
Chapter 3. Reading and Writing Files

During a FLUENT session you may need to import and export several kinds of files. Files that are read include grid, case, data, profile, Scheme, and journal files, and files that are written include case, data, profile, journal, and transcript files. FLUENT also has features that allow you to save panel layouts and hardcopies of graphics windows. You can also export data for use with various visualization and postprocessing tools. These operations are described in the following sections.

- Section 3.1: Shortcuts for Reading and Writing Files
- Section 3.2: Reading Mesh Files
- Section 3.3: Reading and Writing Case and Data Files
- Section 3.4: Reading FLUENT/UNS and RAMPANT Case and Data Files
- Section 3.5: Importing FLUENT 4 Case Files
- Section 3.6: Importing FIDAP Neutral Files
- Section 3.7: Creating and Reading Journal Files
- Section 3.8: Creating Transcript Files
- Section 3.9: Reading and Writing Profile Files
- Section 3.10: Reading and Writing Boundary Conditions
- Section 3.11: Writing a Boundary Grid
- Section 3.12: Saving Hardcopy Files
- Section 3.13: Exporting Data

- Section 3.14: Grid-to-Grid Solution Interpolation
- Section 3.15: Loading Scheme Source Files
- Section 3.16: The `.fluent` File
- Section 3.17: Saving the Panel Layout
- Section 3.18: Formats of FLUENT Case and Data Files

3.1 Shortcuts for Reading and Writing Files

FLUENT has several features that make the reading and writing of files very convenient:

- Automatic appending or detection of default filename suffixes
- Binary file reading and writing
- Automatic detection of file format (text/binary)
- Recent file list
- Reading and writing of compressed files
- Tilde expansion
- Automatic numbering of files
- Ability to disable the overwrite confirmation prompt

3.1.1 Default File Suffixes

There are default file suffixes associated with each type of file that FLUENT reads or writes. For several commonly used files, the solver will automatically append or detect the appropriate suffix when you specify the first part of the filename (the prefix). For example, to write a case file named `myfile.cas`, you can specify just `myfile` and FLUENT will automatically append `.cas`. Similarly, to read the case file named `myfile.cas` into the solver, you can specify just `myfile` and FLUENT will automatically search for a file of that name with the suffix `.cas`.

3.1 Shortcuts for Reading and Writing Files

FLUENT will automatically append and detect default suffixes for case and data files. In addition, FLUENT will automatically do this for PDF files and ray files. The appropriate default file suffix will appear in the Select File dialog box for each type of file.

3.1.2 Binary Files

When you write a case, data, or ray file, FLUENT will save a binary file by default. Binary files take up less space than text files and can be read and written by FLUENT more quickly. Note, however, that you cannot read and edit a binary file, as you can do for a text file. If you need to save a text file, turn off the Write Binary Files option in the Select File dialog box when you are writing the file.

! FLUENT can read binary files that were saved on different platforms, but other products, such as TGrid, cannot. If you are planning to read a case file into TGrid on a different platform, you should save a text file from FLUENT.

3.1.3 Detecting File Format

When you read a case, data, grid, PDF, or ray file, the solver will automatically determine whether it is a text (formatted) or binary file.

3.1.4 Recent File List

There is a shortcut for reading in case and data files that you have recently used in FLUENT. At the bottom of the File/Read submenu there is a list of the four FLUENT case files that you most recently read or wrote. To read one of these files into FLUENT, simply select it in the list. This shortcut allows you to conveniently read a recently-used file without having to select it in the Select File dialog box.

Note that the files listed are the four case files that you have used most recently in any FLUENT session, so some may not be appropriate for your current session (e.g., a 3D case file will be listed even if you are running a 2D version of FLUENT). Also, if you read a case file using this shortcut, the corresponding data file will be read only if it has the same base name as the case file (e.g., `file1.cas` and `file1.dat`) and it was

read (or written) with the case file the last time the case file was read (or written).

3.1.5 Reading and Writing Compressed Files

Reading Compressed Files

You can use the Select File dialog box to read files that have been compressed using `compress` or `gzip`. If you select a compressed file with a `.Z` extension, FLUENT will automatically invoke `zcat` to import the file's data. If you select a compressed file with a `.gz` extension, the solver will invoke `gunzip` to import the file's data.

For example, if you select a file named `flow.msh.gz`, the solver will report the following message,

```
Reading "| gunzip -c flow.msh.gz"...
```

indicating that the result of the `gunzip` is imported into FLUENT via an operating system pipe.

You can also type in the filename without any suffix (e.g., if you are not sure whether or not the desired file is compressed). First, the solver attempts to open a file with the input name. If it cannot find a file with that name, it attempts to locate files with default suffixes and extensions appended to the name. For example, if you enter the name `file-name`, the solver traverses the following list until it finds an existing file:

- `file-name`
- `file-name.gz`
- `file-name.Z`
- `file-name.suffix`
- `file-name.suffix.gz`
- `file-name.suffix.Z`

3.1 Shortcuts for Reading and Writing Files

where suffix is a common extension to the file, such as `.cas` or `.msh`. The solver reports an error if it fails to find an existing file with one of these names.

- ! For Windows systems, only files that were compressed with `gzip` (i.e., files with a `.gz` extension) can be read. Files that were compressed with `compress` cannot be read into FLUENT on a Windows machine.
- ! Do not read a compressed ray file; FLUENT will not be able to access the ray tracing information properly from a compressed ray file.

Writing Compressed Files

You can use the Select File dialog box to write a compressed file by appending a `.Z` or `.gz` extension onto the file name.

For example, if you enter `flow.gz` as the name for a case file, the solver will report the following message:

```
Writing "| gzip -cfv > flow.cas.gz"...
```

The status message indicates that the case file information is being piped into the `gzip` command, and that the output of the compression command is being redirected to the file with the specified name. In this particular example, the `.cas` extension was added automatically.

- ! For Windows systems, compression can be performed only with `gzip`; that is, you can write a compressed file by appending `.gz` to the name, but appending `.Z` will not result in file compression.
- ! Do not write a compressed ray file; FLUENT will not be able to access the ray tracing information properly from a compressed ray file.

3.1.6 Tilde Expansion (UNIX Systems Only)

On UNIX systems, if you specify `~/` as the first two characters of a filename, the `~` will be expanded to be your home directory. Similarly, you can start a filename with `~username/`, and the `~username` will be expanded to the home directory of “username”. If you specify `~/file`

as the case file to be written, FLUENT will save the file `file.cas` in your home directory. You can specify a subdirectory of your home directory as well: if you enter `~/cases/file.cas`, FLUENT will save the file `file.cas` in the `cases` subdirectory.

3.1.7 Automatic Numbering of Files

There are several special characters that you can include in a filename. Using one of these character strings in your filename provides a shortcut for numbering the files based on various parameters (i.e., iteration number, time step, or total number of files saved so far), because you will not need to enter a new filename each time you save a file. (See also Section 3.3.4 for information about saving and numbering case and data files automatically.)

- For unsteady calculations, you can save files with names that reflect the time step at which they are saved by including the character string `%t` in the file name. For example, you can specify `contours-%t.ps` for the file name, and the solver will save a file with the appropriate name (e.g., `contours-0001.ps` if the solution is at the first time step).
- To save a file with a name that reflects the iteration at which it is saved, use the character string `%i` in the file name. For example, you can specify `contours-%i.ps` for the file name, and the solver will save a file with the appropriate name (e.g., `contours-0010.ps` if the solution is at the 10th iteration).
- To save a hardcopy file with a name that reflects the total number of hardcopy files saved so far in the current solver session, use the character string `%n` in the file name.

This option can be used only for hardcopy files.

! Note that when you use the character strings described above in a filename, you will not be prompted for confirmation before FLUENT overwrites an existing file with the same name. Say, for example, that you are repeatedly using the filename `myfile-%t.ps` to save hardcopies with

names that reflect the current time step. If you have already saved `myfile-0001.ps` at the first time step and then you restart the calculation and save another hardcopy at the (new) first time step, the solver will overwrite the original `myfile-0001.ps` without checking with you first.

3.1.8 Disabling the Overwrite Confirmation Prompt

By default, if you ask FLUENT to write a file with the same name as an existing file in that directory, it will ask you to confirm that it is “OK to overwrite” the existing file. If you do not want the solver to ask you for confirmation before it overwrites existing files, you can choose the `file/confirm-overwrite?` text command and answer `no`.

3.2 Reading Mesh Files

Mesh files, also known as grid files, are created with the GAMBIT, TGrid, GeoMesh, and preBFC grid generators, or by several third-party CAD packages. A grid file is—from FLUENT’s point of view—simply a subset of a case file (described in Section 3.3.1). The grid file contains the coordinates of all the nodes, connectivity information that tells how the nodes are connected to one another to form faces and cells, and the zone types and numbers (e.g., wall-1, pressure-inlet-5, symmetry-2) of all the faces. The grid file does not contain any boundary conditions, flow parameters, or solution parameters. For more information about grids, see Chapter 5.

To read a native-format mesh file (i.e., a mesh file that has been saved in FLUENT format) into the solver, use the `File/Read/Case...` menu item, as described in Section 3.3.1. GAMBIT, TGrid, GeoMesh, and preBFC can all write a native-format mesh file. For more information about reading these files, see Sections 5.3.1, 5.3.2, 5.3.3, and 5.3.4.

For information on importing an unpartitioned mesh file into the parallel solver using the partition filter, see Section 28.4.4.

3.2.1 Reading TGrid Mesh Files

Since TGrid has the same file format as FLUENT, you can read a TGrid mesh into the solver using the File/Read/Case... menu item.

File → **Read** → Case...

For more information about reading TGrid mesh files, see Section 5.3.3.

3.2.2 Reading and Importing GAMBIT and GeoMesh Mesh Files

If you create a FLUENT 5/6, FLUENT/UNS, or RAMPANT grid in GAMBIT or GeoMesh, you can read it into FLUENT using the File/Read/Case... menu item.

