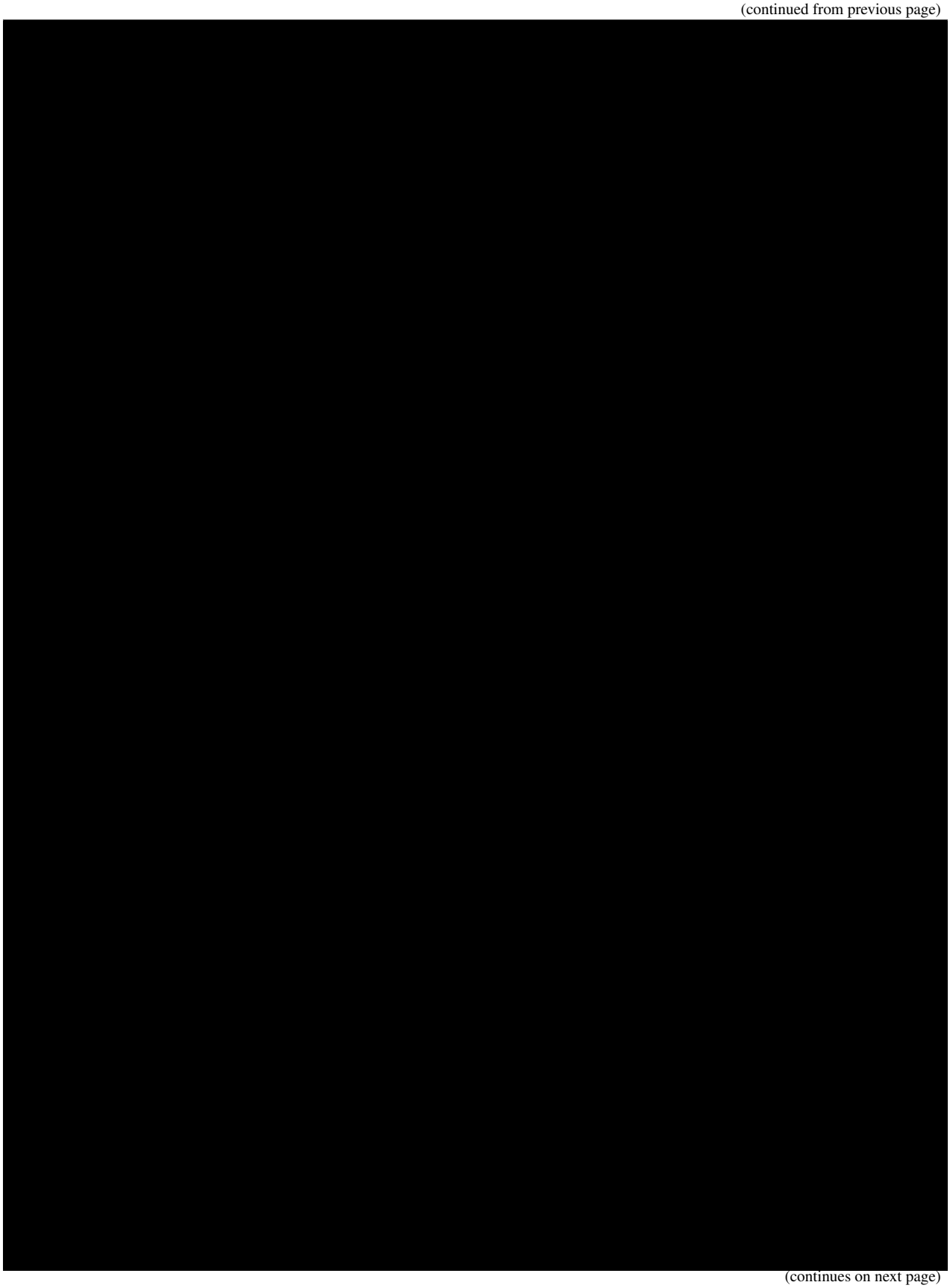
```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
```

List 6.9: Example source code of reading river survey data

```
1
2
```

(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)

6.4.9 Outputting calculation grids

Outputs the calculation grid to the CGNS file.

Unlike ordinary solvers that simply read calculation grids from the CGNS file, these subroutines are to be used in a particular kind of solver in which a grid is created on the solver side or a three-dimensional grid is generated from a two-dimensional grid.

Grid creating program always uses these subroutines.

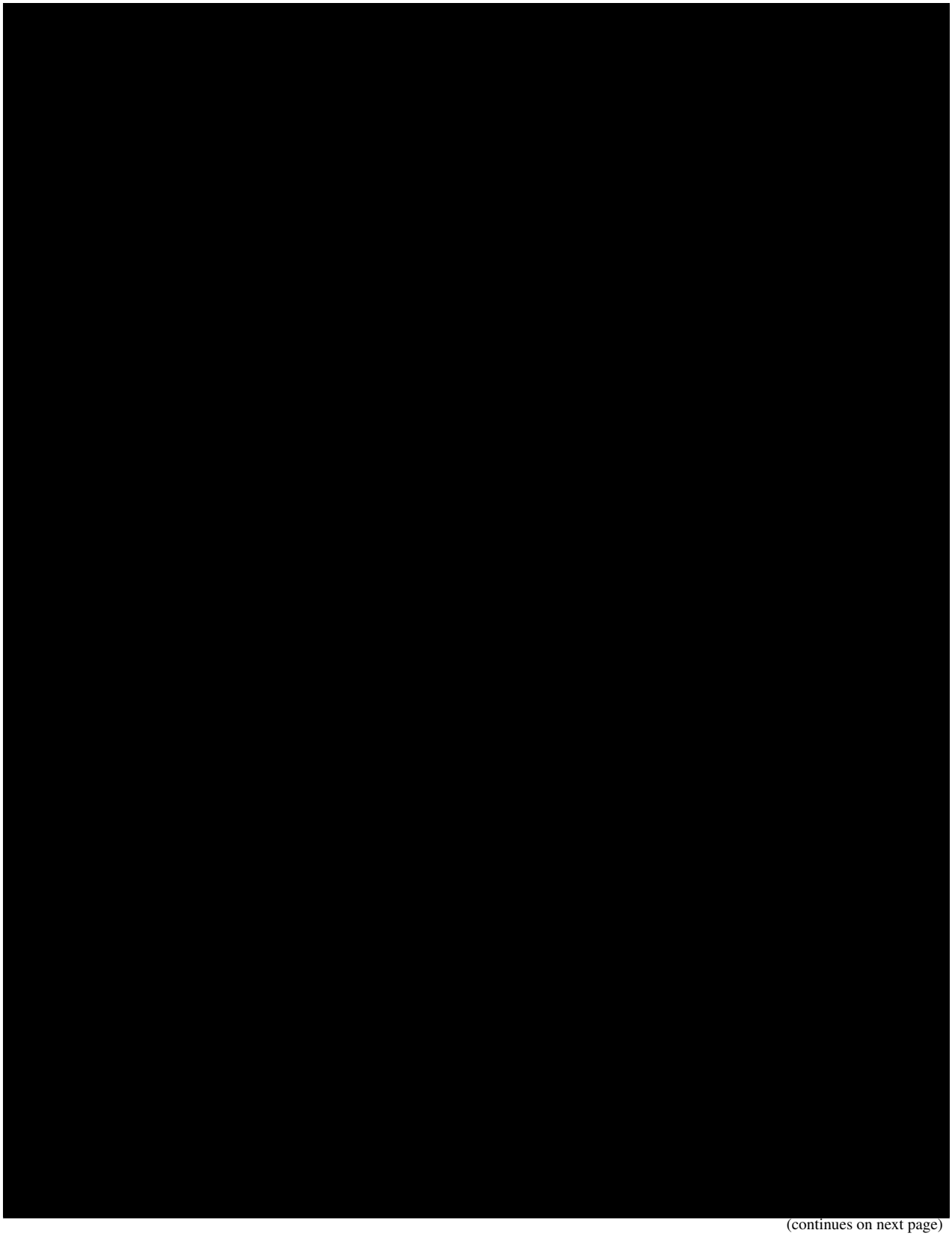
The subroutines here should be used when a solver output the grid in the initial state. When you want to output the grid shape modified after starting calculation, use the subroutines described in *Outputting calculation grids (only in the case of a moving grid)* (page 113).

Table 6.10: Subroutines to use

Subroutine	Remarks
cg_iric_writegridcoord1d_f	Outputs a one-dimensional structured grid
cg_iric_writegridcoord2d_f	Outputs a two-dimensional structured grid
cg_iric_writegridcoord3d_f	Outputs a three-dimensional structured grid
cg_iric_write_grid_real_node_f	Outputs a grid node attribute with real number value
cg_iric_write_grid_integer_node_f	Outputs a grid node attribute with integer value
cg_iric_write_grid_real_cell_f	Outputs a grid cell attribute with real number value
cg_iric_write_grid_integer_cell_f	Outputs a grid cell attribute with integer value

List 6.10 shows an example of the procedure of reading a two-dimensional grid, dividing it to generate a three-dimensional grid, and then outputting the resulting grid.

List 6.10: Example of source code to output a grid



(continues on next page)

(continued from previous page)

```
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
```

6.4.10 Outputting time (or iteration count)

Outputs the timestamp information or the iteration count to the CGNS file.

Be sure to perform this before outputting the calculation grid or calculation results.

Also note that the time and iteration-count information cannot be output at the same time. Output either, not both.

Table 6.11: Subroutines to use

Subroutine	Remarks
cg_iric_write_sol_time_f	Outputs time
cg_iric_write_sol_iteration_f	Outputs iteration count

List 6.11 shows an example of source code to output timestamp information.

List 6.11: Example of source code to output time

```
1
2
3
4
5
6
7
8
9
```

(continues on next page)

(continued from previous page)

6.4.11 Outputting calculation grids (only in the case of a moving grid)

Outputs the calculation grid to the CGNS file.

If the grid shape does not change in the calculation process, this output is not necessary.

Before outputting the calculation grid at a specific time, be sure to output the time (or iteration count) information as described in *Outputting time (or iteration count)* (page 112).

The subroutines described in this section should be used for outputting a calculation grid only when the grid shape is changed in the course of calculation. When outputting a grid in the following cases, use the subroutines described in *Outputting calculation grids* (page 110).

- A new grid has been created in the solver.
- A grid of different number of dimensions or a grid having a different grid node count has been created by performing re-division of the grid or the like.
- A grid is created in the grid generating program

Table 6.12: Subroutines to use

Subroutine	Remarks
cg_iric_write_sol_gridcoord2d_f	Outputs a two-dimensional structured grid
cg_iric_write_sol_gridcoord3d_f	Outputs a three-dimensional structured grid

List 6.12 shows an example of outputting a two-dimensional structured grid after starting calculation.

List 6.12: Example of source code to output grids after starting calculation

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

6.4.12 Outputting calculation results

Outputs the calculation results to the CGNS file.

The calculation result types that iRIClib can output are as follows:

- One value for each time step
- Value defined at grid nodes
- Value defined at grid cells

- Value defined at grid edges
- Value defined at particles
- Value defined at particles (groups supported)
- Value defined at polygon or polydata

In case of outputting every types, you have to use the functions in Table 6.13 and Table 6.14.

Please refer to *One value for each time step* (page 116) to *Value defined at polygons or polylines* (page 125) for detailed procedure to output each types.

Table 6.13: Subroutines to use before and after outputting calculation result for each timestep

Subroutine	Remarks
iric_check_cancel_f	Checks whether user canceled solver execution
iric_check_lock_f	Checks whether the CGNS file is locked by GUI
iric_write_sol_start_f	Inform the GUI that the solver started outputting result
iric_write_sol_end_f	Inform the GUI that the solver finished outputting result
cg_iric_flush_f	Flush calculation result into CGNS file

Table 6.14: Subroutines to output time and iteration count

Subroutine	Remarks
cg_iric_write_sol_time_f	Outputs time
cg_iric_write_sol_iteration_f	Outputs iteration count

Note: Vector quantities and scalar quantities

In iRIClib, the same subroutines are used to output vector quantities and scalar quantities.

When outputting vector quantities, output each component with names like "VelocityX" and "VelocityY".

Please note that if you use names whose last character is "X", "Y", or "Z", the value is not loaded properly by GUI, and user can not visualize the value. You can use lower case letters "x", "y", or "z" instead.

Note: Special names for calculation results

For calculation results, iRIC defines special names, and when you want to output calculation result for certain purposes, you should use those names. Refer to *Calculation results* (page 219) for those names.

Note: Grid nodes, grid cells, and grid edges

For grid related values, now iRIClib has function to output values defined at grid nodes, grid cells and grid edges.

Basically, please select the functions so that you can output the values at the position where the variable is defined in your solver.

There is an exception: Please output vector quantities at grid nodes. If you output vector quantities at grid cells, iRIC GUI can not visualize arrows, streamlines or particles for that value.

One value for each time step

When you output one value for each time step, please use the functions in Table 6.15.

List 6.13 shows an example of the process to output value for each time step.

Table 6.15: Subroutines to use for outputting result value that have one value for each time step

Subroutine	Remarks
cg_iric_write_sol_baseiterative_integer_f	Outputs integer-type calculation results
cg_iric_write_sol_baseiterative_real_f	Outputs double-precision real-type calculation results
cg_iric_write_sol_baseiterative_string_f	Outputs string-type calculation results

List 6.13: Example source code (One value for each time step)

(continues on next page)

(continued from previous page)

```
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
```

Value defined at grid nodes

When you output value defined at grid nodes, please use the functions in Table 6.16.

