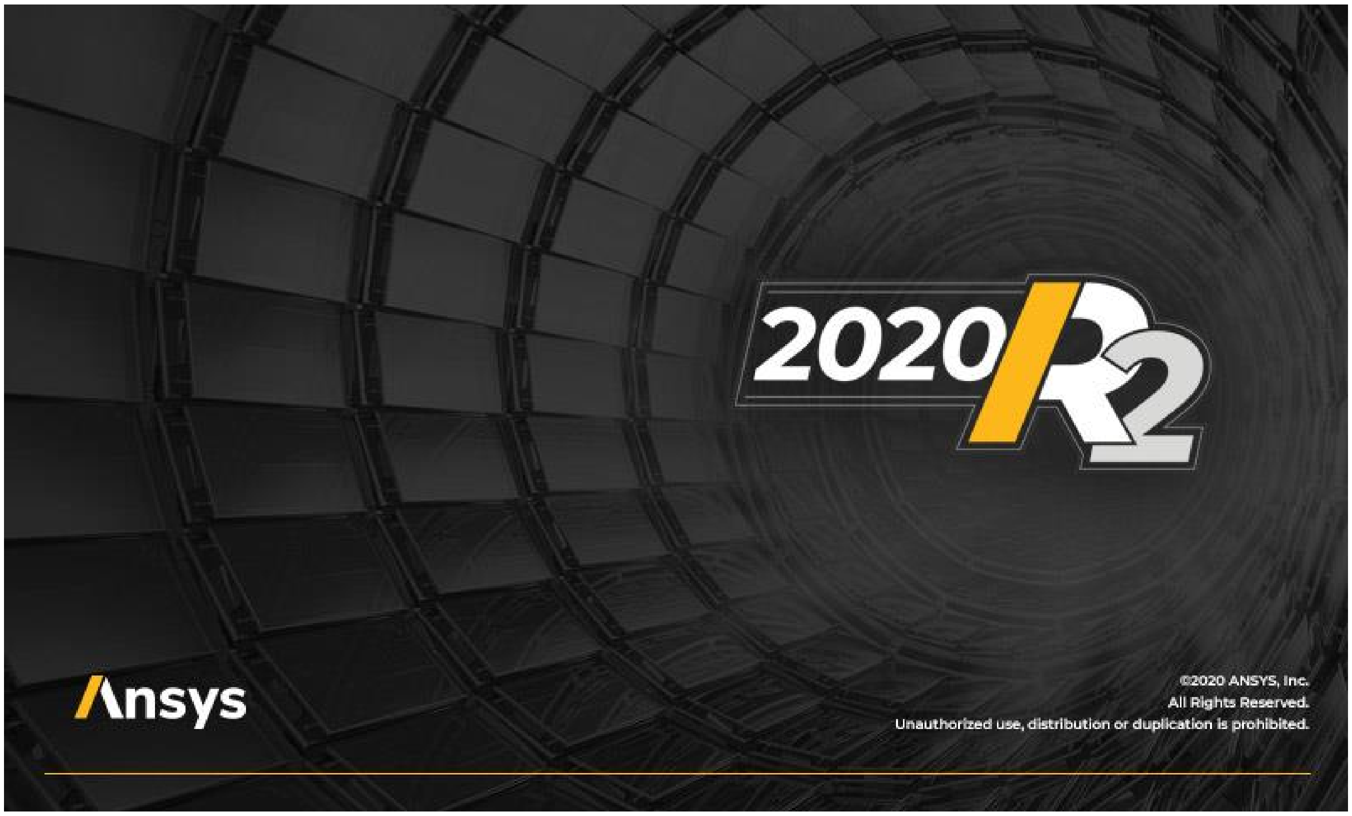

2020 R2
Ansys
©2020 ANSYS, Inc.
All Rights Reserved.
Unauthorized use, distribution or duplication is prohibited.

ANSYS Fluent as a Server User's Guide

ANSYS, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Release 2020 R2
July 2020

Table of Contents

List of Figures

List of Tables

Using This Manual

This preface is divided into the following sections:

1. The Contents of This Manual

The ANSYS Fluent As A Server User's Guide documents the ANSYS Fluent As A Server capability which allows remote connection to, and control of, Fluent sessions. In this manual you will find an overview of the capability, a description of how to set up and use Fluent as a Server, and examples.

2. Typographical Conventions

Several typographical conventions are used in this manual's text to help you find commands in the user interface.