File → **Read** → Case...

Selecting the Case... menu item will open the Select File dialog box (see Section 2.1.2), in which you will specify the name of the file to be read.

If you have saved a neutral file from GAMBIT, rather than a FLUENT grid file, you can import it into FLUENT using the File/Import/GAMBIT... menu item.

File → **Import** → GAMBIT...

For more information about importing files from GAMBIT and GeoMesh, see Sections 5.3.1 and 5.3.2.

3.2.3 Reading preBFC Unstructured Mesh Files

Since preBFC's unstructured triangular grids have the same file format as FLUENT, you can read a preBFC triangular mesh into the solver using the File/Read/Case... menu item.

! Note that you must save the file using the MESH-RAMPANT/TGRID command.

File → **Read** → Case...

For more information about reading preBFC mesh files, see Section 5.3.4.

3.2.4 Importing preBFC Structured Mesh Files

You can read a preBFC structured mesh file into FLUENT using the File/Import/preBFC Structured Mesh... menu item.

File → **Import** → preBFC Structured Mesh...

Selecting the preBFC Structured Mesh... menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the preBFC structured mesh file to be read. The solver will read grid information and zone types from the preBFC mesh file. For more information about importing preBFC mesh files, see Section 5.3.4.

3.2.5 Importing ANSYS Files

To read an ANSYS file, use the File/Import/ANSYS... menu item.

File → **Import** → ANSYS...

Selecting this menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the ANSYS file to be read. The solver will read grid information from the ANSYS file. For more information about importing ANSYS files, see Section 5.3.6.

3.2.6 Importing I-DEAS Universal Files

I-DEAS Universal files can be read into FLUENT with the File/Import/IDEAS Universal... menu item.

File → **Import** → IDEAS Universal...

Selecting the IDEAS Universal... menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the I-DEAS Universal file to be read. The solver will read grid information and zone types from the I-DEAS Universal file. For more information about importing I-DEAS Universal files, see Section 5.3.6.

3.2.7 Importing NASTRAN Files

NASTRAN files can be read into FLUENT with the File/Import/NASTRAN... menu item.

File → **Import** → NASTRAN...

Selecting the NASTRAN... menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the NASTRAN file to be read. The solver will read grid information from the NASTRAN file. For more information about importing NASTRAN files, see Section 5.3.6.

3.2.8 Importing PATRAN Neutral Files

To read a PATRAN Neutral file zoned by named components (that is, a file in which you have grouped nodes with the same specified group name), use the File/Import/PATRAN... menu item.

File → **Import** → PATRAN...

Selecting this menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the PATRAN Neutral file to be read. The solver will read grid information from the PATRAN Neutral file. For more information about importing PATRAN Neutral files, see Section 5.3.6.

3.2.9 Importing Meshes and Data in CGNS Format

You can import meshes in CGNS (CFD general notation system) format into FLUENT using the File/Import/CGNS/Mesh... menu item.

File → **Import** → **CGNS** → Mesh...

Note that this feature is available on a limited number of platforms (listed in the release notes).

To import a mesh and the corresponding CGNS data, you can use the File/Import/CGNS/Mesh & Data... menu item.

File → **Import** → **CGNS** → Mesh & Data...

3.2.10 Reading a New Grid File

After you have set up a case file using a particular grid, you can merge a new grid with the existing boundary conditions, material properties,

solution parameters, etc. One situation in which you might use this feature is if you have generated a better mesh for a problem you have been working on, and you do not want to reenter all of the boundary conditions, properties, and parameters. You can read in the new grid as long as it has the same zone structure as the original grid.

- ! The new and original grids should have the same zones, numbered in the same order. A warning is issued if they do not, because inconsistencies could create problems with the boundary conditions.

New grids are read using the `file/reread-grid` command in the text interface.

`file` → `reread-grid`

3.3 Reading and Writing Case and Data Files

Information related to your FLUENT simulation is stored in two files: the case file and the data file. The commands for reading and writing these files are described in the following sections, along with commands for the automatic saving of case and data at specified intervals. A description of the case and data file formats is provided in Section 3.18.

- Section 3.3.1: Reading and Writing Case Files
- Section 3.3.2: Reading and Writing Data Files
- Section 3.3.3: Reading and Writing Case and Data Files Together
- Section 3.3.4: Automatic Saving of Case and Data Files

Note that FLUENT can read and write either text or binary case and data files; binary files require less storage space and are faster to read and write. To specify whether to write a binary or text file, use the **Write Binary Files** check button in the **Select File** dialog box (see Section 2.1.2). In addition, you can read and write either text or binary files in compressed formats (see Section 3.1.5). FLUENT automatically detects the file type when reading.

! If you adapt the grid, you must save a new case file as well as a data file. Otherwise the new data file will not correspond to the case file (they will have different numbers of cells, for example). FLUENT will warn you when you try to exit the program if you have not saved an up-to-date case or data file.

3.3.1 Reading and Writing Case Files

Case files contain the grid, boundary conditions, and solution parameters for a problem, as well as information about the user interface and graphics environment. For information about the format of case files see Section 3.18. The commands used for reading case files can also be used to read native-format grid files (as described in Section 3.2) because the grid information is a subset of the case information.

3.3 Reading and Writing Case and Data Files

You can read a case file using the Select File dialog box (see Section 2.1.2) invoked by selecting the File/Read/Case... menu item. To write a case file, use the Select File dialog box invoked by selecting the File/Write/Case... menu item.

File → Read → Case...

File → Write → Case...

See Section 1.5.2 for information about executing the appropriate version automatically based on the case file that is read.

Default Suffixes

By convention, case file names are composed of a root with the suffix **.cas**. If you conform to this convention, you do not have to type the suffix when prompted for a filename; it will be added automatically. When FLUENT reads a case file, it first looks for a file with the exact name you typed. If a file with that name is not found, it uses the procedure described in Section 3.1.5 to search for the case file. When FLUENT writes a case file, **.cas** will be added to the name you type unless the name already ends with **.cas**.

3.3.2 Reading and Writing Data Files

Data files contain the values of the flow field in each grid element and the convergence history (residuals) for that flow field. For information about the format of data files see Section 3.18.

You can read a data file using the Select File dialog box (see Section 2.1.2) invoked by selecting the File/Read/Data... menu item. Likewise, you can write a data file using the Select File dialog box invoked by selecting the File/Write/Data... menu item.

File → Read → Data...

File → Write → Data...

Default Suffixes

By convention, data file names are composed of a root with the suffix `.dat`. If you conform to this convention, you do not have to type the suffix when prompted for a filename; it will be added automatically. When FLUENT reads a data file, it first looks for a file with the exact name you typed. If a file with that name is not found, it uses the procedure described in Section 3.1.5 to search for the data file. When it writes a data file, `.dat` will be added to the name you type unless the name already ends with `.dat`.

3.3.3 Reading and Writing Case and Data Files Together

A case file and a data file together contain all the information required to restart a solution. Case files contain the grid, boundary conditions, and solution parameters. Data files contain the values of the flow field in each grid element and the convergence history (residuals) for that flow field.

You can read a case file and a data file together by using the Select File dialog box (see Section 2.1.2) invoked by selecting the File/Read/Case & Data... menu item. To read both files, select the appropriate case file, and the corresponding data file (same name with `.dat` suffix) will also be read. To write a case file and a data file, select the File/Write/Case & Data... menu item.

File → Read → Case & Data...

File → Write → Case & Data...

See Section 1.5.2 for information about executing the appropriate version automatically based on the case file that is read.

3.3.4 Automatic Saving of Case and Data Files

You can request FLUENT to automatically save case and data files at specified intervals during a calculation. This is especially useful for time-dependent calculations, since it will allow you to save the results at different time steps without stopping the calculation and performing the save manually. You can also use the auto-save feature for steady-state

3.3 Reading and Writing Case and Data Files

problems, to examine the solution at different stages in the iteration history.

Automatic saving is specified using the Autosave Case/Data panel (Figure 3.3.1).

File → **Write** → Autosave...

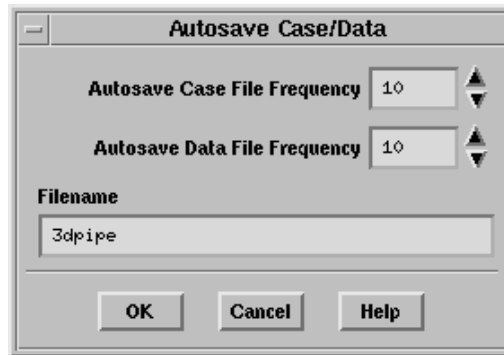


Figure 3.3.1: The Autosave Case/Data Panel

You must set the frequency of saves for case and/or data files, and the root filename. Enter the case-saving frequency in the Autosave Case File Frequency field, and the data-saving frequency in the Autosave Data File Frequency field. Both values are set to zero by default, indicating that no automatic saving is performed.

For steady flows you will specify the frequency in iterations, and for unsteady flows you will specify it in time steps (unless you are using the explicit time stepping formulation, in which case you will specify the frequency in iterations). If you define an Autosave Case File Frequency of 10, for example, a case file will be saved every 10 iterations or time steps.

Enter the filename to be used for the autosave files in the Filename field. The iteration or time-step number and an appropriate suffix (`.cas` or `.dat`) will be added to the specified root name. If the specified Filename ends in `.gz` or `.Z`, appropriate file compression will be performed. (See Section 3.1.5 for details about file compression.)

All of the autosave inputs are stored in the case file.

3.4 Reading FLUENT/UNS and RAMPANT Case and Data Files

Case files created by FLUENT/UNS 3 or 4 or RAMPANT 2, 3, or 4 can be read into FLUENT in the same way that current case files are read (see Section 3.3). If you read a case file created by FLUENT/UNS, FLUENT will select the Segregated solver in the Solver panel. If you read a case file created by RAMPANT, FLUENT will select the Coupled Explicit solver formulation in the Solver panel.