List 6.14 shows an example of the process to output value defined at grid nodes.

Table 6.16: Subroutines to use for outputting result defined at grid nodes

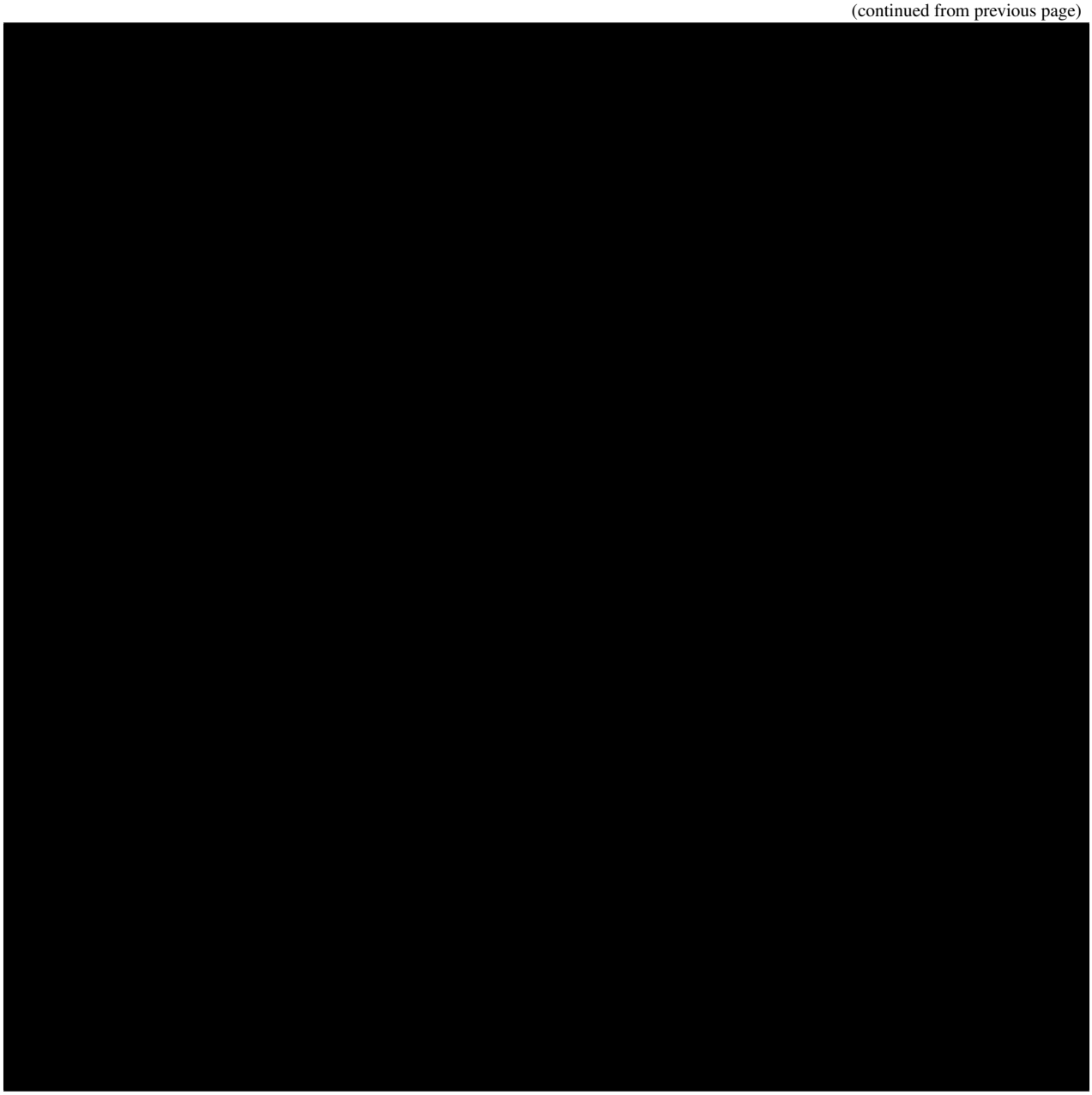
Subroutine	Remarks
cg_iric_write_sol_integer_f	Outputs integer-type calculation results, having a value for each grid node
cg_iric_write_sol_real_f	Outputs double-precision real-type calculation results, having a value for each grid node

List 6.14: Example source code (Value defined at grid nodes)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

(continues on next page)

(continued from previous page)



Value defined at grid cells

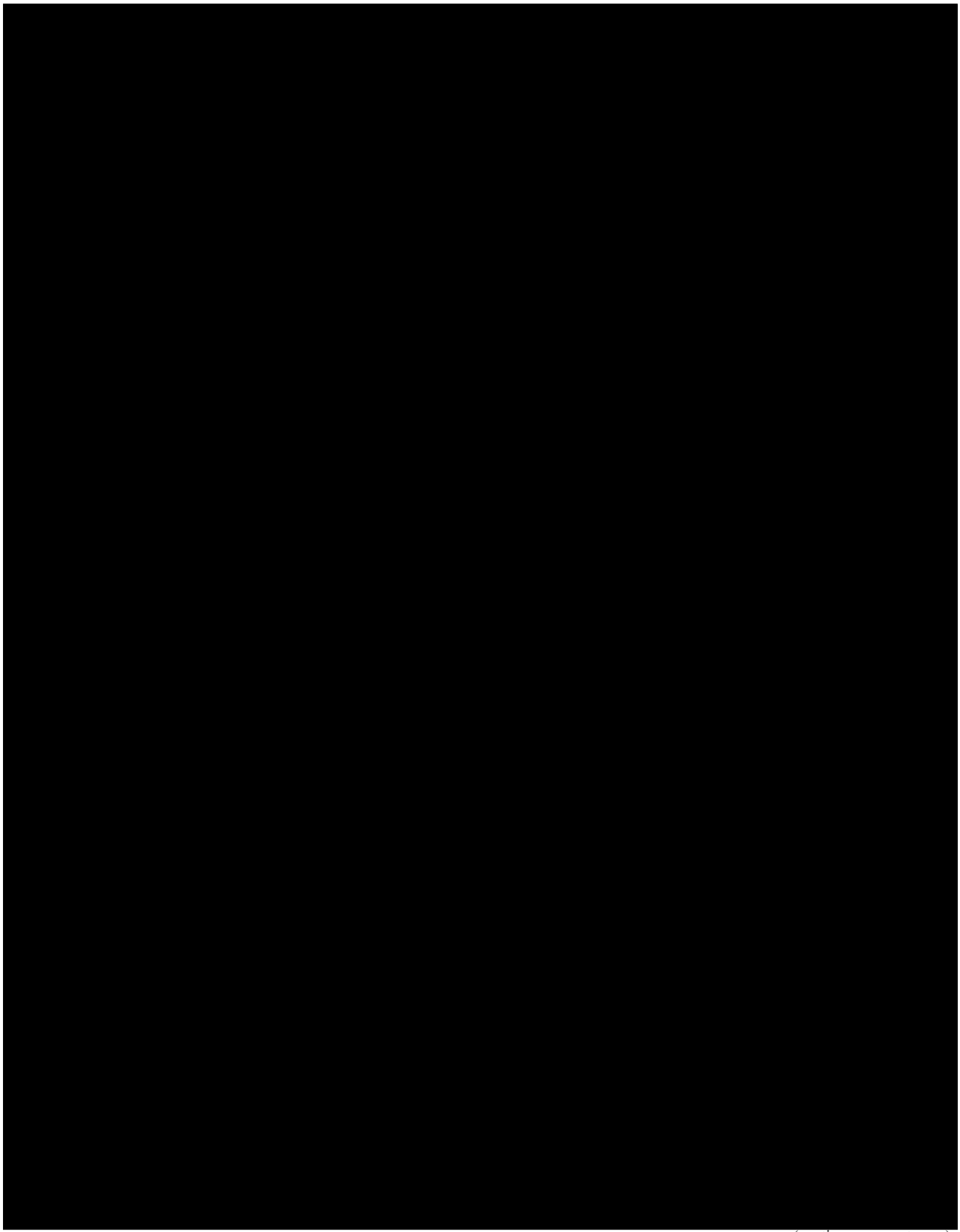
When you output value defined at grid cells, please use the functions in Table 6.17.

List 6.15 shows an example of the process to output value defined at grid cells.

Table 6.17: Subroutines to use for outputting result defined at grid cells

Subroutine	Remarks
cg_iric_write_sol_cell_integer_f	Outputs integer-type calculation results, having a value for each grid cell
cg_iric_write_sol_real_f	Outputs double-precision real-type calculation results, having a value for each grid cell

List 6.15: Example source code (Value defined at grid cells)



(continues on next page)

(continued from previous page)

```
56
57
58
59
60
61
62
```

Value defined at grid edges

When you output value defined at grid edges, please use the functions in Table 6.18.

List 6.16 shows an example of the process to output value defined at grid edges.

Table 6.18: Subroutines to use for outputting result defined at grid edges

Subroutine	Remarks
cg_iric_write_sol_iface_integer_f	Outputs integer-type calculation results, having a value for each grid edge at i-direction
cg_iric_write_sol_iface_real_f	Outputs double-precision real-type calculation results, having a value for each grid edge at i-direction
cg_iric_write_sol_jface_integer_f	Outputs integer-type calculation results, having a value for each grid edge at j-direction
cg_iric_write_sol_jface_real_f	Outputs double-precision real-type calculation results, having a value for each grid edge at j-direction

List 6.16: Example source code (Value defined at grid cells)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

(continues on next page)

(continued from previous page)

```
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```

Value defined at particles

When you output value defined at particles, please use the functions in Table 6.19.

List 6.17 shows an example of the process to output value defined at grid nodes.

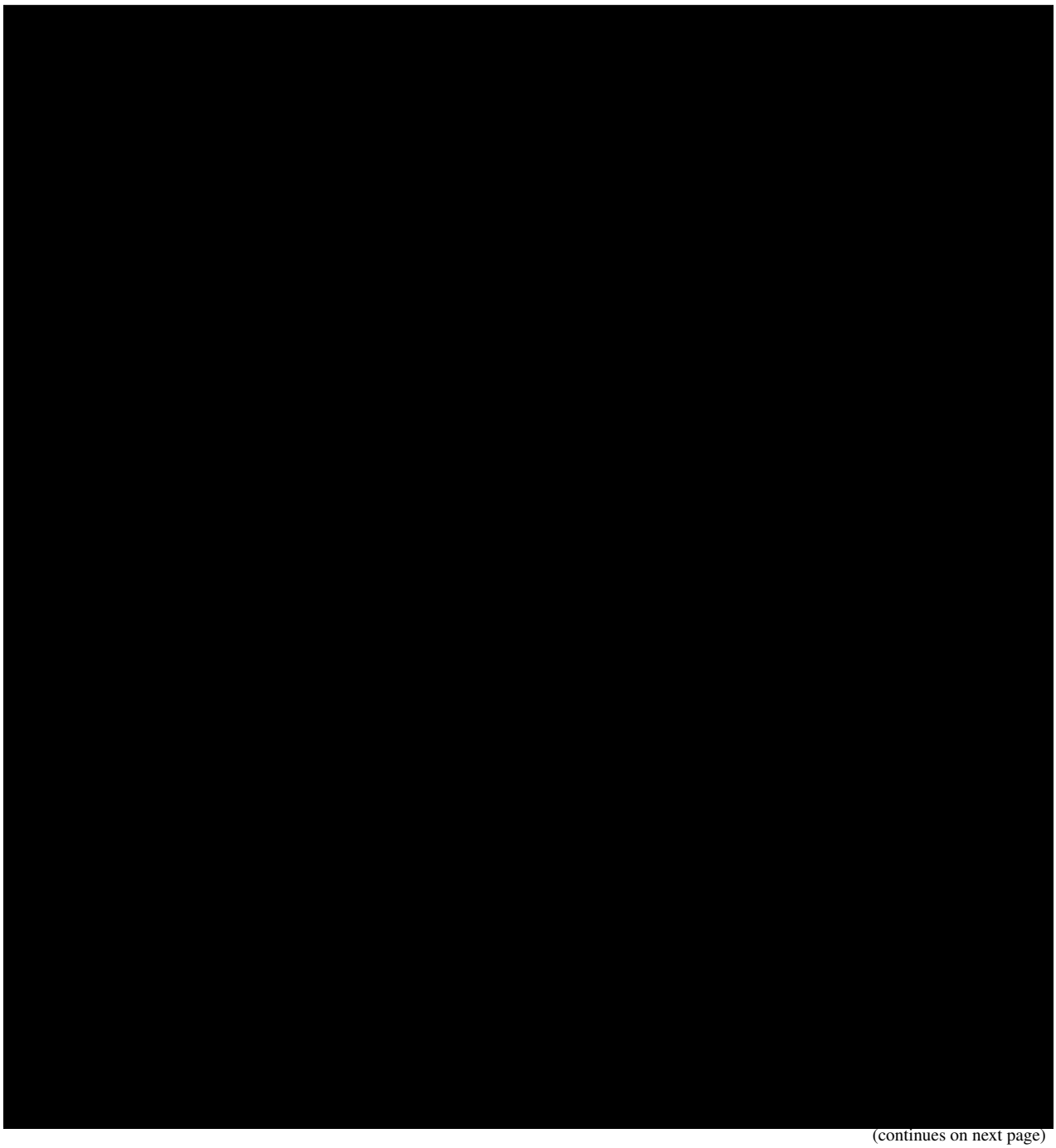
Note: These functions are deprecated

When you want to output value defined at particles, we recommend that you use functions described at *Value defined at particles (groups supported)* (page 123). Using those new functions, you can output particles with multiple groups, and you can visualize particles with different settings for color and size for each group.

Table 6.19: Subroutines to use for outputting result defined at particles

Subroutine	Remarks
cg_iric_write_sol_particle_pos2d_f	Outputs particle positions (two-dimensions)
cg_iric_write_sol_particle_pos3d_f	Outputs particle positions (three-dimensions)
cg_iric_write_sol_particle_integer_f	Outputs integer-type calculation results, giving a value for each particle.
cg_iric_write_sol_particle_real_f	Outputs double-precision real-type calculation results, giving a value for each particle.

List 6.17: Example source code (Value defined at particles)



(continues on next page)

(continued from previous page)