Data files created by FLUENT/UNS 4 or RAMPANT 4 can be read into FLUENT in the same way that current data files are read (see Section 3.3).

3.5 Importing FLUENT 4 Case Files

You can read a FLUENT 4 case file using the File/Import/FLUENT 4 Case... menu item.

File → **Import** → FLUENT 4 Case...

Selecting the FLUENT 4 Case... menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the FLUENT 4 case file to be read. FLUENT will read *only* grid information and zone types from the FLUENT 4 case file. You must specify boundary conditions, model parameters, material properties, and other information after reading this file. For more information about importing FLUENT 4 case files, see Section 5.3.8.

3.6 Importing FIDAP Neutral Files

You can read a FIDAP Neutral file using the File/Import/FIDAP... menu item.

File → **Import** → FIDAP...

Selecting the FIDAP... menu item will invoke the Select File dialog box (see Section 2.1.2), in which you will specify the name of the FIDAP file

to be read. FLUENT will read grid information and zone types from the FIDAP file. You must specify boundary conditions and other information after reading this file. For more information about importing FIDAP Neutral files, see Section 5.3.9.

3.7 Creating and Reading Journal Files

A journal file contains a sequence of FLUENT commands, arranged as they would be typed interactively into the program or entered through the GUI. GUI commands are recorded as Scheme code lines in journal files. FLUENT creates a journal file by recording everything you type on the command line or enter through the GUI. You may also create journal files manually with a text editor. If you want to include comments in your file, be sure to put a ; (semicolon) at the beginning of each comment line. See Section 2.5.1 for an example.

The purpose of a journal file is usually to automate a series of commands rather than entering them repeatedly on the command line. Another use is to produce a record of the input to a program session for later reference, although transcript files are often more useful for this purpose (see Section 3.8).

Command input is taken from the specified journal file until its end is reached, at which time control is returned to the standard input (usually the keyboard). Each line from the journal file is echoed to the standard output (usually the screen) as it is read and processed.

- ! Since a journal file is, by design, just a simple record/playback facility, it knows nothing about the state in which it was recorded or the state in which it is being played back. You should, therefore, try to recreate the state in which the journal was written before you read it into the program. If, for example, your journal file includes an instruction for FLUENT to save a new file with a specified name, you should check that no file with that name exists in your directory before you read in your journal file. If a file with that name *does* exist and you read in your journal file anyway, when the program reaches the write instruction, it will prompt for a confirmation that it is OK to overwrite the old file. Since no response to the confirmation request is contained in the journal file, FLUENT will not be able to continue following the journal's

instructions. Other conditions that may affect the program's ability to perform the instructions contained in a journal file can be created by modifications or manipulations that you make within the program. For example, if your journal file creates several surfaces and displays data on those surfaces, you must be sure to read in appropriate case and data files before reading the journal file.

- ! Only one journal file can be open for recording at a time, but you can write a journal and a transcript file simultaneously. You can also read a journal file at any time.

3.7.1 User Inputs

To start the journaling process, select the **File/Write/Start Journal...** menu item.

File → **Write** → **Start Journal...**

After you enter a name for the file in the **Select File** dialog box (see Section 2.1.2), journal recording will begin and the **Start Journal...** menu item will become the **Stop Journal** menu item. You can end journal recording by selecting **Stop Journal**, or simply by exiting the program.

File → **Write** → **Stop Journal**

You can read a journal file into the program using the **Select File** dialog box invoked by selecting the **File/Read/Journal...** menu item.

File → **Read** → **Journal...**

Journal files are always loaded in the main (i.e., top-level) text menu, regardless of where you are in the text menu hierarchy when you invoke the read command.

3.8 Creating Transcript Files

A transcript file contains a complete record of all standard input to and output from FLUENT (usually all keyboard and GUI input and all screen output). GUI commands are recorded as Scheme code lines in transcript files. FLUENT creates a transcript file by recording everything typed as input or entered through the GUI, and everything printed as output in the text window.

The purpose of a transcript file is to produce a record of the program session for later reference. Because they contain messages and other output, transcript files, unlike journal files (see Section 3.7), cannot be read back into the program.

- ! Only one transcript file can be open for recording at a time, but you can write a transcript and a journal file simultaneously. You can also read a journal file while a transcript recording is in progress.

3.8.1 User Inputs

To start the transcribing process, select the File/Write/Start Transcript... menu item.

File → **Write** → Start Transcript...

After you enter a name for the file in the Select File dialog box (see Section 2.1.2), transcript recording will begin and the Start Transcript... menu item will become the Stop Transcript menu item. You can end transcript recording by selecting Stop Transcript, or simply by exiting the program.

File → **Write** → Stop Transcript

3.9 Reading and Writing Profile Files

Boundary profiles are used to specify flow conditions on a boundary zone of the solution domain. For example, they can be used to prescribe a velocity field on an inlet plane. For more information on boundary profiles, see Section 6.25.

Reading Profile Files

Boundary profile files can be read using the Select File dialog box invoked by selecting the File/Read/Profile... menu item.

File → **Read** → Profile...

Writing Profile Files

You can also create a profile file from the conditions on a specified boundary or surface. For example, you can create a profile file from the outlet conditions of one case and then read that profile into another case and use the outlet profile data as the inlet conditions for the new case.

To write a profile file, you will use the Write Profile panel (Figure 3.9.1).

File → **Write** → Profile...

1. Retain the default option of Define New Profiles.
2. Select the surface(s) from which you want to extract data for the profile(s) in the Surfaces list.
3. Choose the variable(s) for which you want to create profiles in the Values list.
4. Click on the Write... button and specify the profile file name in the resulting Select File dialog box.

FLUENT will save the grid coordinates of the data points in the selected surface(s) and the values of the selected variables at those positions. When you read the profile file back into the solver, the “surface” name

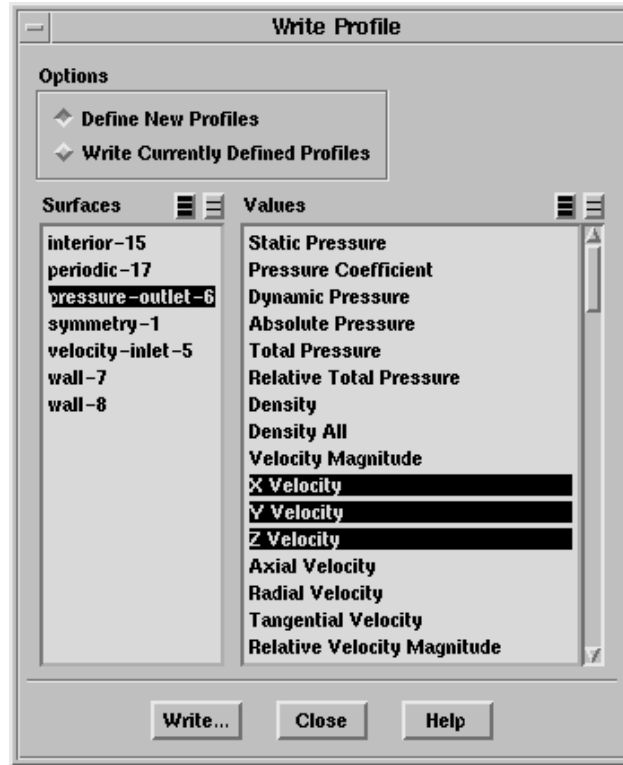


Figure 3.9.1: The Write Profile Panel

will be the profile name and the “value” name will be the field name that appears in the drop-down lists in the boundary condition panels.

If you have made any modifications to the boundary profiles since you read them in (e.g., if you reoriented an existing profile to create a new one), or if you wish to store all the profiles used in a case file in a separate file, you can simply select the Write Currently Defined Profiles option and click on the Write... button. All currently defined profiles will be saved in the file you specify in the resulting Select File dialog box. This file can be read back into the solver whenever you wish to use these profiles again.

3.10 Reading and Writing Boundary Conditions

To save all currently-defined boundary conditions to a file, select the `write-bc` text command and specify a name for the file.

`file` → `write-bc`

FLUENT will write the boundary condition to a file, using the same format as the “zone” section of the case file. See Section 3.18 for details about the case file format.

To read boundary conditions from a file and apply them to the corresponding zones in your model, select the `read-bc` text command.

`file` → `read-bc`

FLUENT will set the boundary conditions in the current model by comparing the zone name associated with each set of conditions in the file with the zone names in the model. If the model does not contain a matching zone name for a set of boundary conditions, those conditions will be ignored.

- ! If you read boundary conditions into a model that contains a different mesh topology (e.g., a cell zone has been removed), you should check the conditions at boundaries within and adjacent to the region of the topological change. This is particularly important for wall zones.

If you want FLUENT to apply a set of conditions to multiple zones with similar names, or to a single zone with a name you’re not completely sure of in advance, you can edit the boundary-condition file you saved with the `write-bc` command to include “wildcards” (*) within the zone names. For example, if you want to apply a particular set of conditions to `wall-12`, `wall-15`, and `wall-17` in your current model, you should edit the boundary-condition file so that the zone name associated with the desired conditions is `wall-*`.

3.11 Writing a Boundary Grid

You can write the boundary zones (surface grid) to a file. This file can be read and used by TGrid to produce a volume mesh. You may find this feature useful if you are unsatisfied with a mesh that you obtained from another grid generation program.

A boundary grid can be written using the Select File dialog box (see Section 2.1.2) invoked by selecting the File/Write/Boundary Grid... menu item.

File → **Write** → Boundary Grid...

3.12 Saving Hardcopy Files

Graphics window displays can be saved in various formats, including TIFF, PICT, and PostScript. There may be slight differences, however, between hardcopies and the displayed graphics windows, since hardcopies are generated using the internal software renderer, while the graphics windows may utilize specialized graphics hardware for optimum performance. Many systems provide a utility to “dump” the contents of a graphics window into a raster file. This is generally the fastest method of generating a hardcopy (since the scene is already rendered in the graphics window), and guarantees that the hardcopy will be identical to the window.