```
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
```

Value defined at particles (groups supported)

When you output value defined at particles, please use the functions in Table 6.20.

Using the functions explained here, you can output multiple groups of particles. To output data please call *cg_iric_write_sol_particlegroup_groupbegin_f* and *cg_iric_write_sol_particlegroup_groupend_f*, before and after outputting data.

List 6.18 shows an example of the process to output value defined at particles.

Note: In functions explained here, data for one particle is output with each function call.

Table 6.20: Subroutines to use for outputting result defined at particles (groups supported)

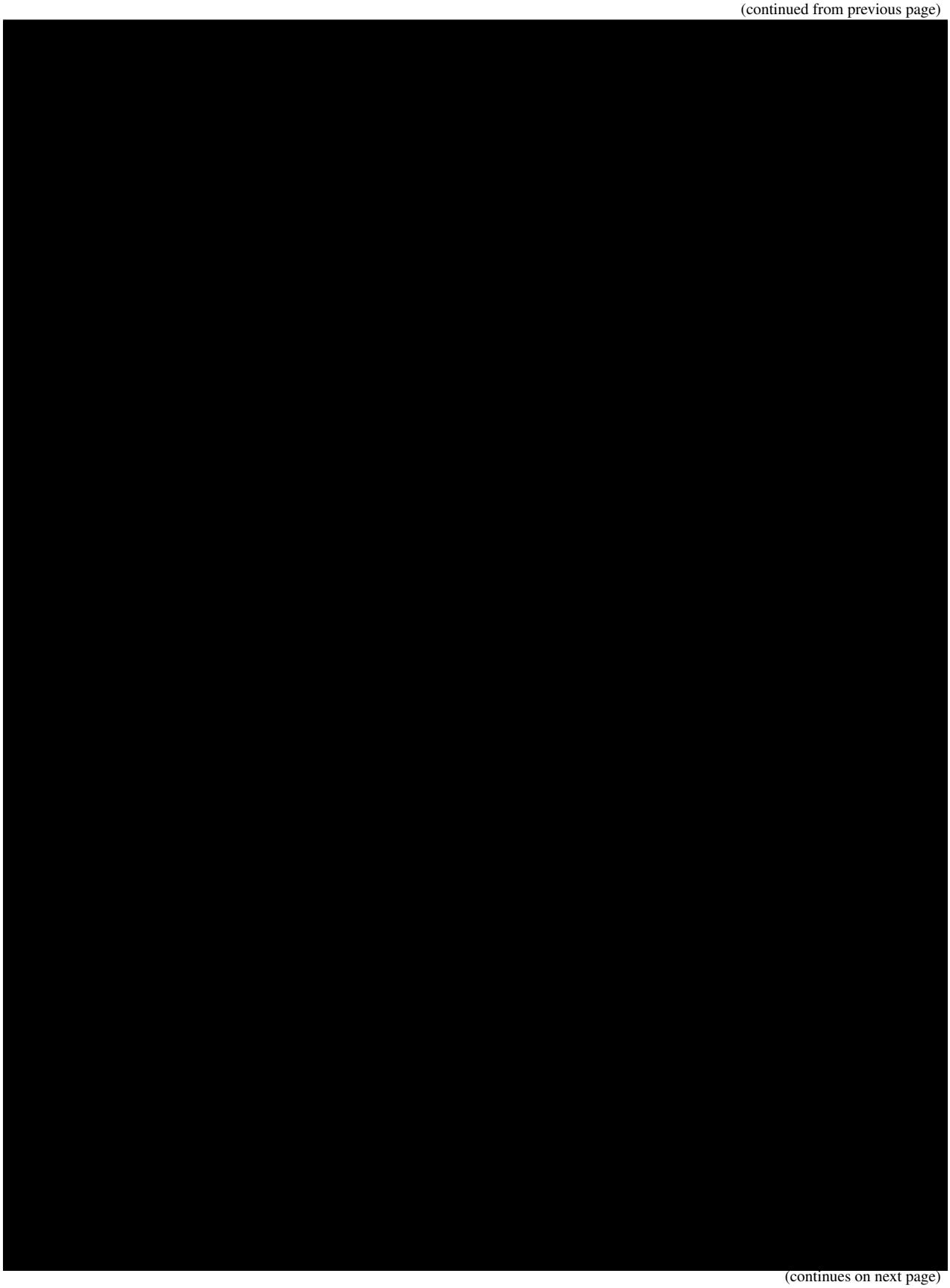
Subroutine	Remarks
cg_iric_write_sol_particlegroup_groupbegin_f	Start outputting calculation result defined at particles.
cg_iric_write_sol_particlegroup_groupend_f	Finish outputting calculation result defined at particles.
cg_iric_write_sol_particlegroup_pos2d_f	Outputs particle positions (two-dimensions)
cg_iric_write_sol_particlegroup_pos3d_f	Outputs particle positions (three-dimensions)
cg_iric_write_sol_particlegroup_integer_f	Outputs integer-type calculation results, giving a value for each particle.
cg_iric_write_sol_particlegroup_real_f	Outputs double-precision real-type calculation results, giving a value for each particle.

List 6.18: Example source code (Value defined at particles (groups supported))

```
1
2
```

(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)

Value defined at polygons or polylines

When you output value defined at polygons or polylines, please use the functions in Table 6.21.

When outputting polygons or polylines, you can output multiple groups. To output data please call *cg_iric_write_sol_polydata_groupbegin_f* and *cg_iric_write_sol_polydata_groupend_f*, before and after outputting data.

List 6.19 shows an example of the process to output value defined at polygons or polylines.

Note: In functions for outputting value defined at particles, all coordinates and values are output with one function calls. But in case of polygons or polylines, data for only one polygon or polyline is output with each function call.

Note: The functions to output value defined at polygons or polylines support only two-dimension data.

Note: You can mix calculation result defined at polygons and polylines in one group.

Note: For polygons and polylines the scalar quantity value only is supported.

Table 6.21: Subroutines to use for outputting result defined at polygons or polylines

Subroutine	Remarks
cg_iric_write_sol_polydata_groupbegin_f	Start outputting calculation result defined at polygons or polylines.
cg_iric_write_sol_polydata_groupend_f	Finish outputting calculation result defined at polygons or polylines.
cg_iric_write_sol_polydata_polygon_f	Output calculation result defined as polygon
cg_iric_write_sol_polydata_polyline_f	Output calculation result defined as polyline
cg_iric_write_sol_polydata_integer_f	Outputs integer-type calculation results, giving a value for a polygon or polyline
cg_iric_write_sol_polydata_real_f	Outputs double-precision real-type calculation results, giving a value for a polygon or polyline

List 6.19: Example source code (Value defined at polygons or polylines)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

(continues on next page)

(continued from previous page)

6.4.13 Reading calculation result

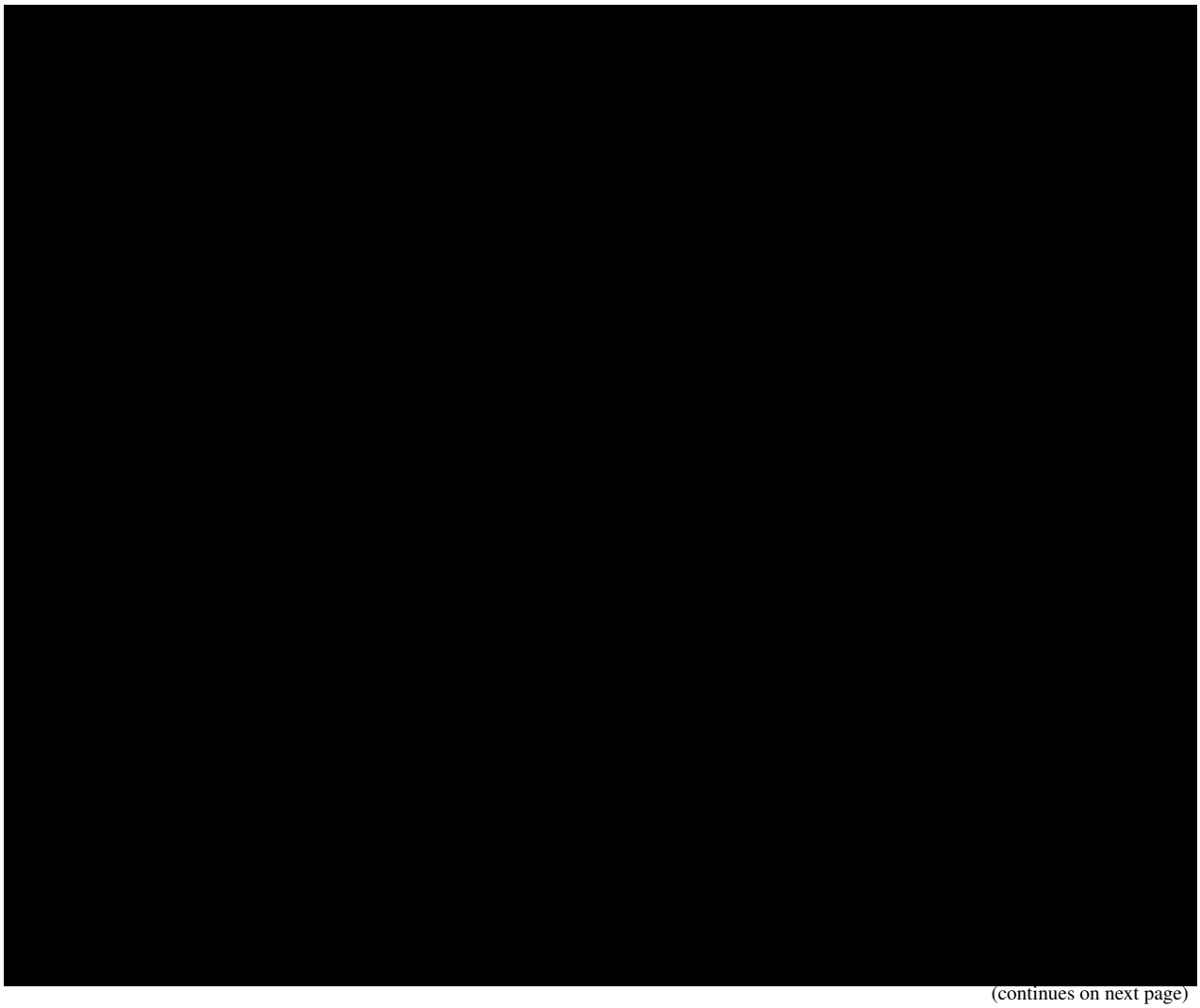
Read calculation result from CGNS files.

Table 6.22: Subroutines to use

Subroutine	Remarks
cg_iric_read_sol_count_f	Reads the number of calculation result
cg_iric_read_sol_time_f	Reads the time value
cg_iric_read_sol_iteration_f	Reads the loop iteration value
cg_iric_read_sol_baseiterative_integer_f	Reads the integer-type calculation result value
cg_iric_read_sol_baseiterative_real_f	Reads the double-precision real-type calculation result value
cg_iric_read_sol_gridcoord2d_f	Reads the 2D structured grid (for moving grid calculation)
cg_iric_read_sol_gridcoord3d_f	Reads the 3D structured grid (for moving grid calculation)
cg_iric_read_sol_integer_f	Reads the integer-type calculation result, having a value for each grid node
cg_iric_read_sol_real_f	Reads the double-precision real-type calculation result, having a value for each grid node
cg_iric_read_sol_cell_integer_f	Reads the integer-type calculation result, having a value for each grid cell
cg_iric_read_sol_cell_real_f	Reads the double-precision real-type calculation result, having a value for each grid cell

List 6.20 shows an example of reading caluculation result from CGNS file, and output to standard output.

List 6.20: Example of reading calculation result



(continues on next page)

(continued from previous page)