3.12.1 Using the Graphics Hardcopy Panel

To set hardcopy parameters and save hardcopy files, you will use the Graphics Hardcopy panel (Figure 3.12.1).

File → Hardcopy...

The procedure for saving a hardcopy file is listed below, followed by details about each step.

1. Choose the hardcopy file format.
2. (optional) Specify the file type, if applicable.

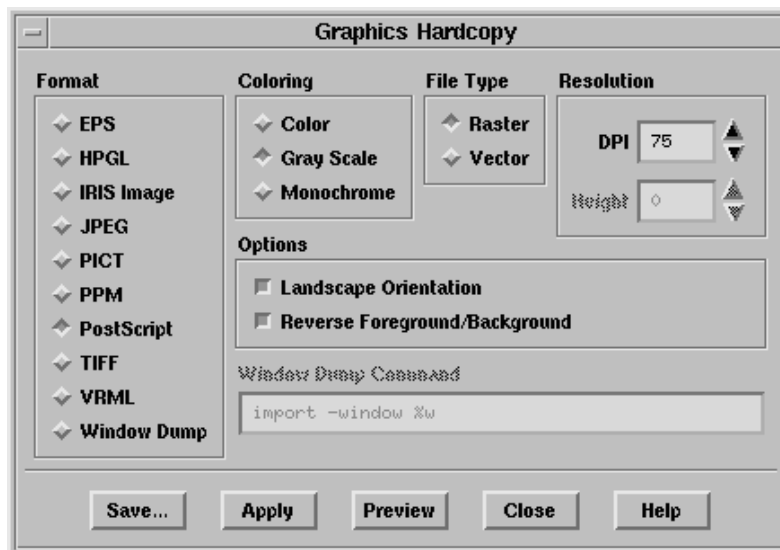


Figure 3.12.1: The Graphics Hardcopy Panel

3. Set the coloring.
4. (optional) Define the resolution, if applicable.
5. Set any of the options described below.
6. If you are generating a window dump, specify the command to be used for the dump.
7. (optional) Preview the result.
8. Click on the **Save...** button and enter the filename in the resulting Select File dialog box. (See Section 3.1.7 for information on special features related to filename specification.)

If you want to save the current hardcopy settings, but you are not ready to save a hardcopy yet, you can click on the **Apply** button instead of the **Save...** button. The applied settings will become the defaults for subsequent hardcopies.

Choosing the Hardcopy File Format

To choose the hardcopy file format, select one of the following items in the Format list:

EPS (Encapsulated PostScript) output is the same as PostScript output, with the addition of Adobe Document Structuring Conventions (v2) statements. Currently, no preview bitmap is included in EPS output. Often, programs that import EPS files use the preview bitmap to display on-screen, although the actual vector PostScript information is used for printing (on a PostScript device). You can save EPS files in raster or vector format.

HPGL is a vector file format designed for pen plotters. The HPGL driver supports a limited set of colors and is not capable of rendering some scenes properly.

IRIS Image is the native raster image file format on SGI computers. The IRIS Image driver may not be available on all platforms.

JPEG is a common raster file format.

PICT is the native graphics file format on Macintosh computers. PICT files may contain either vector or raster information (or both). Typically, “draw” programs generate vector information, while “paint” programs use raster formats. You can choose to save a PICT file in raster or vector format.

PPM output is a common raster file format.

PostScript is a common vector file format. You can also choose to save a PostScript file in raster format.

TIFF is a common raster file format.

VRML is a graphics interchange format that allows export of 3D geometrical entities that you can display in the FLUENT graphics window. This format can commonly be used by VR systems and in particular the 3D geometry can be viewed and manipulated in a web-browser graphics window.

- ! Note that non-geometric entities such as text, titles, color bars, and orientation axis are not exported. In addition, most display or visibility characteristics set in **FLUENT**, such as lighting, shading method, transparency, face and edge visibility, outer face culling, and hidden line removal, are not explicitly exported but are controlled by the software used to view the VRML file.

Window Dump (UNIX systems only) selects a window dump operation for generating the hardcopy. With this format, you will need to specify the appropriate **Window Dump Command**. See below for details.

Choosing the File Type

If you are saving a PostScript, EPS (Encapsulated PostScript), or PICT file, you can choose **Raster** or **Vector** as the **File Type**. A vector file defines the graphics image as a combination of geometric primitives like lines, polygons, and text. A raster file defines the color of each individual pixel in the image. Vector files are usually scalable to any resolution, while raster files have a fixed resolution. Supported vector formats include PostScript, EPS, HPGL, and PICT. Supported raster formats are IRIS image, JPEG, PICT, PPM, PostScript, EPS, and TIFF.

- ! In general, for the quickest print time, you may want to save vector files for simple 2D displays and raster files for complicated scenes.

Specifying the Color Mode

For all formats except the window dump you can specify which type of **Coloring** you want to use for the hardcopy file. For a color-scale copy select **Color**, for a gray-scale copy select **Gray Scale**, and for a black-and-white copy, select **Monochrome**. Note that most monochrome PostScript devices will render **Color** images in shades of gray, but to ensure that the color ramp is rendered as a linearly-increasing gray ramp, you should select **Gray Scale**.

Defining the Resolution

For raster hardcopy files, you can control the resolution of the hardcopy image by specifying the size (in pixels). Set the desired **Width** and **Height** under **Resolution**. If the **Width** and **Height** are both zero, the hardcopy is generated at the same resolution as the active graphics window. (To check the size of the active window in pixels, you can click on **Info** in the **Display Options** panel.)

Note that for PostScript, EPS, and PICT files, you will specify the resolution in dots per inch (DPI) instead of setting the width and height.

Hardcopy Options

For all hardcopy formats except the window dump, you can control two additional settings under **Options**. First, you can specify the orientation of the hardcopy using the **Landscape Orientation** button. If this option is turned on, the hardcopy is made in landscape mode; otherwise, it is made in portrait mode. Second, you can control the foreground/background color using the **Reverse Foreground/Background** option. If this option is enabled, the foreground and background colors of graphics windows being hardcopied will be swapped. This feature allows you to make hardcopies with a white background and a black foreground, while the graphics windows are displayed with a black background and white foreground.

Hardcopy Options for PostScript Files

FLUENT provides options that allow you to save PostScript files that can be printed more quickly. The following options can be found in the **display/set/hardcopy/driver/post-format** text menu:

fast-raster enables a raster file that may be larger than the standard raster file, but will print much more quickly.

raster enables the standard raster file.

rle-raster enables a run-length encoded raster file that will be about the same size as the standard raster file, but will print slightly more

quickly. (This is the default file type.)

vector enables the standard vector file.

(Vector and raster files are described above.)

Window Dumps (UNIX Systems Only)

If you select the Window Dump format, the program will use the specified Window Dump Command to save the hardcopy file. For example, if you want to use **xwd** to capture a window, you will set the Window Dump Command to

```
xwd -id %w >
```

FLUENT will automatically interpret **%w** to be the ID number of the active window when the dump occurs. In the **Select File** dialog box that appears when you click on the **Save...** button, enter the filename for the output from the window dump (e.g., **myfile.xwd**).

If you are planning to make an animation, you might want to save the window dumps into numbered files, using the **%n** variable. To do this, you will use the same Window Dump Command as that shown in the example above, but for the filename in the **Select File** dialog box you will enter **myfile%n.xwd**. Each time you create a new window dump, the value of **%n** will increase by one, so you do not need to tack numbers onto the hardcopy filenames manually.

If you are planning to use the **ImageMagick** animate program, saving the files in MIFF format (the native **ImageMagick** format) is more efficient. In such cases, you should use the **ImageMagick** tool **import**. For the Window Dump Command you will enter

```
import -window %w
```

(which is the default command). In the **Select File** dialog box that appears when you click on the **Save...** button, specify the output format to be MIFF by using the **.miff** suffix at the end of your filename.

The window-dump feature is system- and graphics-driver-specific. Thus the commands available to you for dumping windows will depend heavily on your particular configuration.

Another issue to consider when saving window dumps is that the window dump will capture the window exactly as it is displayed, including resolution, colors, transparency, etc. (For this reason, all of the inputs that control these characteristics are disabled in the **Graphics Hardcopy** panel when you enable the **Window Dump** format.) If you are using an 8-bit graphics display, you might want to use one of the built-in raster drivers (e.g., TIFF) to generate higher-quality 24-bit color output rather than dumping the 8-bit window.

Previewing the Hardcopy Image

Before you save a hardcopy file, you have the option of previewing what the saved image will look like. If you click on **Preview**, the current settings are applied to the active graphics window so that you can investigate the effects of different options interactively before saving the final, approved hardcopy.

3.13 Exporting Data

The current release of FLUENT allows you to export data to Abaqus, ANSYS, ASCII, AVS, CGNS, Data Explorer, EnSight (formerly known as MPGS), FAST, FIELDVIEW, I-DEAS, NASTRAN, PATRAN, and Tecplot. Section 3.13.1 explains how to save data in these formats, and Section 3.13.2 describes each type of file.

Note that, of these formats, only EnSight and FIELDVIEW case and data files can be exported using the parallel version of FLUENT.

- ! If you intend to export data to I-DEAS, you should ensure that your mesh does not contain pyramid elements, as these are currently not supported by I-DEAS.

3.13.1 Using the Export Panel

To write data to one of these products for data visualization and post-processing, you will use the Export panel (Figure 3.13.1).

File → Export...