```
35
36
37
38
39
40
41
42
```

The functions are used in calculation analysis program (See *Steps of developing a calculation result analysis program* (page 31)).

6.4.14 Outputting Error code

Outputs error code to CGNS files. It is used only in grid generating programs.

Table 6.23: Subroutines to use

Subroutine	Remarks
cg_iric_write_errorcode_f	Outputs error code

6.4.15 Closing a CGNS file

Closes the CGNS file that has been opened by cg_open_f. The subroutine is defined in cgnslib.

Table 6.24: Subroutines to use

Subroutine	Remarks
cg_close_f	Closes the CGNS file

6.5 Reference

In this section, functions, formats, arguments of each function is explained.

In each page for functions, the data type of arguments are described for FORTRAN. For data type of arguments in C/C++ and Python, please refer to Table 6.25.

Please note that in Python, ier (that stores the error code) is not output, and exception is thrown instead, if error occurs. To impelement error handling, please use try-except expression.

Table 6.25: Relathionship between argument data type

FORTRAN	C/C++	Python
integer	int (int* for output)	int
double precision	double (double* for output)	float
real	float (float* for output)	–
character(*)	char*	str
integer, dimension(:), allocatable	int*	numpy.ndarray(dtype=int32)
double precision, dimension(:), allocatable	double*	numpy.ndarray(dtype=float64)
real, dimension(:), allocatable	float*	–

6.5.1 List of subroutines

The table below shows a list of subroutines and their classifications.

Table 6.26: List of iRIClib subrouti

Classification	Name	Description
Opening a CGNS file	cg_open_f	Opens a CGNS
Initializing iRIClib	cg_iric_init_f	Initializes the C
Initializing iRIClib	cg_iric_initread_f	Initializes the C
Setting up options	cg_initoption_f	Set up solver o
Reading the calculation conditions	cg_iric_read_integer_f	Gets the value
Reading the calculation conditions	cg_iric_read_real_f	Gets the value
Reading the calculation conditions	cg_iric_read_realsingle_f	Gets the value
Reading the calculation conditions	cg_iric_read_string_f	Gets the value
Reading the calculation conditions	cg_iric_read_functionalsize_f	Gets the size o
Reading the calculation conditions	cg_iric_read_functional_f	Gets the value
Reading the calculation conditions	cg_iric_read_functional_realsingle_f	Gets the value
Reading the calculation conditions	cg_iric_read_functionalwithname_f	Gets the value
Reading a calculation grid	cg_iric_gotogridcoord2d_f	Makes prepara
Reading a calculation grid	cg_iric_gotogridcoord3d_f	Makes prepara
Reading a calculation grid	cg_iric_getgridcoord2d_f	Reads the x an
Reading a calculation grid	cg_iric_getgridcoord3d_f	Reads the x, y
Reading a calculation grid	cg_iric_read_grid_integer_node_f	Reads the integ
Reading a calculation grid	cg_iric_read_grid_real_node_f	Reads double-p
Reading a calculation grid	cg_iric_read_grid_integer_cell_f	Reads the integ
Reading a calculation grid	cg_iric_read_grid_real_cell_f	Reads the doub
Reading a calculation grid	cg_iric_read_complex_count_f	Reads the num
Reading a calculation grid	cg_iric_read_complex_integer_f	Reads the integ
Reading a calculation grid	cg_iric_read_complex_real_f	Reads the doub
Reading a calculation grid	cg_iric_read_complex_realsingle_f	Reads the sing
Reading a calculation grid	cg_iric_read_complex_string_f	Reads the strin
Reading a calculation grid	cg_iric_read_complex_functionalsize_f	Checks the size
Reading a calculation grid	cg_iric_read_complex_functional_f	Reads function
Reading a calculation grid	cg_iric_read_complex_functionalwithname_f	Reads function
Reading a calculation grid	cg_iric_read_complex_functional_realsingle_f	Reads function
Reading a calculation grid	cg_iric_read_grid_complex_node_f	Reads the com
Reading a calculation grid	cg_iric_read_grid_complex_cell_f	Reads the com
Reading a calculation grid	cg_iric_read_grid_functionaltimesize_f	Reads the num
Reading a calculation grid	cg_iric_read_grid_functionaltime_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functionaldimensionsize_f	Reads the num
Reading a calculation grid	cg_iric_read_grid_functionaldimension_integer_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functionaldimension_real_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functional_integer_node_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functional_real_node_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functional_integer_cell_f	Reads the valu
Reading a calculation grid	cg_iric_read_grid_functional_real_cell_f	Reads the valu
Reading boundary conditions	cg_iric_read_bc_count_f	Reads the num
Reading boundary conditions	cg_iric_read_bc_indicessize_f	Reads the num
Reading boundary conditions	cg_iric_read_bc_indices_f	Reads the list o
Reading boundary conditions	cg_iric_read_bc_integer_f	Gets the value
Reading boundary conditions	cg_iric_read_bc_real_f	Gets the value

Table 6.26 – continued from previou

Classification	Name	Description
Reading boundary conditions	cg_iric_read_bc_realsingle_f	Gets the value
Reading boundary conditions	cg_iric_read_bc_string_f	Gets the value
Reading boundary conditions	cg_iric_read_bc_functionalsize_f	Gets the size o
Reading boundary conditions	cg_iric_read_bc_functional_f	Gets the value
Reading boundary conditions	cg_iric_read_bc_functional_realsingle_f	Gets the value
Reading boundary conditions	cg_iric_read_bc_functionalwithname_f	Gets the value
Reading geographic data	cg_iric_read_geo_count_f	Reads the num
Reading geographic data	cg_iric_read_geo_filename_f	Reads the file n
Reading geographic data	iric_geo_polygon_open_f	Opens the geog
Reading geographic data	iric_geo_polygon_read_integervalue_f	Reads the valu
Reading geographic data	iric_geo_polygon_read_realvalue_f	Reads the valu
Reading geographic data	iric_geo_polygon_read_pointcount_f	Reads the num
Reading geographic data	iric_geo_polygon_read_points_f	Reads the coor
Reading geographic data	iric_geo_polygon_read_holecount_f	Reads the num
Reading geographic data	iric_geo_polygon_read_holepointcount_f	Reads the num
Reading geographic data	iric_geo_polygon_read_holepoints_f	Reads the coor
Reading geographic data	iric_geo_polygon_close_f	Closes the geo
Reading geographic data	iric_geo_riversurvey_open_f	Opens the geog
Reading geographic data	iric_geo_riversurvey_read_count_f	Reads the num
Reading geographic data	iric_geo_riversurvey_read_position_f	Reads the coor
Reading geographic data	iric_geo_riversurvey_read_direction_f	Reads the direc
Reading geographic data	iric_geo_riversurvey_read_name_f	Reads the nam
Reading geographic data	iric_geo_riversurvey_read_realname_f	Reads the nam
Reading geographic data	iric_geo_riversurvey_read_leftshift_f	Reads the shift
Reading geographic data	iric_geo_riversurvey_read_altitudecount_f	Reads the num
Reading geographic data	iric_geo_riversurvey_read_altitudes_f	Reads the altitu
Reading geographic data	iric_geo_riversurvey_read_fixedpointl_f	Reads the data
Reading geographic data	iric_geo_riversurvey_read_fixedpointr_f	Reads the data
Reading geographic data	iric_geo_riversurvey_read_watersurfaceelevation_f	Reads the wate
Reading geographic data	iric_geo_riversurvey_close_f	Closes the geo
Outputting a calculation grid	cg_iric_writegridcoord1d_f	Outputs a one-
Outputting a calculation grid	cg_iric_writegridcoord2d_f	Outputs a two-
Outputting a calculation grid	cg_iric_writegridcoord3d_f	Outputs a three
Outputting a calculation grid	cg_iric_write_grid_integer_node_f	Outputs a grid
Outputting a calculation grid	cg_iric_write_grid_real_node_f	Outputs a grid
Outputting a calculation grid	cg_iric_write_grid_integer_cell_f	Outputs a grid
Outputting a calculation grid	cg_iric_write_grid_real_cell_f	Outputs a grid
Outputting time (or iteration count) information	cg_iric_write_sol_time_f	Outputs time
Outputting time (or iteration count) information	cg_iric_write_sol_iteration_f	Outputs the ite
Outputting calculation results	cg_iric_write_sol_gridcoord2d_f	Outputs a two-
Outputting calculation results	cg_iric_write_sol_gridcoord3d_f	Outputs a three
Outputting calculation results	cg_iric_write_sol_baseiterative_integer_f	Outputs integer
Outputting calculation results	cg_iric_write_sol_baseiterative_real_f	Outputs double
Outputting calculation results	cg_iric_write_sol_baseiterative_string_f	Outputs string-
Outputting calculation results	cg_iric_write_sol_integer_f	Outputs integer
Outputting calculation results	cg_iric_write_sol_real_f	Outputs double
Outputting calculation results	cg_iric_write_sol_cell_integer_f	Outputs integer
Outputting calculation results	cg_iric_write_sol_cell_real_f	Outputs double
Outputting calculation results	cg_iric_write_sol_iface_integer_f	Outputs integer

Table 6.26 – continued from previou

Classification	Name	Description
Outputting calculation results	cg_iric_write_sol_iface_real_f	Outputs double
Outputting calculation results	cg_iric_write_sol_jface_integer_f	Outputs integer
Outputting calculation results	cg_iric_write_sol_jface_real_f	Outputs double
Outputting calculation results (particles)	cg_iric_write_sol_particle_pos2d_f	Outputs particl
Outputting calculation results (particles)	cg_iric_write_sol_particle_pos3d_f	Outputs particl
Outputting calculation results (particles)	cg_iric_write_sol_particle_integer_f	Outputs integer
Outputting calculation results (particles)	cg_iric_write_sol_particle_real_f	Outputs double
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_groupbegin_f	Start outputting
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_groupend_f	Finish outputti
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_pos2d_f	Outputs particl
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_pos3d_f	Outputs particl
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_integer_f	Outputs integer
Outputting calculation results (particles)	cg_iric_write_sol_particlegroup_real_f	Outputs double
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_groupbegin_f	Start outputting
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_groupend_f	Finish outputti
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_polygon_f	Output calcula
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_polyline_f	Output calcula
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_integer_f	Outputs integer
Outputting calculation results (polygons, polylines)	cg_iric_write_sol_polydata_real_f	Outputs double
Functions to call befor and after outputting calculation results	iric_check_cancel_f	Checks whethe
Functions to call befor and after outputting calculation results	iric_check_lock_f	Checks whethe
Functions to call befor and after outputting calculation results	iric_write_sol_start_f	Inform the GU
Functions to call befor and after outputting calculation results	iric_write_sol_end_f	Inform the GU
Functions to call befor and after outputting calculation results	cg_iric_flush_f	Flush calculati
Reading calculation results	cg_iric_read_sol_count_f	Reads the num
Reading calculation results	cg_iric_read_sol_time_f	Reads the time
Reading calculation results	cg_iric_read_sol_iteration_f	Reads the loop
Reading calculation results	cg_iric_read_sol_baseiterative_integer_f	Reads the integ
Reading calculation results	cg_iric_read_sol_baseiterative_real_f	Reads the doub
Reading calculation results	cg_iric_read_sol_baseiterative_string_f	Reads the strin
Reading calculation results	cg_iric_read_sol_gridcoord2d_f	Reads the 2D s
Reading calculation results	cg_iric_read_sol_gridcoord3d_f	Reads the 3D s
Reading calculation results	cg_iric_read_sol_integer_f	Reads the integ
Reading calculation results	cg_iric_read_sol_real_f	Reads the doub
Reading calculation results	cg_iric_read_sol_cell_integer_f	Reads the integ
Reading calculation results	cg_iric_read_sol_cell_real_f	Reads the doub
Reading calculation results	cg_iric_read_sol_iface_integer_f	Reads the integ
Reading calculation results	cg_iric_read_sol_iface_real_f	Reads the doub
Reading calculation results	cg_iric_read_sol_jface_integer_f	Reads the integ
Reading calculation results	cg_iric_read_sol_jface_real_f	Reads the doub
Outputting error codes	cg_iric_write_errorcode_f	Outputs error c
Closing the CGNS file	cg_close_f	Closes a CGNS

The functions with "O" value for column "Multi" has functions that are used for the same purpose and used when handling multiple CGNS files. The "Multi" version of the functions end with "_mul_f" instead of "_f", and the first argument is file ID.

For example, the functions used for reading integer-type calculation result are as follows:

- Function used when handling single CGNS file

[redacted]

- Function used when handling multiple CGNS file.

[redacted]

The difference between single version and multiple version is shown in Table 6.27.

Table 6.27: Differences between functions for handling single or mulitple CGNS file

Item	For Single CGNS file	For Multiple CGNS file
Name	Ends with "_f"	Ends with "_mul_f"
Arguments	See the following sections	The first argument is File ID (integer)
Target CGNS file	File that is identified by the File ID that was specified as the argument of "cg_iric_init_f" or "cg_iric_initread_f"	File that is identified by the File ID that is specified as the first argument.

6.5.2 cg_open_f

- Opens a CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.28: Arguments of cg_open_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
mode	parameter (integer)	I	File Open mode
fid	integer	O	File ID
ier	integer	O	Error code. 0 means success.

6.5.3 cg_iric_init_f

- Initializes the internal variables that are used for reading and

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.29: Arguments of cg_iric_init_f

Variable name	Type	I/O	Description
fid	integer	I	File ID
ier	integer	O	Error code. 0 means success. In case of grid generating program, 1 means success.

6.5.4 cg_iric_initread_f

- Initializes the internal variables that are used for reading CGNS

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.30: Arguments of cg_iric_initread_f

Variable name	Type	I/O	Description
fid	integer	I	File ID
ier	integer	O	Error code. 0 means success.

6.5.5 iric_initoption_f

- Set up the options for the solver.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.31: Arguments of cg_iric_initoption_f

Variable name	Type	I/O	Description
optionval	integer	I	The value that corresponds to an option
ier	integer	O	Error code. 0 means success.

Appendix

Table 6.32: Values for optionval

optionval value	Content
IRIC_OPTION_CANCEL	Inform that the solver detects cancel operation using iric_check_cancel_f
IRIC_OPTION_DIVIDESOLUTIONS	Saves calculation results in divided CGNS files

6.5.6 cg_iric_read_integer_f

- Reads the value of a integer-type variable from the CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.33: Arguments of cg_iric_read_integer_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
intvalue	integer	O	Integer read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.7 cg_iric_read_real_f

- Reads the value of a double-precision real-type variable from the

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.34: Arguments of cg_iric_read_real_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
realvalue	double precision	O	Real number read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.8 cg_iric_read_realsingle_f

- Reads the value of a single-precision real-type variable from the

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.35: Arguments of cg_iric_read_realsingle_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
realvalue	real	O	Real number read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.9 cg_iric_read_string_f

- Reads the value of a string-type variable from the CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.36: Arguments of cg_iric_read_string_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
strvalue	character(*)	O	Character string read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.10 cg_iric_read_functionalsize_f

- Reads the size of a functional-type variable from the CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.37: Arguments of cg_iric_read_functionalsize_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
size	integer	O	Length of the array that has been read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.11 cg_iric_read_functional_f

- Reads the value of a functional-type double-precision real variable

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.38: Arguments of cg_iric_read_functional_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
x	double precision , dimension(:), allocatable	O	Array of x values
y	double precision, dimension(:), allocatable	O	Array of y values
ier	integer	O	Error code. 0 means success.

6.5.12 cg_iric_read_functional_realsingle_f

- Reads the value of a functional-type single-precision real variable

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.39: Arguments of cg_iric_read_functional_realsingle_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
x	real , dimension(:), allocatable	O	Array of x values
y	real, dimension(:), allocatable	O	Array of y values
ier	integer	O	Error code. 0 means success.

6.5.13 cg_iric_read_functionalwithname_f

- Reads the value of a functional-type real variable from the CGNS

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.40: Arguments of cg_iric_read_functionalwithname_f

Variable name	Type	I/O	Description
label	character(*)	I	Name of the variable defined in the solver definition file
name	character(*)	I	Name of the variable value name defined in the solver definition file
data	real , dimension(:), allocatable	O	Array of values
ier	integer	O	Error code. 0 means success.

6.5.14 cg_iric_gotogridcoord2d_f

- Makes preparations for reading a two-dimensional structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.41: Arguments of cg_iric_gotogridcoord2d_f

Variable name	Type	I/O	Description
nx	integer	O	Number of grid nodes in the i direction
ny	integer	O	Number of grid nodes in the j direction
ier	integer	O	Error code. 0 means success.

6.5.15 cg_iric_gotogridcoord3d_f

- Makes preparations for reading a 3D structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.42: Arguments of cg_iric_gotogridcoord3d_f

Variable name	Type	I/O	Description
nx	integer	O	Number of grid nodes in the i direction
ny	integer	O	Number of grid nodes in the j direction
nz	integer	O	Number of grid nodes in the k direction
ier	integer	O	Error code. 0 means success.

6.5.16 cg_iric_getgridcoord2d_f

- Reads a two-dimensional structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.43: Arguments of cg_iric_getgridcoord2d_f

Variable name	Type	I/O	Description
x	double precision, dimension(:), allocatable	O	x coordinate value of a grid node
y	double precision, dimension(:), allocatable	O	y coordinate value of a grid node
ier	integer	O	Error code. 0 means success.

6.5.17 cg_iric_getgridcoord3d_f

- Subroutine to reads a three-dimensional structured grid

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.44: Arguments of cg_iric_getgridcoord3d_f

Variable name	Type	I/O	Description
x	double precision, dimension(:), allocatable	O	x coordinate value of a grid node
y	double precision, dimension(:), allocatable	O	y coordinate value of a grid node
z	double precision, dimension(:), allocatable	O	z coordinate value of a grid node
ier	integer	O	Error code. 0 means success.

6.5.18 cg_iric_read_grid_integer_node_f

- Reads the integer attribute values defined for nodes of a structured

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.45: Arguments of cg_iric_read_grid_integer_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.19 cg_iric_read_grid_real_node_f

- Reads the double-precision real-type attribute values defined for

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.46: Arguments of cg_iric_read_grid_real_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	double precision, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.20 cg_iric_read_grid_integer_cell_f

- Reads the integer attribute values defined for cells of a structured

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.47: Arguments of cg_iric_read_grid_integer_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.21 cg_iric_read_grid_real_cell_f

- Reads the double-precision real-type attribute values defined for

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.48: Arguments of cg_iric_read_grid_real_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	double precision, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.22 cg_iric_read_complex_count_f

- Reads the number of groups of complex type grid attribute

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.49: Arguments of cg_iric_read_complex_count_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	O	The number of complex type grid attribute group
ier	integer	O	Error code. 0 means success.

6.5.23 cg_iric_read_complex_integer_f

- Reads the integer attribute values of complex type grid attribute

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.50: Arguments of cg_iric_read_complex_integer_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
value	integer	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.24 cg_iric_read_complex_real_f

- Reads the double precision attribute values of complex type grid

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.51: Arguments of cg_iric_read_complex_real_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
value	double precision	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.25 cg_iric_read_complex_realsingle_f

- Reads the single precision attribute values of complex type grid

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.52: Arguments of cg_iric_read_complex_realsingle_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
value	Real	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.26 cg_iric_read_complex_string_f

- Reads the string attribute values of complex type grid attribute

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.53: Arguments of cg_iric_read_complex_string_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
value	character(*)	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.27 cg_iric_read_complex_functionalsize_f

- Checks the size of a functional-type attribute of complex type grid

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.54: Arguments of cg_iric_read_complex_functionalsize_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
size	integer	O	The length of condition value array
ier	integer	O	Error code. 0 means success.

6.5.28 cg_iric_read_complex_functional_f

- Reads functional attribute data of complex type grid attribute

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.55: Arguments of cg_iric_read_complex_functional_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
x	double precision, dimension(:), allocatable	O	x value array
ier	integer	O	Error code. 0 means success.

6.5.29 cg_iric_read_complex_functionalwithname_f

- Reads functional attribute of complex type grid attribute (with

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.56: Arguments of cg_iric_read_complex_functionalwithname_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
paramname	character(*)	I	Value name
data	double precision, dimension(:), allocatable	O	Value array
ier	integer	O	Error code. 0 means success.

6.5.30 cg_iric_read_complex_functional_realsingle_f

- Reads functional attribute data of complex type grid attribute

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.57: Arguments of cg_iric_read_complex_functional_realsingle_f

Variable name	Type	I/O	Description
type	character(*)	I	Attribute name
num	integer	I	Group number
name	character(*)	I	Condition name
x	real, dimension(:), allocatable	O	x value array
y	real, dimension(:), allocatable	O	y value array
ier	integer	O	Error code. 0 means success.

6.5.31 cg_iric_read_grid_complex_node_f

- Reads the complex attribute values defined for grid nodes

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.58: Arguments of cg_iric_read_grid_complex_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.32 cg_iric_read_grid_complex_cell_f

- Reads the complex attribute values defined for grid cells

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.59: Arguments of cg_iric_read_grid_complex_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.33 cg_iric_read_grid_functionaltimesize_f

- Reads the number of values of dimension "Time" for functional grid

Format (FORTRAN)

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.60: Arguments of cg_iric_read_grid_functionaltimesize_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
count	integer	O	The number of values of dimension "Time"
ier	integer	O	Error code. 0 means success.

6.5.34 cg_iric_read_grid_functionaltime_f

- Reads the values of dimension "Time" for functional grid attribute

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.61: Arguments of cg_iric_read_grid_functionaltime_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	double precision, dimension(:), allocatable	O	The values of dimension "Time"
ier	integer	O	Error code. 0 means success.

6.5.35 cg_iric_read_grid_functionaldimensionsize_f

- Reads the number of values of dimension for functional grid attribute

Format (FORTRAN)

Format (C/C++)

Format (Python)

This function does not exists for Python.

Arguments

Table 6.62: Arguments of cg_iric_read_grid_functionaldimensionsize_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimname	character(*)	I	Dimension name
count	integer	O	The number of values of dimension "Time"
ier	integer	O	Error code. 0 means success.

6.5.36 cg_iric_read_grid_functionaldimension_integer_f

- Reads the values of integer dimension for functional grid attribute

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.63: Arguments of cg_iric_read_grid_functionaldimension_integer_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimname	character(*)	I	Dimension name
values	integer, dimension(:), allocatable	O	The values of dimension
ier	integer	O	Error code. 0 means success.

6.5.37 cg_iric_read_grid_functionaldimension_real_f

- Reads the values of double-precision dimension for functional grid

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.64: Arguments of cg_iric_read_grid_functionaldimension_real_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimname	character(*)	I	Dimension name
values	double precision, dimension(:), allocatable	O	The values of dimension
ier	integer	O	Error code. 0 means success.

6.5.38 cg_iric_read_grid_functional_integer_node_f

- Reads the values of functional integer grid attribute with dimension

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.65: Arguments of cg_iric_read_grid_functional_integer_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimid	integer	I	ID of "Time" (1 to the number of Time)
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.39 cg_iric_read_grid_functional_real_node_f

- Reads the values of functional double-precision grid attribute with

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.66: Arguments of cg_iric_read_grid_functional_real_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimid	integer	I	ID of "Time" (1 to the number of Time)
values	double precision, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.40 cg_iric_read_grid_functional_integer_cell_f

- Reads the values of functional integer grid attribute with dimension

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.67: Arguments of cg_iric_read_grid_functional_integer_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimid	integer	I	ID of "Time" (1 to the number of Time)
values	integer, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.41 cg_iric_read_grid_functional_real_cell_f

- Reads the values of functional double-precision grid attribute with

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.68: Arguments of cg_iric_read_grid_functional_real_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
dimid	integer	I	ID of "Time" (1 to the number of Time)
values	double precision, dimension(:), allocatable	O	Attribute value
ier	integer	O	Error code. 0 means success.

6.5.42 cg_iric_read_bc_count_f

- Reads the number of boundary condition.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.69: Arguments of cg_iric_bc_count_f

Variable name	Type	I/O	Description
type	character(*)	I	The type name of boundary condition you want to know the count.
num	integer	O	The number of boundary condition

6.5.43 cg_iric_read_bc_indicessize_f

- Reads the number of elements (nodes or cells) where the boundary

Format (FORTRAN)

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.70: Arguments of cg_iric_read_bc_indicessize_f

Variable name	Type	I/O	Description
type	character(*)	I	The type name of boundary condition you want to know the indices size
num	integer	O	The boundary condition ID number
size	integer	O	The number of elements (nodes or cells) where the boundary condition is set.
ier	integer	O	Error code. 0 means success.

6.5.44 cg_iric_read_bc_indices_f

- Reads the elements (nodes or cells) where the boundary condition is

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.71: Arguments of cg_iric_read_bc_indices_f

Variable name	Type	I/O	Description
type	character(*)	I	The type name of boundary condition you want to know the indices size
num	integer	O	The boundary condition ID number
ier	integer	O	Error code. 0 means success.

Note

Values returned to indices depends on the position where boundary condition is defined, like shown in Table 6.72.

Please note that when the boundary condition is defined at nodes or cells, it outputs two values for each element, and when defined at edges, it outputs four values for each element.

Table 6.72: Relationship between the position where boundary condition is defined and the values of indices

Position where boundary condition is defined	Values of indices
Nodes	I of grid node 1, J of grid node 1 . . . , I of grid node N, J of grid node N
Cells	I of grid cell 1, J of grid cell 1 . . . , I of grid cell N, J of grid cell N
Edges	I of start position of edge 1, J of start position of edge 1, I of end position of edge 1, J of end position of edge 1 . . . , I of start position of edge N, J of start position of edge N, I of end position of edge N, J of end position of edge N

6.5.45 cg_iric_read_bc_integer_f

- Reads the value of a string-type variable from the CGNS file.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.73: Arguments of cg_iric_read_bc_integer_f

Variable name	Type	I/O	Description
type	character(*)	I	Name of boundary condition
num	integer	I	Boundary condition number
label	character(*)	I	Name of the boundary condition variable defined in the solver definition file
intvalue	integer	O	Integer read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.46 cg_iric_read_bc_real_f

- Reads the value of a double-precision real-type variable from the

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.74: Arguments of cg_iric_read_bc_real_f

Variable name	Type	I/O	Description
type	character(*)	I	Name of boundary condition
num	integer	I	Boundary condition number
label	character(*)	I	Name of the variable defined in the solver definition file
realvalue	double precision	O	Real number read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.47 cg_iric_read_bc_realsingle_f

- Reads the value of a single-precision real-type variable from the

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments of cg_iric_read_bc_realsingle_f

file cg_iric_read_bc_realsingle_f_args.csv

header-rows 1

6.5.48 cg_iric_read_bc_string_f

- Reads the value of a string-type variable from the CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.75: Arguments of cg_iric_read_bc_string_f

Variable name	Type	I/O	Description
type	character(*)	I	Name of boundary condition
num	integer	I	Boundary condition number
label	character(*)	I	Name of the variable defined in the solver definition file
strvalue	character(*)	O	Character string read from the CGSN file
ier	integer	O	Error code. 0 means success.

6.5.49 cg_iric_read_bc_functionalsize_f

- Reads the size of a functional-type variable from the CGNS file.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments of cg_iric_read_bc_functionalsize_f

file cg_iric_read_bc_functionalsize_f_args.csv

header-rows 1

6.5.50 cg_iric_read_bc_functional_f

- Reads the value of a functional-type double-precision real variable

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.76: Arguments of cg_iric_read_bc_functional_f

Variable name	Type	I/O	Description
type	character(*)	I	Name of boundary condition
num	integer	I	Boundary condition number
label	character(*)	I	Name of the variable defined in the solver definition file
x	double precision, dimension(:), allocatable	O	Array of x values
y	double precision, dimension(:), allocatable	O	Array of y values
ier	integer	O	Error code. 0 means success.

6.5.51 cg_iric_read_bc_functional_realsingle_f

- Reads the value of a functional-type single-precision real variable

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments of cg_iric_read_bc_functional_realsingle_f

file cg_iric_read_bc_functional_realsingle_f_args.csv

header-rows 1

6.5.52 cg_iric_read_bc_functionalwithname_f

- Reads the value of a functional-type real variable from the CGNS

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.77: Arguments of cg_iric_read_bc_functionalwithname_f

Variable name	Type	I/O	Description
type	character(*)	I	Name of boundary condition
num	integer	I	Boundary condition number
label	character(*)	I	Name of the variable defined in the solver definition file
name	character(*)	I	Name of the variable value name defined in the solver definition file
data	real , dimension(:), allocatable	O	Array of values
ier	integer	O	Error code. 0 means success.

6.5.53 cg_iric_read_geo_count_f

- Reads the number of geographic data

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.78: Arguments of cg_iric_read_geo_count_f

Variable name	Type	I/O	Description
name	character(*)	I	Geographic data group name
geocount	integer	O	The number of geographic data
ier	integer	O	Error code. 0 means success.

6.5.54 cg_iric_read_geo_filename_f

- Reads the file name and data type of geographic data

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.79: Arguments of cg_iric_read_geo_filename_f

Variable name	Type	I/O	Description
name	character(*)	I	Geographic data group name
geoid	integer	I	Geographic data number
geofilename	character(*)	O	Filename
geotype	integer	O	Geographic data type
ier	integer	O	Error code. 0 means success.

6.5.55 iric_geo_polygon_open_f

- Opens the geographic data file that contains polygon data

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.80: Arguments of iric_geo_polygon_open_f

Variable name	Type	I/O	Description
filename	character(*)	I	File name
pid	integer	O	Polygon ID for opened file
ier	integer	O	Error code. 0 means success.

6.5.56 iric_geo_polygon_read_integervalue_f

- Reads the value of polygon data as integer

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.81: Arguments of iric_geo_polygon_read_integervalue_f

Variable name	Type	I/O	Description
Pid	integer	I	Polygon ID
intval	integer	O	The value of the polygon
Ier	integer	O	Error code. 0 means success.

6.5.57 iric_geo_polygon_read_realvalue_f

- Reads the value of polygon datas double precision real

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.82: Arguments of iric_geo_polygon_read_realvalue_f

Variable name	Type	I/O	Description
Pid	integer	I	Polygon ID
realval	double precision	O	The value of the polygon
Ier	integer	O	Error code. 0 means success.

6.5.58 iric_geo_polygon_read_pointcount_f

- Reads the number of polygon vertices

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.83: Arguments of iric_geo_polygon_read_pointcount_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
count	integer	O	The number of vertices of the polygon
ier	integer	O	Error code. 0 means success.

6.5.59 iric_geo_polygon_read_points_f

- Reads the coorinates of polygon vertices

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.84: Arguments of iric_geo_polygon_read_points_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
x	double precision , dimension(:), allocatable	O	X coordinates of polygon vertices
y	double precision , dimension(:), allocatable	O	Y coordinates of polygon vertices
ier	integer	O	Error code. 0 means success.

6.5.60 iric_geo_polygon_read_holecount_f

- Reads the number of holes in the polygon

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Arguments

Table 6.85: Arguments of iric_geo_polygon_read_holecount_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
holecount	integer	O	The number of holes
ier	integer	O	Error code. 0 means success.

6.5.61 iric_geo_polygon_read_holepointcount_f

- Reads the number of vertices of hole polygon

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.86: Arguments of iric_geo_polygon_read_holepointcount_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
holeid	integer	I	Hole ID
count	integer	O	The number of vertices of the hole polygon
ier	integer	O	Error code. 0 means success.

6.5.62 iric_geo_polygon_read_holepoints_f

- Reads the coordinates of hole polygon vertices

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.87: Arguments of iric_geo_polygon_read_holepoints_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
holeid	integer	I	Hole ID
x	double precision , dimension(:), allocatable	O	X coordinates of hole polygon vertices
y	double precision , dimension(:), allocatable	O	Y coordinates of hole polygon vertices
ier	integer	O	Error code. 0 means success.

6.5.63 iric_geo_polygon_close_f

- Closes the geographic data file

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.88: Arguments of iric_geo_polygon_close_f

Variable name	Type	I/O	Description
pid	integer	I	Polygon ID
ier	integer	O	Error code. 0 means success.

6.5.64 iric_geo_riversurvey_open_f

- Opens the geographic data file that contains river survey data

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.89: Arguments of iric_geo_riversurvey_open_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
rid	integer	O	River Survey Data ID
ier	integer	O	Error code. 0 means success.

6.5.65 iric_geo_riversurvey_read_count_f

- Reads the number of the crosssections in river survey data

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.90: Arguments of iric_geo_riversurvey_read_count_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
count	integer	O	The number of crosssections
ier	integer	O	Error code. 0 means success.

6.5.66 iric_geo_riversurvey_read_position_f

- Reads the coordinates of the crosssection center point

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.91: Arguments of iric_geo_riversurvey_read_position_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
x	double precision	O	X coordinate of the center point
y	double precision	O	Y coordinate of the center point
ier	integer	O	Error code. 0 means success.

6.5.67 iric_geo_riversurvey_read_direction_f

- Reads the direction of the crosssection as normalized vector

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.92: Arguments of iric_geo_riversurvey_read_direction_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
vx	double precision	O	X component of the normalized direction vector
vx	double precision	O	Y component of the normalized direction vector
ier	integer	O	Error code. 0 means success.

6.5.68 iric_geo_riversurvey_read_name_f

- Reads the name of the crosssection as string

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.93: Arguments of iric_geo_riversurvey_read_name_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
name	character(*)	O	Name of the crosssection
ier	integer	O	Error code. 0 means success.

6.5.69 iric_geo_riversurvey_read_realname_f

- Reads the name of the crosssection as real number

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.94: Arguments of iric_geo_riversurvey_read_realname_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
realname	double precision	O	Name of the crosssection
ier	integer	O	Error code. 0 means success.

6.5.70 iric_geo_riversurvey_read_leftshift_f

- Reads the shift offset value of the crosssection

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.95: Arguments of iric_geo_riversurvey_read_leftshift_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
shift	double precision	O	The amount of left shift
ier	integer	O	Error code. 0 means success.

6.5.71 iric_geo_riversurvey_read_altitudecount_f

- Reads the number of altitude data of the crosssection

Format (FORTRAN)