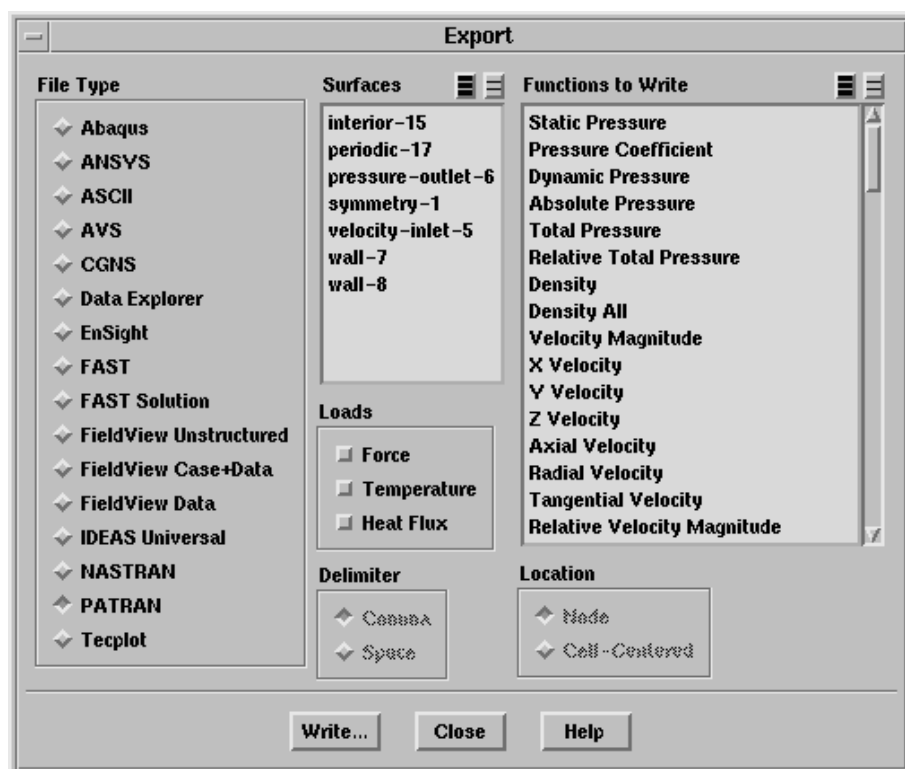


Figure 3.13.1: The Export Panel

The procedure is listed below:

1. Select the type of file you want to write in the File Type list.
2. If you chose Abaqus, ASCII, Data Explorer, IDEAS Universal, NASTRAN, PATRAN, or Tecplot, select the surface(s) for which you

want to write data in the **Surfaces** list. If no surfaces are selected, the entire domain is exported.

3. For all file types except **ANSYS**, **FAST Solution**, and **NASTRAN**, select the variable(s) for which data is to be saved in the **Functions to Write** list.
4. (optional) For **Abaqus**, **ASCII**, **IDEAS Universal**, **NASTRAN**, and **PATRAN** files, select the **Loads** to be written (**Force**, **Temperature**, and/or **Heat Flux**). Saving these loads will allow you to analyze the structural stresses (fluid pressure or thermal) in an FEA program. Note that loads are written only on boundary walls when the entire domain is exported (i.e., if you select no **Surfaces**).
5. (optional) For **ASCII** files, select the **Delimiter** separating the fields (**Comma** or **Space**) and the **Location** from which the values of scalar functions are to be taken (**Node** values or **Cell-Centered** values).
6. Click on the **Write...** button to save a file for the specified function(s) in the specified format, using the **Select File** dialog box.

3.13.2 Export File Formats

Below, the files that are written when you select each export file type are listed:

Abaqus: a single file containing coordinates, connectivity, optional loads, zone groups, velocity, and selected scalars will be written. You can specify which scalars you want in the **Functions to Write** list. Export of data in solid zones to **Abaqus** is available for 3D models only.

ANSYS: a single file containing coordinates, connectivity, zone groups, velocity, pressure, temperature, and turbulence quantities will be written. The file written is an **ANSYS** results file with a **.rfl** extension. Note that export to **ANSYS** is available on a limited number of platforms (listed in the release notes).

ASCII: a single **ASCII** file containing the coordinate and connectivity information and specified scalar function data.

AVS: an AVS version 4 UCD file containing coordinate and connectivity information and specified scalar function data.

CGNS (CFD general notation system): a single file containing coordinates, connectivity, velocity, and selected scalars will be written. You can specify which scalars you want in the **Functions to Write** list.

Data Explorer: a single file containing coordinate, connectivity, velocity, and specified function data.

EnSight (formerly MPGS): a geometry file containing the coordinates and connectivity information, a velocity file containing the velocity, a scalar file for each selected variable or function, and a results file listing all the file names used.

FAST: a grid file in extended Plot3D format containing coordinates and connectivity, a velocity file containing the velocity, and a scalar file for each selected variable or function. This file type is valid only for triangular and tetrahedral grids.

FAST Solution: a single file containing density, velocity, and total energy. This file type is valid only for triangular and tetrahedral grids.

FieldView Unstructured: a single binary file containing coordinate and connectivity information and specified scalar function data. This file type is valid only for 3D grids.

FieldView Case+Data: a **FLUENT** case file that can be read by **FIELDVIEW**, and a data file containing node-averaged values for the selected variables.

FieldView Data: a data file containing node-averaged values for the selected variables. (For transient simulations, you will often export multiple **FIELDVIEW** data files but you will usually want to save the case file just once. In such cases, you can use the **FieldView Case+Data** option to save the first data set with a case file, and then use the **FieldView Data** option to save subsequent data sets without a case file.)

3.14 Grid-to-Grid Solution Interpolation

IDEAS Universal: a single file containing coordinates, connectivity, optional loads, zone groups, velocity, and selected scalars.

NASTRAN: a single file containing coordinates, connectivity, optional loads, zone groups, and velocity. Pressure is written as PLOAD, and heat flux is written as QHBDY data. If wall zones are selected in the Surfaces list, nodal forces are written for the walls.

PATRAN: a single file containing coordinates, connectivity, optional loads, zone groups, velocity, and selected scalars. Pressure is written as a distributed load. If wall zones are selected in the Surfaces list, nodal forces are written for the walls. The PATRAN result template file is written. This file lists the scalars present in the nodal result file (.rst).

Tecplot: a single file containing the coordinates and scalar functions in the appropriate tabular format.


3.14 Grid-to-Grid Solution Interpolation

FLUENT can interpolate solution data for a given geometry from one grid to another, allowing you to compute a solution using one grid (e.g., hexahedral) and then change to another grid (e.g., hybrid) and continue the calculation using the first solution as a starting point.

3.14.1 Performing Grid-to-Grid Solution Interpolation

The procedure for grid-to-grid solution interpolation is as follows:

1. Set up the model and calculate a solution on the initial grid.
2. Write an interpolation file for the solution data to be interpolated onto the new grid, using the Interpolate Data panel (Figure 3.14.1).

 → Interpolate...

- (a) Under Options, select Write Data.
- (b) In the Cell Zones list, select the cell zones for which you want to save data to be interpolated.

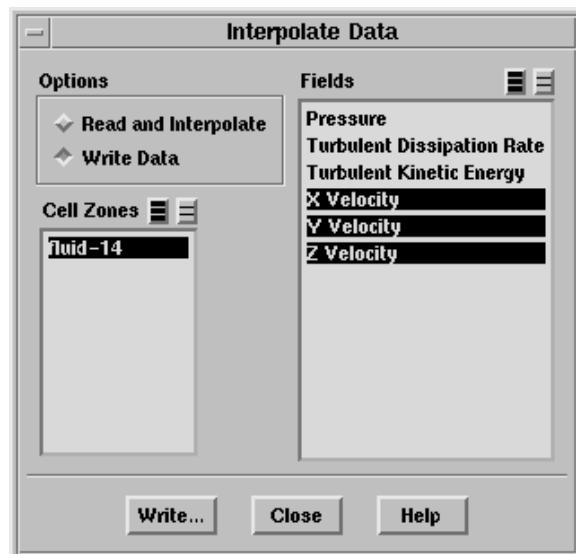


Figure 3.14.1: The Interpolate Data Panel

- !
- If your case includes both fluid and solid zones, you will need to write the data for the fluid zones and the data for the solid zones to separate files.
- (c) Choose the variable(s) for which you want to interpolate data in the Fields list. All FLUENT solution variables are available for interpolation.
 - (d) Click on the Write... button and specify the interpolation file name in the resulting Select File dialog box. The file format is described in Section 3.14.2.
3. If the new and original grids have the same zones, numbered in the same order, simply read in the new grid.

`file` → `reread-grid`

If the new and original grids have a different number of zones or their zones are numbered differently, you will need to set up a new case:

- (a) Read in the new grid, using the appropriate menu item in the File/Read/ or File/Import/ menu.
- (b) Define the appropriate models.

!

It is very important that you enable all of the models that were enabled in the original case. If, for example, the energy equation was enabled in the original case and you forget to enable it in the new case, the temperature data in the interpolation file will not be interpolated.

- (c) Define the boundary conditions, material properties, etc.

4. Read in the data to be interpolated.

File → Interpolate...

- (a) Under **Options**, select **Read and Interpolate**.
- (b) In the **Cell Zones** list, select the cell zones for which you want to read and interpolate data.

Note that, if the solution has not been initialized, computed, or read, all zones in the **Cell Zones** list will be selected by default, to ensure that no zone remains without data after the interpolation. If all zones already have data (from initialization or a previously computed or read solution), you can select a subset of the **Cell Zones** to read and interpolate data onto a specific zone (or zones).

- (c) Click on the **Read...** button and specify the interpolation file name in the resulting **Select File** dialog box.

!

If your case includes both fluid and solid zones, you will need to perform these steps twice (once to interpolate the data for the fluid zones and once to interpolate the data for the solid zones), since the two sets of data were saved to separate files.

5. Lower the under-relaxation factors and calculate on the new grid for a few iterations to avoid sudden changes due to any imbalance of fluxes after interpolation. Then increase the under-relaxation factors and compute a solution on the new grid.