```
```

Format (C/C++)

```
```

Format (Python)

This function does not exists for Python.

Arguments

Table 6.96: Arguments of iric_geo_riversurvey_read_altitudecount_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
count	integer	O	The number of altitude data
ier	integer	O	Error code. 0 means success.

6.5.72 iric_geo_riversurvey_read_altitudes_f

- Reads the altitude data of the crosssection

Format (FORTRAN)

```
```

Format (C/C++)

```
```

Format (Python)

This function does not exists for Python.

Arguments

Table 6.97: Arguments of iric_geo_riversurvey_read_altitudes_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	pointid	I	Crosssection ID
position	double precision , dimension(:), allocatable	O	Altitude position (less than 0: left bank side, grater than 0: right bank side)
height	double precision , dimension(:), allocatable	O	Altitude height (elevation)
active	integer, dimension(:), allocatable	O	Altitude data active/inactive (1: active, 0: inactive)
ier	integer	O	Error code. 0 means success.

6.5.73 iric_geo_riversurvey_read_fixedpointl_f

- Reads the data of left bank extension line of the crosssection

Format (FORTRAN)

Format (C/C++)

Format (Python)

This function does not exists for Python.

Arguments

Table 6.98: Arguments of iric_geo_riversurvey_read_fixedpointl_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
set	integer	O	If defined, the value is 1
directionx	double precision	O	X component of normalized direction vector
direction	double precision	O	Y component of normalized direction vector
index	integer	O	The ID of the altitude data where the left bank extension line starts
ier	integer	O	Error code. 0 means success.

6.5.74 iric_geo_riversurvey_read_fixedpointr_f

- Reads the data of right bank extension line of the crosssection

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.99: Arguments of iric_geo_riversurvey_read_fixedpointr_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
set	integer	O	If defined, the value is 1
directionx	double precision	O	X component of normalized direction vector
direction	double precision	O	Y component of normalized direction vector
index	integer	O	The ID of the altitude data where the right bank extension line starts
ier	integer	O	Error code. 0 means success.

6.5.75 iric_geo_riversurvey_read_watersurfaceelevation_f

- Reads the water elevation at the crosssection

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.100: Arguments of iric_geo_riversurvey_read_watersurfaceelevation_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
pointid	integer	I	Crosssection ID
set	integer	O	If defined the value is 1
value	double precision	O	Water surface elevation
ier	integer	O	Error code. 0 means success.

6.5.76 iric_geo_riversurvey_close_f

- Closes the geographic data file

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.101: Arguments of iric_geo_riversurvey_close_f

Variable name	Type	I/O	Description
rid	integer	I	River Survey Data ID
ier	integer	O	Error code. 0 means success.

6.5.77 cg_iric_writegridcoord1d_f

- Outputs a one-dimensional structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.102: Arguments of cg_iric_writegridcoord1d_f

Variable name	Type	I/O	Description
nx	integer	I	Number of grid nodes in the i direction
x	double precision, dimension(:), allocatable	I	x coordinate value of a grid node
ier	integer	O	Error code. 0 means success.

6.5.78 cg_iric_writegridcoord2d_f

- Outputs a two-dimensional structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.103: Arguments of cg_iric_writegridcoord2d_f

Variable name	Type	I/O	Description
nx	integer	I	Number of grid nodes in the i direction
ny	integer	I	Number of grid nodes in the j direction
x	double precision, dimension(:,:), allocatable	I	x coordinate value of a grid node
y	double precision, dimension(:,:), allocatable	I	y coordinate value of a grid node
ier	integer	O	Error code. 0 means success.

6.5.79 cg_iric_writegridcoord3d_f

- Outputs a three-dimensional structured grid.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.104: Arguments of cg_iric_writegridcoord3d_f

Variable name	Type	I/O	Description
nx	integer	I	Number of grid nodes in the i direction
ny	integer	I	Number of grid nodes in the j direction
nz	integer	I	Number of grid nodes in the k direction
x	double precision, dimension(:), allocatable	I	x coordinate value of a grid node
y	double precision, dimension(:), allocatable	I	y coordinate value of a grid node
z	double precision, dimension(:), allocatable	I	z coordinate value of a grid node
ier	integer	O	Error code. 0 means success.

6.5.80 cg_iric_write_grid_integer_node_f

- Outputs grid attribute values defined at grid nodes with integer

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.105: Arguments of cg_iric_write_grid_integer_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), llocatable	O	Attribute values
ier	integer	O	Error code. 0 means success.

6.5.81 cg_iric_write_grid_real_node_f

- Outputs grid attribute values defined at grid nodes with real number

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.106: Arguments of cg_iric_write_grid_real_node_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	double precision, dimension(:), allocatable	O	Attribute values
ier	integer	O	Error code. 0 means success.

6.5.82 cg_iric_write_grid_integer_cell_f

- Outputs grid attribute values defined at grid cells with integer value.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.107: Arguments of cg_iric_write_grid_integer_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	integer, dimension(:), allocatable	O	Attribute values
ier	integer	O	Error code. 0 means success.

6.5.83 cg_iric_write_grid_real_cell_f

- Outputs grid attribute values defined at grid cells with real number value.

Format (FORTRAN)

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.108: Arguments of cg_iric_write_grid_real_cell_f

Variable name	Type	I/O	Description
label	character(*)	I	Attribute name
values	double precision, dimension(:), allocatable	O	Attribute values
ier	integer	O	Error code. 0 means success.

6.5.84 cg_iric_write_sol_time_f

- Outputs time.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.109: Arguments of cg_iric_write_sol_time_f

Variable name	Type	I/O	Description
time	double precision	I	Time
ier	integer	O	Error code. 0 means success.

6.5.85 cg_iric_write_sol_iteration_f

- Outputs iteration count.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.110: Arguments of cg_iric_write_sol_iteration_f

Variable name	Type	I/O	Description
iteration	integer	I	Iteration count
ier	integer	O	Error code. 0 means success.

6.5.86 cg_iric_write_sol_gridcoord2d_f

- Outputs a two-dimensional structured grid.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.111: Arguments of cg_iric_write_sol_gridcoord2d_f

Variable name	Type	I/O	Description
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate
ier	integer	O	Error code. 0 means success.

6.5.87 cg_iric_write_sol_gridcoord3d_f

- Outputs a three-dimensional structured grid.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.112: Arguments of cg_iric_write_sol_gridcoord3d_f

Variable name	Type	I/O	Description
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate.
z	double precision, dimension(:), allocatable	I	z coordinate
ier	integer	O	Error code. 0 means success.

6.5.88 cg_iric_write_sol_baseiterative_integer_f

- Outputs integer-type calculation results.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.113: Arguments of cg_iric_write_sol_baseiterative_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output
val	integer	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.89 cg_iric_write_sol_baseiterative_real_f

- Outputs double-precision real-type calculation results.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.114: Arguments of cg_iric_write_sol_baseiterative_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output
val	double precision	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.90 cg_iric_write_sol_baseiterative_string_f

- Outputs string-type calculation results.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.115: Arguments of cg_iric_write_sol_baseiterative_string_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output
val	character*	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.91 cg_iric_write_sol_integer_f

- Outputs integer-type calculation results, giving a value for each grid node.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.116: Arguments of cg_iric_write_sol_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output
val	integer, dimension(:,:), allocatable	I	Value to be output In the case of a 3D grid, the type should be integer, dimension(:,:,:), allocatable.
ier	integer	O	Error code. 0 means success.

6.5.92 cg_iric_write_sol_real_f

- Outputs double-precision real-type calculation results, having a value for each grid node.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.117: Arguments of cg_iric_write_sol_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision, dimension(:,:), allocatable	I	Value to be output In the case of a 3D grid, the type should be double precision, dimension(:,:,:), allocatable.
ier	integer	O	Error code. 0 means success.

6.5.93 cg_iric_write_sol_cell_integer_f

- Outputs integer-type calculation results, giving a value for each grid cell.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.118: Arguments of cg_iric_write_sol_cell_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output
val	integer, dimension(:,:), allocatable	I	Value to be output In the case of a 3D grid, the type should be integer, dimension(:,:,:), allocatable.
ier	integer	O	Error code. 0 means success.

6.5.94 cg_iric_write_sol_cell_real_f

- Outputs double-precision real-type calculation results, having a value for each grid cell.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.119: Arguments of cg_iric_write_sol_cell_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision, dimension(:,:), allocatable	I	Value to be output In the case of a 3D grid, the type should be double precision, dimension(:,:,:), allocatable.
ier	integer	O	Error code. 0 means success.

6.5.95 cg_iric_write_sol_iface_integer_f

- Outputs integer-type calculation results, giving a value for each grid edge at i-direction.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.120: Arguments of cg_iric_write_sol_iface_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	integer, dimension(:,:), allocatable	I	Value
ier	integer	O	Error code. 0 means success.

6.5.96 cg_iric_write_sol_iface_real_f

- Outputs double-precision real-type calculation results, giving a value for each grid edge at i-direction.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.121: Arguments of cg_iric_write_sol_iface_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision, dimension(:,:), allocatable	I	Value
ier	integer	O	Error code. 0 means success.

6.5.97 cg_iric_write_sol_jface_integer_f

- Outputs integer-type calculation results, giving a value for each grid edge at j-direction.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.122: Arguments of cg_iric_write_sol_jface_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	integer, dimension(:,:), allocatable	I	Value
ier	integer	O	Error code. 0 means success.

6.5.98 cg_iric_write_sol_jface_real_f

- Outputs double-precision real-type calculation results, giving a value for each grid edge at j-direction.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.123: Arguments of cg_iric_write_sol_jface_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision, dimension(:,:), allocatable	I	Value
ier	integer	O	Error code. 0 means success.

6.5.99 cg_iric_write_sol_particle_pos2d_f

- Outputs particle positions (two-dimensions)

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.124: Arguments of cg_iric_write_sol_particle_pos2d_f

Variable name	Type	I/O	Description
count	integer	I	The number of particles
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate.
ier	integer	O	Error code. 0 means success.

6.5.100 cg_iric_write_sol_particle_pos3d_f

- Outputs particle positions (three-dimensions)

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.125: Arguments of cg_iric_write_sol_particle_pos3d_f

Variable name	Type	I/O	Description
count	integer	I	The number of particles
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate.
z	double precision, dimension(:), allocatable	I	z coordinate.
ier	integer	O	Error code. 0 means success.

6.5.101 cg_iric_write_sol_particle_integer_f

- Outputs integer-type calculation results, giving a value for each particle.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.126: Arguments of cg_iric_write_sol_particle_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	integer, dimension(:), allocatable	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.102 cg_iric_write_sol_particle_real_f

- Outputs double-precision real-type calculation results, giving a value for each particle.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.127: Arguments of cg_iric_write_sol_particle_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision, dimension(:), allocatable	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.103 cg_iric_write_sol_particlegroup_groupbegin_f

- Start outputting calculation result defined at particles

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.128: Arguments of cg_iric_write_sol_particlegroup_groupbegin_f

Variable name	Type	I/O	Description
name	character*	I	Name of the group to be output.
ier	integer	O	Error code. 0 means success.

6.5.104 cg_iric_write_sol_particlegroup_groupend_f

- Finish outputting calculation result defined as particles.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.129: Arguments of cg_iric_write_sol_particlegroup_groupend_f

Variable name	Type	I/O	Description
ier	integer	O	Error code. 0 means success.

6.5.105 cg_iric_write_sol_particlegroup_pos2d_f

- Outputs particle positions (two-dimensions)

Format (FORTRAN)

Format (C/C++)

Format (Python)

[redacted]

Arguments

Table 6.130: Arguments of cg_iric_write_sol_particlegroup_pos2d_f

Variable name	Type	I/O	Description
x	double precision	I	x coordinate.
y	double precision	I	y coordinate.
ier	integer	O	Error code. 0 means success.

6.5.106 cg_iric_write_sol_particlegroup_pos3d_f

- Outputs particle positions (three-dimensions)

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.131: Arguments of cg_iric_write_sol_particlegroup_pos3d_f

Variable name	Type	I/O	Description
x	double precision	I	x coordinate.
y	double precision	I	y coordinate.
z	double precision	I	z coordinate.
ier	integer	O	Error code. 0 means success.

6.5.107 cg_iric_write_sol_particlegroup_integer_f

- Outputs integer-type calculation results, giving a value for each particle.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.132: Arguments of cg_iric_write_sol_particlegroup_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	integer	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.108 cg_iric_write_sol_particlegroup_real_f

- Outputs double-precision real-type calculation results, giving a value for each particle.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.133: Arguments of cg_iric_write_sol_particlegroup_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.109 cg_iric_write_sol_polydata_groupbegin_f

- Start outputting calculation result defined at polygons or polylines.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.134: Arguments of cg_iric_write_sol_polydata_groupbegin_f

Variable name	Type	I/O	Description
name	character*	I	Name of the group to be output.
ier	integer	O	Error code. 0 means success.

6.5.110 cg_iric_write_sol_polydata_groupend_f

- Finish outputting calculation result defined at polygons or polylines.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.135: Arguments of cg_iric_write_sol_polydata_groupend_f

Variable name	Type	I/O	Description
ier	integer	O	Error code. 0 means success.

6.5.111 cg_iric_write_sol_polydata_polygon_f

- Output calculation result defined as polygon.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.136: Arguments of cg_iric_write_sol_polydata_polygon_f

Variable name	Type	I/O	Description
numpoints	integer	I	The number of points in the polygon
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate.
ier	integer	O	Error code. 0 means success.

6.5.112 cg_iric_write_sol_polydata_polyline_f

- Output calculation result defined as polyline.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.137: Arguments of cg_iric_write_sol_polydata_polyline_f

Variable name	Type	I/O	Description
numpoints	integer	I	The number of points in the polyline
x	double precision, dimension(:), allocatable	I	x coordinate.
y	double precision, dimension(:), allocatable	I	y coordinate.
ier	integer	O	Error code. 0 means success.

6.5.113 cg_iric_write_sol_polydata_integer_f

- Outputs integer-type calculation results, giving a value for a polygon or polyline.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.138: Arguments of cg_iric_write_sol_polydata_integer_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	integer	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.114 cg_iric_write_sol_polydata_real_f

- Outputs double-precision real-type calculation results, giving a value for a polygon or polyline.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.139: Arguments of cg_iric_write_sol_polydata_real_f

Variable name	Type	I/O	Description
label	character*	I	Name of the value to be output.
val	double precision	I	Value to be output
ier	integer	O	Error code. 0 means success.

6.5.115 iric_check_cancel_f

- Checks whether user canceled solver execution

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.140: Arguments of iric_check_cancel_f

Variable name	Type	I/O	Description
canceled	integer	O	If canceled 1 returned

6.5.116 iric_check_lock_f

- Checks whether the CGNS file is locked by GUI

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.141: Arguments of iric_check_lock_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
locked	integer	O	If locked 1 returned.

6.5.117 iric_write_sol_start_f

- Inform the GUI that the solver started outputting result

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.142: Arguments of iric_write_sol_start_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
ier	integer	O	Error code. 0 means success.

6.5.118 iric_write_sol_end_f

- Inform the GUI that the solver finished outputting result

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

This function does not exists for Python.

Arguments

Table 6.143: Arguments of iric_write_sol_end_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
ier	integer	O	Error code. 0 means success.

6.5.119 cg_iric_flush_f

- Flush calculation result into CGNS file

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.144: Arguments of cg_iric_flush_f

Variable name	Type	I/O	Description
filename	character(*)	I	Filename
fid	integer	I/O	File ID
ier	integer	O	Error code. 0 means success.

6.5.120 cg_iric_read_sol_count_f

- Reads the number of calculation result

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.145: Arguments of cg_iric_read_sol_count_f

Variable name	Type	I/O	Description
count	integer	O	The number of the calculation result
ier	integer	O	Error code. 0 means success.

6.5.121 cg_iric_read_sol_time_f

- Reads the time value

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.146: Arguments of cg_iric_read_sol_time_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
time	double precision	O	Time
ier	integer	O	Error code. 0 means success.

6.5.122 cg_iric_read_sol_iteration_f

- Reads the loop iteration value

Format (FORTRAN)

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.147: Arguments of cg_iric_read_sol_iteration_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
iteration	integer	O	Iteration value
ier	integer	O	Error code. 0 means success.

6.5.123 cg_iric_read_sol_baseiterative_integer_f

- Reads the integer-type calculation result value

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.148: Arguments of cg_iric_read_sol_baseiterative_integer_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	integer	O	Value
ier	integer	O	Error code. 0 means success.

6.5.124 cg_iric_read_sol_baseiterative_real_f

- Reads the double-precision real-type calculation result value

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.149: Arguments of cg_iric_read_sol_baseiterative_real_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	double precision	O	Value
ier	integer	O	Error code. 0 means success.

6.5.125 cg_iric_read_sol_baseiterative_string_f

- Reads the string-type calculation result value

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.150: Arguments of cg_iric_read_sol_baseiterative_string_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	character(*)	O	Value
ier	integer	O	Error code. 0 means success.

6.5.126 cg_iric_read_sol_gridcoord2d_f

- Reads the 2D structured grid (for moving grid calculation)

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.151: Arguments of cg_iric_read_sol_gridcoord2d_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
x	double precision, dimension(:), allocatable	O	x coordinates
y	double precision, dimension(:), allocatable	O	y coordinates
ier	integer	O	Error code. 0 means success.

6.5.127 cg_iric_read_sol_gridcoord3d_f

- Reads the 3D structured grid (for moving grid calculation)

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.152: Arguments of cg_iric_read_sol_gridcoord3d_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
x	double precision, dimension(:), allocatable	O	Xcoordinates
y	double precision, dimension(:), allocatable	O	Y coordinates
z	double precision, dimension(:), allocatable	O	Z coordinates
ier	integer	O	Error code. 0 means success.

6.5.128 cg_iric_read_sol_integer_f

- Reads the integer-type calculation result, having a value for each grid node.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.153: Arguments of cg_iric_read_sol_integer_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	integer, dimension(:,:),allocatable	O	Value (In case of 3D grid, integer, dimension(:,:,:), allocatable)
ier	integer	O	Error code. 0 means success.

6.5.129 cg_iric_read_sol_real_f

- Reads the double-precision real-type calculation result, having a value for each grid node.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.154: Arguments of cg_iric_read_sol_real_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	double precision, dimension(:,:), allocatable	O	Value (In case of 3D grid, double precision, dimension(:,:,:), allocatable)
ier	integer	O	Error code. 0 means success.

6.5.130 cg_iric_read_sol_cell_integer_f

- Reads the integer-type calculation result, having a value for each grid cell.

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.155: Arguments of cg_iric_read_sol_cell_integer_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	integer, dimension(:,:),allocatable	O	Value (In case of 3D grid, integer, dimension(:,:,:), allocatable)
ier	integer	O	Error code. 0 means success.

6.5.131 cg_iric_read_sol_cell_real_f

- Reads the double-precision real-type calculation result, having a value for each grid cell.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.156: Arguments of cg_iric_read_sol_cell_real_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	double precision, dimension(:,:), allocatable	O	Value (In case of 3D grid, double precision, dimension(:,:,:), allocatable)
ier	integer	O	Error code. 0 means success.

6.5.132 cg_iric_read_sol_iface_integer_f

- Reads the integer-type calculation result, having a value for each grid edge at i-direction.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.157: Arguments of cg_iric_read_sol_iface_integer_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	integer, dimension(:,:), allocatable	O	Value
ier	integer	O	Error code. 0 means success.

6.5.133 cg_iric_read_sol_iface_real_f

- Reads the double-precision real-type calculation result, having a value for each grid edge at i-direction.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.158: Arguments of cg_iric_read_sol_iface_real_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	double precision, dimension(:,:), allocatable	O	Value
ier	integer	O	Error code. 0 means success.

6.5.134 cg_iric_read_sol_jface_integer_f

- Reads the integer-type calculation result, having a value for each grid edge at j-direction.

Format (FORTRAN)

Format (C/C++)

Format (Python)

Arguments

Table 6.159: Arguments of cg_iric_read_sol_jface_integer_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	integer, dimension(:,:), allocatable	O	Value
ier	integer	O	Error code. 0 means success.

6.5.135 cg_iric_read_sol_jface_real_f

- Reads the double-precision real-type calculation result, having a value for each grid edge at j-direction.

Format (FORTRAN)

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.160: Arguments of cg_iric_read_sol_jface_real_f

Variable name	Type	I/O	Description
step	integer	I	Result Step Number
label	character(*)	I	Name
val	double precision, dimension(:,:), allocatable	O	Value
ier	integer	O	Error code. 0 means success.

6.5.136 cg_iric_write_errorcode_f

- Outputs error code

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.161: Arguments of cg_iric_write_errorcode_f

Variable name	Type	I/O	Description
code	integer	I	The error code that the grid generating program returns.
ier	integer	O	Error code. 0 means success.

6.5.137 cg_close_f

- Closing the CGNS file

Format (FORTRAN)

[redacted]

Format (C/C++)

[redacted]

Format (Python)

[redacted]

Arguments

Table 6.162: Arguments of cg_close_f

Variable name	Type	I/O	Description
fid	integer	I	File ID
ier	integer	O	Error code. 0 means success.

CHAPTER 7

Other Informations

7.1 Handling command line arguments in Fortran programs

When iRIC launches solvers (or grid generating programs), the name of calculation data file (or grid generating data file) is passed as an argument. So, solvers (or grid generating programs) have to process the file name and opens that file.

In FORTRAN, the functions prepared for handling arguments are different by compilers. In this section, functions for handling arguments are explained for Intel Fortran Complier and GNU Fortran compiler.

7.1.1 Intel Fortran Compiler

Obtain the number of command line arguments using nargs(), and obtain the argument value using getarg().

List 7.1: Example source code for reading arguments for Intel Fortran Compiler

7.1.2 GNU Fortran, G95

Obtain the number of command line arguments using iargc(), and obtain the argument value using getarg().

Note that nargs(), getargs() in GNU Fortran has different specification to those in Intel Fortran Compiler.

List 7.2: Example source code for reading arguments for GNU Fortran or G95