3.14.2 Format of the Interpolation File

The format of the interpolation file is as follows:

- The first line is the interpolation file version. It is 1.0 for FLUENT 5 and 2.0 for FLUENT 6.
- The second line is the dimensionality (2 or 3).
- The third line is the total number of points.
- The fourth line is the total number of fields (temperature, pressure, etc.) included.
- A list of field names follows, from line 5. You can see a complete list of the field names used by FLUENT by selecting the `display/contours` text command and viewing the available choices for `contours of`. The list depends on the models turned on.
- After the list of field names, comes a list of x , y , and (in 3D) z coordinates for all the data points.
- All the field values at all the points in the same order as their names are listed last. The number of coordinate and field points should match the number given in line 3.

An example is shown below:

```
2
2
34800
3
x-velocity
pressure
y-velocity
-0.068062
-0.0680413
...
```

3.15 Reading Scheme Source Files

A Scheme source file can be loaded in three ways: through the menu system as a scheme file, through the menu system as a journal file, or through Scheme itself.

For large source files you will normally use the **Select File** dialog box (see Section 2.1.2) invoked by selecting the **File/Read/Scheme...** menu item

File → **Read** → Scheme...

or the Scheme load function.

```
> (load "file.scm")
```

Shorter files can also be loaded with the **File/Read/Journal...** menu item or the **file/read-journal** command in the text interface (or its **.** or **source** alias).

```
> . file.scm  
  
> source file.scm
```

In this case, each character of the file is echoed to the console as it is read in the same way as if you were typing the contents of the file in by hand.

3.16 The .fluent File

When starting up, FLUENT looks in your home directory for an optional file called **.fluent**. If it finds the file, it loads it with the Scheme load function. This file can contain Scheme functions that customize the code's operation.

3.17 Saving the Panel Layout

The Save Layout command in the File pull-down menu allows you to save the present panel and window layout. Specifically, you can arrange panels and graphics windows on your screen in a preferred configuration and select the File/Save Layout menu item.

File → Save Layout

A `.cxlayout` file is written in your home directory. (If you subsequently arrange different panels and save the layout again, the positions of these panels will be added to the positions of the panels that you saved earlier. If you move a panel for which a position is already saved, and then you save the layout, the new position will be written to the `.cxlayout` file.) In subsequent sessions, when you open a panel or create a graphics window, it will be positioned based on the saved configuration. Any panel or window not specified in the saved configuration will use the default position. Note that the `.cxlayout` file in your home directory applies to all Cortex applications (i.e., FLUENT, FLUENT/Post, MixSim, and TGrid).

3.18 Formats of Case and Data Files

This section describes the contents and formats of FLUENT case and data files. These files are broken into several sections according to the following guidelines:

- Each section is enclosed in parentheses and begins with a decimal integer indicating its type. This integer is different for formatted and binary files, as described in Section 3.18.1.
- All groups of items are enclosed in parentheses. This makes skipping to ends of (sub)sections and parsing them very easy. It also allows for easy and compatible addition of new items in future releases.
- Header information for lists of items is enclosed in separate sets of parentheses preceding the items, and the items are enclosed in their own parentheses.

Descriptions of the sections are grouped according to function. If you are creating grids for FLUENT, you only need to consider the sections described in Section 3.18.2. If you are attempting to import solutions into another postprocessor, you should study Sections 3.18.2 and 3.18.4. Section 3.18.3 describes the sections that store boundary conditions, material properties, and solver control settings. See Section 3.18.1 for details about differences between formatted and binary files.

Note that the case and data files may contain other sections that are intended for internal use only.

3.18.1 Formatting Conventions in Binary and Formatted Files

For formatted files, examples of file sections are given in Sections 3.18.2 and 3.18.3. For binary files, the header indices described in this section (e.g., 10 for the node section) are preceded by 20 for single-precision binary data, or by 30 for double-precision binary data (e.g., 2010 or 3010 instead of 10). Also, the end of the binary data is indicated by **End of Binary Section 2010** or **End of Binary Section 3010** before the

closing parameters of the section. An example is shown below (with the binary data represented by periods):

```
(2010 (2 1 2aad 2 3)(  
.  
.  
.  
)  
End of Binary Section 2010)
```

3.18.2 Grid Sections

Grid sections are stored in the case file. (A “grid” file is a subset of a case file, containing only those sections pertaining to the grid.) Following are descriptions of the currently defined grid sections.

The section ID numbers are indicated below in both symbolic and numeric forms. The symbolic representations are available as symbols in a Scheme source file (`xfile.scm`), which is available from Fluent Inc., or as macros in a C header file (`xfile.h`), which is located in the following directory in your installation area:

```
/Fluent.Inc/fluent6.x/client/src/xfile.h
```

Comment

Index:	0
Scheme symbol:	<code>xf-comment</code>
C macro:	<code>XF_COMMENT</code>
Status:	optional

Comment sections can appear anywhere in the file (except within other sections), and appear as

```
(0 "comment text")
```

It is recommended that each long section, or group of related sections, be preceded by a comment section explaining what is to follow. For example,


```
(0 "Variables:")  
(37 (  
  (relax-mass-flow 1)  
  (default-coefficient ())  
  (default-method 0)  
))
```

Header

Index:	1
Scheme symbol:	xf-header
C macro:	XF_HEADER
Status:	optional

Header sections can appear anywhere in the file (except within other sections), and appear as

```
(1 "TGrid 2.1.1")
```

The purpose of this section is to identify the program that wrote the file. Although it can appear anywhere, it is typically one of the first sections in the file. Additional header sections indicate other programs that may have been used in generating the file and thus provide a history mechanism showing where the file came from and how it was processed.

Dimensions

Index:	2
Scheme symbol:	xf-dimension
C macro:	XF_DIMENSION
Status:	optional

The dimensionality of the grid

```
(2 ND)
```

where ND is 2 or 3. This section is currently supported as a check that the grid has the appropriate dimensionality.

Nodes

Index:	10
Scheme symbol:	xf-node
C macro:	XF_NODE
Status:	required

```
(10 (zone-id first-index last-index type ND)(  
  x1 y1 z1  
  x2 y2 z2  
  .  
  .  
  .  
))
```

If **zone-id** is zero, this is a declaration section providing the total number of nodes in the grid. **first-index** will then be one, **last-index** will be the total number of nodes *in hexadecimal*, **type** is equal to 1, **ND** is the dimensionality of the grid, and there are no coordinates following. The parentheses for the coordinates are not there either. For example,

```
(10 (0 1 2d5 1 2))
```

If **zone-id** is greater than zero, it indicates the zone to which the nodes belong. **first-index** and **last-index** are then the indices of the nodes in the zone, *in hexadecimal*. Of course, **last-index** in each zone must be less than or equal to the value in the declaration section. Type is always equal to 1.

ND is an optional argument that indicates the dimensionality of the node data, where **ND** is 2 or 3.

If the number of dimensions in the grid is two, as specified by the Dimensions section described above or in the node header, then only *x* and *y* coordinates are present on each line.

Below is a two-dimensional example:

```
(10 (1 1 2d5 1 2)(
1.500000e-01 2.500000e-02
1.625000e-01 1.250000e-02
.
.
.
1.750000e-01 0.000000e+00
2.000000e-01 2.500000e-02
1.875000e-01 1.250000e-02
))
```

Because the grid connectivity is composed of integers representing pointers (see Cells and Faces, below), using hexadecimal conserves space in the file and provides for faster file input and output. The header indices are also in hexadecimal so that they match the indices in the bodies of the grid connectivity sections. The **zone-id** and **type** are also in hexadecimal for consistency.

Periodic Shadow Faces

Index:	18
Scheme symbol:	xf-periodic-face
C macro:	XF_PERIODIC_FACE
Status:	required only for grids with periodic boundaries

This section indicates the pairings of periodic faces on periodic boundaries. Grids without periodic boundaries do not have sections of this type. The format of the section is as follows:

```
(18 (first-index last-index periodic-zone shadow-zone)(
f00 f01
f10 f21
f20 f21
.
.
.
))
```

where **first-index** is the index of the first periodic face pair in the list, **last-index** is the last one, **periodic-zone** is the zone ID of the periodic face zone, and **shadow-zone** is the zone ID of the corresponding shadow face zone. These are in hexadecimal format.

The indices in the section body (**f***) refer to the faces on each of the periodic boundaries (in hexadecimal), the indices being offsets into the list of faces for the grid. Note that **first-index** and **last-index** do *not* refer to face indices; they refer to indices in the list of periodic pairs.

A partial example of such a section follows.

```
(18 (1 2b a c) (  
12 1f  
13 21  
ad 1c2  
.  
.  
.  
))
```

Cells

Index:	12
Scheme symbol:	xf-cell
C macro:	XF_CELL
Status:	required

The declaration section for cells is similar to that for nodes.

```
(12 (zone-id first-index last-index type element-type))
```

Again, **zone-id** is zero to indicate that it is a declaration of the total number of cells. If **last-index** is zero, then there are no cells in the grid. This is useful when the file contains only a surface mesh to alert FLUENT that it cannot be used. In a declaration section, the **type** is ignored, and is usually given as zero, while the **element-type** is ignored completely. For example,

```
(12 (0 1 3e3 0))
```

states that there are 3e3 (hexadecimal) = 995 cells in the grid. This declaration section is required and must precede the regular cell sections.

The **element-type** in a regular cell section header indicates the type of cells in the section, as follows:

element-type	<i>description</i>	<i>nodes/cell</i>	<i>faces/cell</i>
0	mixed		
1	triangular	3	3
2	tetrahedral	4	4
3	quadrilateral	4	4
4	hexahedral	8	6
5	pyramid	5	5
6	wedge	6	5

Regular cell sections have no body, but they have a header of the same format where **first-index** and **last-index** indicate the range for the particular zone, **type** indicates whether the cell zone is an active zone (solid or fluid), or inactive zone (currently only parent cells resulting from hanging node adaption). Active zones are represented with **type=1**, while inactive zones are represented with **type=32**.

Note that in past versions of FLUENT, a distinction was made used between solid and fluid zones. This is now determined by properties (i.e., material type).

A **type** of zero indicates a dead zone and will be skipped by FLUENT. If a zone is of mixed type (**element-type=0**), it will have a body that lists the **element type** of each cell. For example,

```
(12 (9 1 3d 0 0)(
1 1 1 3 3 1 1 3 1
.
.
.
))
```

states that there are 3d (hexadecimal) = 61 cells in cell zone 9, of which the first 3 are triangles, the next 2 are quadrilaterals, and so on.

Faces

Index:	13
Scheme symbol:	xf-face
C macro:	XF_FACE
Status:	required

The face section has a header with the same format as that for cells (but with a section index of 13).

```
(13 (zone-id first-index last-index type element-type))
```

Again, a **zone-id** of zero indicates a declaration section with no body, and **element-type** indicates the type of faces in that zone.

The body of a regular face section contains the grid connectivity, and each line appears as follows:

```
n0 n1 n2 cr cl
```

where **n*** are the defining nodes (vertices) of the face, and **c*** are the adjacent cells. This is an example of the triangular face format; the actual number of nodes depends on the **element type**. The ordering of the cell indices is important. The first cell, **cr**, is the cell on the right side of the face and **cl** is the cell on the left side. Handedness is determined by the right-hand rule: if you curl the fingers of your right hand in the order of the nodes, your thumb will point to the right side of the face. In 2D grids, the \hat{k} vector pointing outside the grid plane is used to determine the right-hand-side cell (**cr**) from $\hat{k} \times \hat{r}$. If there is no adjacent cell, then either **cr** or **cl** is zero. (All cells, faces, and nodes have positive indices.) For files containing only a boundary mesh, both these values are zero. If it is a two-dimensional grid, **n2** is omitted.

If the face zone is of mixed type (**element-type** = 0), the body of the section will include the face type and will appear as follows:

`type v0 v1 v2 c0 c1`

where `type` is the type of face, as defined in the following table:

<code>element-type</code>	<i>face type</i>	<i>nodes/face</i>
0	mixed	
2	linear	2
3	triangular	3
4	quadrilateral	4

The current valid boundary condition types are defined in the following table:

<code>bc name</code>	<i>bc id</i>
interior	2
wall	3
pressure-inlet, inlet-vent, intake-fan	4
pressure-outlet, exhaust-fan, outlet-vent	5
symmetry	7
periodic-shadow	8
pressure-far-field	9
velocity-inlet	10
periodic	12
fan, porous-jump, radiator	14
mass-flow-inlet	20
interface	24
parent (hanging node)	31
outflow	36
axis	37

For non-conformal grid interfaces, the faces resulting from the intersection of the non-conformal grids are placed in a separate face zone. A factor of 1000 is added to the `type` of these sections, e.g., 1003 is a wall zone.

Face Tree

Index: 59
Scheme symbol: `xf-face-tree`
C macro: `XF_FACE_TREE`
Status: only for grids with hanging-node adaption

This section indicates the face hierarchy of the grid containing hanging nodes. The format of the section is as follows:

```
(59 (face-id0 face-id1 parent-zone-id child-zone-id)
(
  number-of-kids kid-id-0 kid-id-1 ... kid-id-n
  .
  .
  .
))
```

where `face-id0` is the index of the first parent face in the section, `face-id1` is the index of the last parent face in the section, `parent-zone-id` is the ID of the zone containing the parent faces, `child-zone-id` is the ID of the zone containing the children faces, `number-of-kids` is the number of children of the parent face, and `kid-id-n` are the face IDs of the children. These are in hexadecimal format.

Cell Tree

Index: 58
Scheme symbol: `xf-cell-tree`
C macro: `XF_CELL_TREE`
Status: only for grids with hanging-node adaption

This section indicates the cell hierarchy of the grid containing hanging nodes. The format of the section is as follows:

```
(58 (cell-id0 cell-id1 parent-zone-id child-zone-id)
(
  number-of-kids kid-id-0 kid-id-1 ... kid-id-n
```



```
.
.
.
))
```

where **cell-id0** is the index of the first parent cell in the section, **cell-id1** is the index of the last parent cell in the section, **parent-zone-id** is the ID of the zone containing the parent cells, **child-zone-id** is the ID of the zone containing the children cells, **number-of-kids** is the number of children of the parent cell, and **kid-id-n** are the cell IDs of the children. These are in hexadecimal format.

Interface Face Parents

Index: 61
Scheme symbol: **xf-face-parents**
C macro: **XF_FACE_PARENTS**
Status: only for grids with non-conformal interfaces

This section indicates the relationship between the intersection faces and original faces. The intersection faces (children) are produced from intersecting two non-conformal surfaces (parents) and are some fraction of the original face. Each child will refer to at least one parent.

The format of the section is as follows:

```
(61 (face-id0 face-id1)
(
parent-id-0 parent-id-1
.
.
.
))
```

where **face-id0** is the index of the first child face in the section, **face-id1** is the index of the last child face in the section, **parent-id-0** is the index of the right-side parent face, and **parent-id-1** is the index of the left-side parent face. These are in hexadecimal format.

If you read a non-conformal grid from FLUENT into TGrid, TGrid will skip this section, so it will not maintain all the information necessary to preserve the non-conformal interface. When you read the grid back into FLUENT, you will need to recreate the interface.

Example Files

Example 1

Figure 3.18.1 illustrates a simple quadrilateral mesh with no periodic boundaries or hanging nodes.

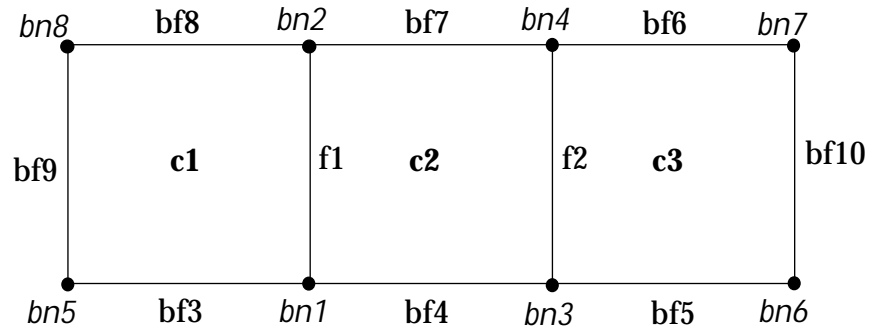


Figure 3.18.1: Quadrilateral Mesh

The following describes this mesh:

```
(0 "Grid:")

(0 "Dimensions:")
(2 2)

(12 (0 1 3 0))
(13 (0 1 a 0))
(10 (0 1 8 0 2))

(12 (7 1 3 1 3))
```

3.18 Formats of Case and Data Files

```
(13 (2 1 2 2 2)(  
1 2 1 2  
3 4 2 3))
```

```
(13 (3 3 5 3 2)(  
5 1 1 0  
1 3 2 0  
3 6 3 0))
```

```
(13 (4 6 8 3 2)(  
7 4 3 0  
4 2 2 0  
2 8 1 0))
```

```
(13 (5 9 9 a 2)(  
8 5 1 0))
```

```
(13 (6 a a 24 2)(  
6 7 3 0))
```

```
(10 (1 1 8 1 2)  
(  
1.00000000e+00 0.00000000e+00  
1.00000000e+00 1.00000000e+00  
2.00000000e+00 0.00000000e+00  
2.00000000e+00 1.00000000e+00  
0.00000000e+00 0.00000000e+00  
3.00000000e+00 0.00000000e+00  
3.00000000e+00 1.00000000e+00  
0.00000000e+00 1.00000000e+00))
```

Example 2

Figure 3.18.2 illustrates a simple quadrilateral mesh with periodic boundaries but no hanging nodes. In this example, bf9 and bf10 are faces on the periodic zones.

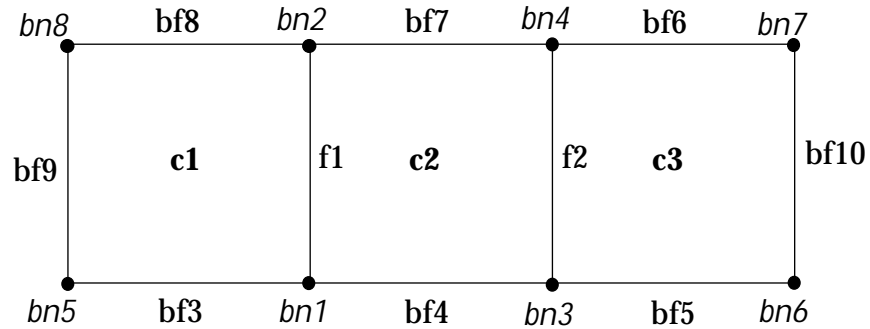


Figure 3.18.2: Quadrilateral Mesh with Periodic Boundaries

The following describes this mesh:

```
(0 "Dimensions:")
(2 2)

(0 "Grid:")

(12 (0 1 3 0))
(13 (0 1 a 0))
(10 (0 1 8 0 2))

(12 (7 1 3 1 3))

(13 (2 1 2 2 2) (
1 2 1 2
3 4 2 3))

(13 (3 3 5 3 2) (
```

3.18 Formats of Case and Data Files

```
5 1 1 0
1 3 2 0
3 6 3 0))
```

```
(13 (4 6 8 3 2)(
7 4 3 0
4 2 2 0
2 8 1 0))
```

```
(13 (5 9 9 c 2)(
8 5 1 0))
```

```
(13 (1 a a 8 2)(
6 7 3 0))
```

```
(18 (1 1 5 1)(
9 a))
```

```
(10 (1 1 8 1 2)(
1.00000000e+00 0.00000000e+00
1.00000000e+00 1.00000000e+00
2.00000000e+00 0.00000000e+00
2.00000000e+00 1.00000000e+00
0.00000000e+00 0.00000000e+00
3.00000000e+00 0.00000000e+00
3.00000000e+00 1.00000000e+00
0.00000000e+00 1.00000000e+00))
```

Example 3

Figure 3.18.3 illustrates a simple quadrilateral mesh with hanging nodes.