```
1
2
3
4
5
6
7
8
```

7.2 Linking iRIClib, cgnslib using Fortran

When you develop solvers (or grid generating programs), you have to link the program with iRIClib and cgnslib. You have to use different library files for different compilers like Intel Fortran Compiler and GNU Fortran. Table 7.1 shows the files prepared for each compiler.

For header file, "libcgns_f.h", "iriclib_f.h" can be used for all compilers commonly.

Table 7.1: Files prepared fore each compiler

Compiler	iRIClib library	cgnslib libraray
Intel Fortran Compiler	iriclib_x64_ifort.lib	cgnsdll_x64_ifort.lib
GNU Fortran(gfortran)	iriclib.lib	cgnsdll.lib

We will explain the procedure to compile the source code (solver.f). We assume that the settings for compilers (like path settings) are already finished.

7.2.1 Intel Fortran Compiler (Windows)

Put solver.f, cgnsdll_x64_ifort.lib, iriclib_x64_ifort.lib, cgnslib_f.h, iriclib_f.h in a same folder, move to that folder with command prompt, and run the following command to create an executable file named solver.exe.

```
```

When compiling is done, a file named solver.exe.manifest is also created. When copying the solver to another machine, make sure to copy this file and to place them together in the same folder.

7.2.2 GNU Fortran

Put solver.f, cgnsdll.lib, iriclib.lib, cgnslib_f.h, iriclib_f.h in a same folder, move to that folder with command prompt, and run the following command to create an executable file named solver.exe.

```
```

7.3 Special names for grid attributes and calculation results

In iRIC, some special names for grid attribute and calculation results are defined for certain purposes. Use those names when the solver uses the grid attributes or calculation results that match the purposes.

7.3.1 Grid attributes

Table 7.2 shows the special names defined for grid attributes.

Table 7.2: Special names for grid attributes

Name	Description
Elevation	Grid attribute that contains elevation of grid nodes (Unit: meter). Define "Elevation" attribute as an attribute defined at grid node, with real number type.

When you use "Elevation" for grid attribute, define an Item element as a child of GridRelatedCondition element, like List 7.3. You can change caption attribute value to an arbitrary value.

List 7.3: Example of "Elevation" element definition

When you create a grid generating program and want to output elevation value, use name "Elevation". iRIC will automatically load "Elevation" value.

List 7.4 shows an example of code written in Fortran.

List 7.4: Example of source code to output elevation value in grid generating program

7.3.2 Calculation results

Table 7.3 shows the special names defined for calculation results. Specify these names as arguments of subroutines defined in iRIClib.

List 7.5 shows an example of solver source code that outputs all special calculation result.

Table 7.3: Special names for calculation results

Name	Description	Required
Elevation	Outputs bed elevation (unit: meter). Output the value as real number calculation result. You can add unit in the tail as the part of the name, like "Elevation(m)".	Yes
WaterSurfaceElevation	Outputs water surface elevation (unit: meter). Output the value as real number calculation result. You can add unit in the tail, like "WaterSurfaceElevation(m)".	
IBC	Valid/invalid flag. At invalid region (i. e. dry region), the value is 0, and at valid region (i. e. wet region), the value is 1.	

List 7.5: Example of source code to output calculation results with the special names

7.4 Information on CGNS file and CGNS library

7.4.1 General concept of CGNS file format

CGNS, which stands for CFG General Notation System, refers to a general-purpose file format for storing data for use in numeric hydrodynamics. It can be used across various platforms of different OSes and CPUs. In addition to its standard data format being defined for use in numeric hydrodynamics, it has expandability that allows the addition of elements specific to each solver.

An input/output library for CGNS, called cgnslib, is provided. It can be used in the following languages.

- C, C++
- FORTRAN
- Python

Originally developed by the Boeing Company and NASA, it is currently maintained by an open-source community.

7.4.2 How to view a CGNS file

This section describes how to view a CGNS file that has been created by iRIC using HDFView. HDFView is a software tool published as free software.

Installing HDFView

First, install HDFView. The installer of HDFView can be downloaded from

https://www.hdfgroup.org/downloads/index.html

From the HDF web page, click the "Current Release" link in Figure 7.1. You will be taken to download page. On this screen, click on the file that fits your environment (64 bit or 32 bit) to download.

by unzipping and running the installer, you can install HDFView.

Viewing a CGNS file using HDFView

Launch HDFView and view a CGNS file.

To do so, first launch HDFView from the start menu. Then, from the following menu, select the CGNS to open.

File –> Open

Please note that "*.cgn" is not included in the open target in default. Switch file type to "all files" to select CGNS file as open target.

An example of the HDFView screen after opening a CGNS file is shown in Figure 7.3.

Figure 7.1: HDF group web page

Figure 7.2: Download page

Figure 7.3: Example of ADFviewer screen

In the left side of the screen, the tree structure of the CGNS file contents is shown. When you double click on the item in the tree structure, The data contained in that node is displayed in the main region.

7.4.3 Reference URLs

For information on CGNS files and CGNS libraries, refer to the URLs in Table 7.4 .

Table 7.4: Reference URLs for CGNS file format CGNS libraries

Item	URL
Homepage	http://cgns.sourceforge.net/
Function reference	http://cgns.github.io/CGNS_docs_current/midlevel/index.html
Data structure inside a CGNS file	http://cgns.github.io/CGNS_docs_current/sids/index.html
Sample programs	http://cgns.github.io/CGNS_docs_current/user/examples.html

7.5 Registering file to Repository for iRIC installer build

7.5.1 Abstract

iRIC project uses we service github for managing the files to publish through iRIC offline installer, and online update.

Solver developer can do the following things by registering the files for your newest solvers, to github:

- Update the solver that is going to be bundled to offline installer, that is going to be built next time.
- Make iRIC users who have already installed iRIC can get the newest solver through online update, using menu [Option] -> [Maintainance].

In this section the procedure to register the files for newest solver to github.

7.5.2 Procedure

Solver developer can register the files through the procedure below:

1. Install Subversion client software (needed only for the first time)
2. Get the folder from server (checkout)
3. Copy new files
4. Register new files to server (commit)

You can register files through two verson management systems: Subversion and git. In this document, the way to use Subversion is described, because it is more simple than git.

The detail of the procedure is described below:

7.5.3 Install Subversion client software (needed only for the first time)

Install

Install the client software for Subversion. In this procedure, we adopt TortoiseSVN, that is the de facto standard Subversion client for Windows.

Access the following URL, and download the installer for TortoiseSVN.

https://tortoisesvn.net/downloads.html

On the page there are two buttons for downloading installer: One for 32bit OS, another for 64bit OS. Please download the installer that is suitable your environment.

When installer finished installing, restart Windows.

Setting

If you use an environment where you need to access internet through Proxy server, Please setup the setting in the procedure below:

Select the menu below, from the right-clicking menu of explorer:

TortoiseSVN –> Settings

The Setting dialog is shown. Please select "Network" in the tree view shown in the left side of the dialog, and page like Figure 7.4 is shown. Setup the setting to fit your environment, and click on [OK] button.

Figure 7.4: [TortoiseSVN Setting] dialog

7.5.4 Get the folder from server (checkout)

Checkout the folder, that stores the solver you want to update, that is a subfolder of the following URL:

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages

For example, in case of FaSTMECH, please checkout the following URL:

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages/solver.fastmech

In the procedure below, the way to get the folder for FaSTMECH is shown.

Create folder

Create the folder in which you are going to save files you get from the server.

In this example, we create a folder e:tmpfastmech.

Get the folder from server

Get the folder from server, using TortoiseSVN.

On the explorer, select the folder that you've created in the step above, and select the folloing menu from the right-clicking menu:

SVN Checkout

Then, the dialog in Figure 7.5 is shown.

Figure 7.5: The dialog to checkout files

In the text box next to [URL], input the following URL.

https://github.com/i-RIC/online_update.git/trunk/dev_src/packages

Then, click on the button with caption [. . .] next to the text box. Then, the dialog in Figure 7.6 is shown.

In this dialog, please select the folder that stores the files for the solver you want to update (In this case "solver.fastmech"), and click on [OK]. Then, the [URL] is updated.

On the dialog Figure 7.5, please check again that URLand are set up correctly, and then click on [OK].

Then the dialog like Figure 7.7 is shown, and it starts downloading files from the server.

When downloading files finishes, explorer looks like in Figure 7.8. You'll notice that the files checked out from the server is shown with check mark icon.

Figure 7.6: The dialog to checkout files (select folder)

Figure 7.7: File checkout progress dialog

Figure 7.8: The example of explorer to show files checked out from server

7.5.5 Copy new files

Copy new files that you want to bundle to installer, to the folder you've checked out in the step above.

When you copy files, The icon next to each files will be like below:

- Files that are overwritten is shown with an icon mark [!].
- Files that are copyed as new files is shown without an additional icon mark.

To copy files added as new files to the server, select the file, and select the file below from the right-clicking menu:

TortoiseSVN –> Add

After you do the step above, the file will be shown with an icon mark [+].

Figure 7.9 shows an example of explorer after overwriting "Fastmech.exe", and adding "newdll.dll".

Warning

When you update solver, you have to update not only the exolver executable files, but also *definition.xml*, to update the value of version number. This is because [iRIC maintainance] can not recognize that the solver is updated, if the version number is the same.

The version number you have to update is stored as *version* attribute of *SolverDefinition* element in *definition.xml*.

7.5.6 Register new files to server (commit)

Register the new files to server.

Select the folder in which you've registered new files, and select the menu below, from right-clicking menu:

SVN Commit

Then the dialog in Figure 7.10 is shown.

Figure 7.9: The example of explorer to after copying files

Figure 7.10: The dialog to commit new files

Make sure that the files you want to update or add are shown with check boxes checked, Add log message about the update, and click on [OK].

The dialog in Figure 7.11 will be shown. Please input the Username and Password, and click on [OK].

Figure 7.11: [Authentication] dialog

Please contact the adninistrator of iRIC, to know the username and password.

CHAPTER 8

To Readers

- Please indicate that using the iRIC software, if you publish a paper with the results using the iRIC software.
- The datasets provided at the Web site are sample data. Please use them only for test computation.
- Let us know your suggestions, comments and concerns at http://i-ric.org.