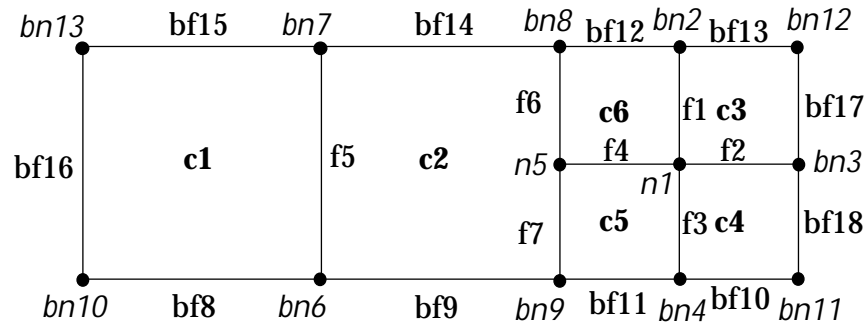


Figure 3.18.3: Quadrilateral Mesh with Hanging Nodes

The following describes this mesh:

```
(0 "Grid:")

(0 "Dimensions:")
(2 2)

(12 (0 1 7 0))
(13 (0 1 16 0))
(10 (0 1 d 0 2))

(12 (7 1 6 1 3))
(12 (1 7 7 20 3))

(58 (7 7 1 7) (
  4 6 5 4 3))

(13 (2 1 7 2 2) (
  1 2 6 3
  1 3 3 4
```

3.18 Formats of Case and Data Files

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

```
(13 (3 8 b 3 2)(
a 6 1 0
6 9 2 0
4 b 4 0
9 4 5 0))
```

```
(13 (4 c f 3 2)(
2 8 6 0
c 2 3 0
8 7 2 0
7 d 1 0))
```

```
(13 (5 10 10 a 2)(
d a 1 0))
```

```
(13 (6 11 12 24 2)(
3 c 3 0
b 3 4 0))
```

```
(13 (b 13 13 1f 2)(
c 8 7 0))
```

```
(13 (a 14 14 1f 2)(
b c 7 0))
```

```
(13 (9 15 15 1f 2)(
9 b 7 0))
```

```
(13 (8 16 16 1f 2)(
9 8 2 7))
```

```
(59 (13 13 b 4) (
  2 d c))
```

```
(59 (14 14 a 6) (
  2 12 11))
```

```
(59 (15 15 9 3) (
  2 b a))
```

```
(59 (16 16 8 2) (
  2 7 6))
```

```
(10 (1 1 d 1 2)
(
  2.50000000e+00 5.00000000e-01
  2.50000000e+00 1.00000000e+00
  3.00000000e+00 5.00000000e-01
  2.50000000e+00 0.00000000e+00
  2.00000000e+00 5.00000000e-01
  1.00000000e+00 0.00000000e+00
  1.00000000e+00 1.00000000e+00
  2.00000000e+00 1.00000000e+00
  2.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00
  3.00000000e+00 0.00000000e+00
  3.00000000e+00 1.00000000e+00
  0.00000000e+00 1.00000000e+00))
```


3.18.3 Other (Non-Grid) Case Sections

The following sections store boundary conditions, material properties, and solver control settings.

Zone

Index:	39 (45)
Scheme symbol:	xf-rp-tv
C macro:	XF_RP_TV
Status:	required

There is typically one zone section for each zone referenced by the grid. Although some grid zones may not have corresponding zone sections, there cannot be more than one zone section for each zone.

A zone section has the following form:

```
(39 (zone-id zone-type zone-name)(  
  (condition1 . value1)  
  (condition2 . value2)  
  (condition3 . value3)  
  .  
  .  
  .  
))
```

Grid generators and other preprocessors need only provide the section header and leave the list of conditions empty, as in

```
(39 (zone-id zone-type zone-name)())
```

The empty parentheses at the end are required. The solver adds conditions as appropriate, depending on the zone type. When only **zone-id**, **zone-type**, and **zone-name** are specified, the index 45 is preferred for a zone section. However, the index 39 must be used if boundary conditions are present, because any and all remaining information in a section of index 45 after **zone-id**, **zone-type**, and **zone-name** will be ignored.

Here the `zone-id` is in *decimal* format. This is in contrast to the use of hexadecimal in the grid sections.

The `zone-type` is one of the following:

```
axis
exhaust fan
fan
fluid
inlet vent
intake fan
interface
interior
mass-flow-inlet
outlet vent
outflow
periodic
porous-jump
pressure-far-field
pressure-inlet
pressure-outlet
radiator
shadow
solid
symmetry
velocity-inlet
wall
```

The `interior`, `fan`, `porous-jump`, and `radiator` types can be assigned only to zones of faces inside the domain. The `interior` type is used for the faces within a cell zone; the others are for interior faces that form infinitely thin surfaces within the domain. **FLUENT** allows the `wall` type to be assigned to face zones both on the inside and on the boundaries of the domain. Some zone types are valid only for certain types of grid components. For example, cell (element) zones can be assigned only one of the following types:

fluid
solid

All of the other types listed above can be used only for boundary (face) zones.

The **zone-name** is a user-specified label for the zone. It must be a valid Scheme symbol¹ and is written without quotes. The rules for a valid **zone-name** (Scheme symbol) are as follows:

- The first character must be a lowercase letter² or a special-initial.
- Each subsequent character must be a lowercase letter, a special-initial, a digit, or a special-subsequent.

where a special-initial character is one of the following:

! \$ % & * / : < = > ? ~ _ ^

and a special-subsequent is one of the following:

. + -

Examples of valid zone names are **inlet-port/cold!**, **eggs/easy**, and **e=m*c^2**.

Below are some examples of zone sections produced by grid generators and preprocessors.

```
(39 (1 fluid fuel)())
```

```
(39 (8 pressure-inlet pressure-inlet-8)())
```

```
(39 (2 wall wing-skin)())
```

```
(39 (3 symmetry mid-plane)())
```

¹See Revised⁽⁴⁾ Report on the Algorithmic Language Scheme, William Clinger and Jonathan Rees (Editors), 2 November 1991, Section 7.1.1.

²The Standard actually only requires that case be insignificant; the Fluent Inc. implementation accomplishes this by converting all uppercase input to lowercase.

Partitions

Index: 40
Scheme symbol: `xf-partition`
C macro: `XF_PARTITION`
Status: only for partitioned grids

This section indicates each cell's partition. The format of the section is as follows:

```
(40 (zone-id first-index last-index partition-count)(  
p1  
p2  
p3  
.  
.  
.  
pn  
))
```

`p1` is the partition of the cell whose ID is `first-index`, `p2` is the partition of the cell whose ID is `first-index+1`, etc., and `pn` is the partition of the cell whose ID is `last-index`. Partition IDs must be between 0 and `partition-count-1`, where `partition-count` is the total number of partitions.

3.18.4 Data Sections

The following sections store iterations, residuals, and data field values.

Grid Size

Index: 33
Scheme symbol: `xf-grid-size`
C macro: `XF_GRID_SIZE`
Status: optional

This section indicates the number of cells, faces, and nodes in the grid that corresponds to the data in the file. This information is used to check that the data and grid match. The format is

```
(33 (n-elements n-faces n-nodes))
```

where the integers are written in decimal.

Data Field

Index: 300
Scheme symbol: `xf-rf-seg-data`
C macro: `XF_RF_SEG_DATA`
Status: required

This section lists a flow field solution variable for a cell or face zone. The data are stored in the same order as the cells or faces in the case file. Separate sections are written out for each variable for each face or cell zone on which the variable is stored. The format is

```
(300 (sub-section-id zone-id size n-time-levels  
      n-phases first-id last-id)  
( data for cell or face with id = first-id  
  data-for-cell-or-face with id = first-id+1  
  ..  
  data-for-cell-or-face with id = last-id  
)
```

where `sub-section-id` is a (decimal) integer that identifies the variable field (e.g., 1 for pressure, 2 for velocity). The complete list of these is available in the header file (`xfile.h`), which is located in the following directory in your installation area:

```
/Fluent.Inc/fluent6.x/client/src/xfile.h
```

`zone-id` is the ID number of the cell or face zone and matches the ID used in case file. `size` denotes the length of the variable vector (1 for a scalar, 2 or 3 for a vector, equal to the number of species for variables defined for each species). `n-time-levels` and `n-phases` currently are not used.

A sample data file section for the velocity field in a cell zone for a steady-state, single-phase, 2D problem is shown below:

```
(300 (2 16 2 0 0 17 100)
(8.08462024e-01 8.11823010e-02
8.78750622e-01 3.15509699e-02
1.06139672e+00 -3.74040119e-02
...
1.33301604e+00 -5.04243895e-02
6.21703446e-01 -2.46118382e-02
4.41687912e-01 -1.27046436e-01
1.03528820e-01 -1.01711005e-01
))
```

The variables that are listed in the data file depend on the models active at the time the file is written. Variables that are required by the solver based on the current model settings but are missing from the data file are set to their default values when the data file is read. Any extra variables that are present in the data file but are not relevant according to current model settings are ignored.

Residuals

Index:	301
Scheme symbol:	xf-rf-seg-residuals
C macro:	XF_RF_SEG_RESIDUALS
Status:	optional

This section lists the values of the residuals for a particular data field variable at each iteration:

```
(301 (n residual-subsection-id size)(
r1
r2
.
.
.
rn
))
```

where **n** is the number of residuals and **size** is the length of the variable vector (1 for a scalar, 2 or 3 for a vector, equal to the number of species

3.18 Formats of Case and Data Files

for variables defined for each species). The **residual-subsection-id** is an integer (decimal) indicating the equation for which the residual is stored in the section, according to the C constants defined in a header file (**xfile.h**) available in your installation area, as noted in Section 3.18.2.

The equations for which residuals are listed in the data file depend on the models active at the time the file is written. If the residual history is missing from the data file for a currently active equation, it is initialized with zeros.