- Different type styles are used to indicate graphical user interface items and text interface items. For example:

  Iso-Surface dialog box

  `surface/iso-surface` text command

- The text interface type style is also used when illustrating exactly what appears on the screen to distinguish it from the narrative text. In this context, user inputs are typically shown in boldface. For example,

```
solve/initialize/set-fmg-initialization

 Customize your FMG initialization:
   set the number of multigrid levels [5]

   set FMG parameters on levels ..

    residual reduction on level 1 is:  [0.001]
    number of cycles on level 1 is:  [10]  100

    residual reduction on level 2 is:  [0.001]
    number of cycles on level 2 is:  [50]  100
```

- Mini flow charts are used to guide you through the ribbon or the tree, leading you to a specific option, dialog box, or task page. The following tables list the meaning of each symbol in the mini flow charts.

Table 1: Mini Flow Chart Symbol Descriptions

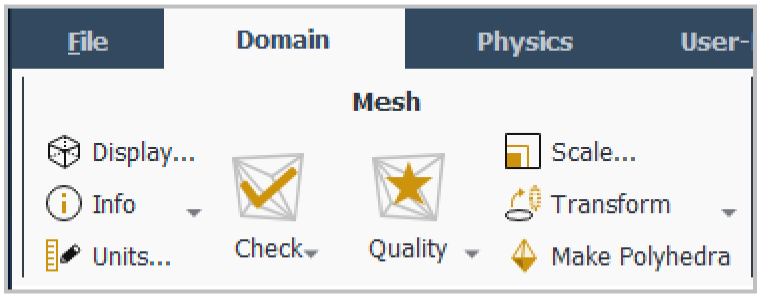
Symbol	Indicated Action
▭	Look at the ribbon
☰	Look at the tree

✥	Double-click to open task page
☰	Select from task page
🖱	Right-click the preceding item

For example,

Setting Up Domain → **Mesh** → **Transform** → **Translate...**

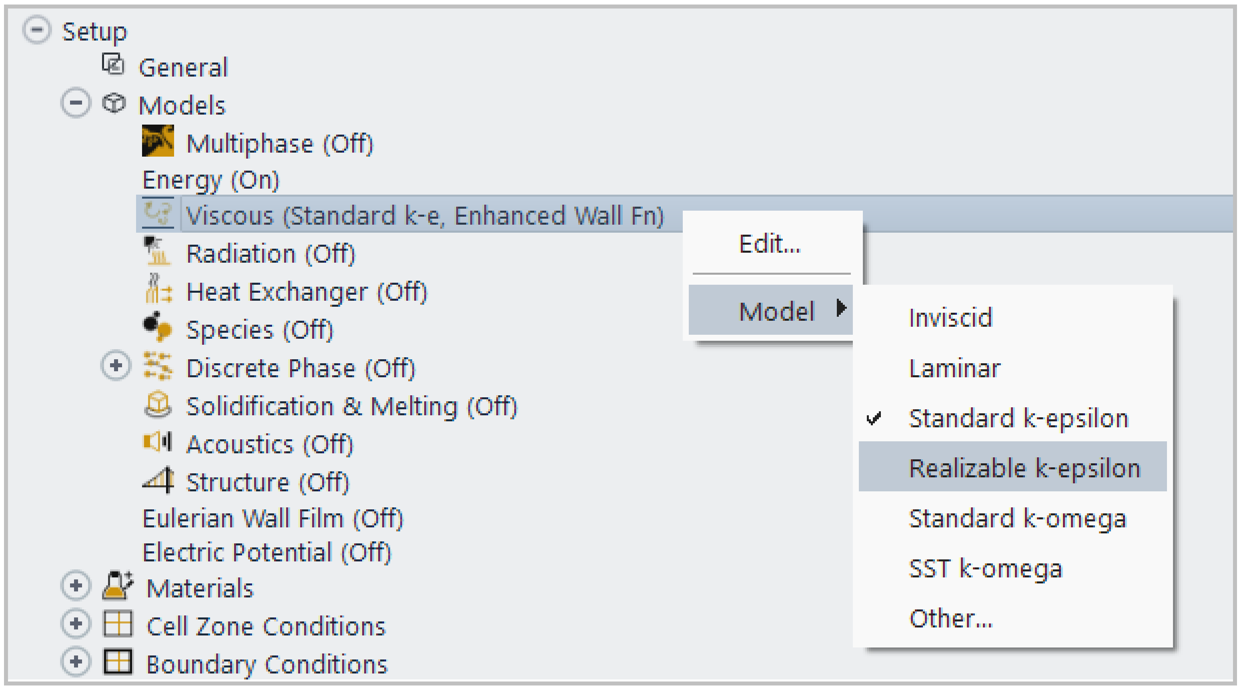
indicates selecting the **Setting Up Domain** ribbon tab, clicking **Transform** (in the **Mesh** group box) and selecting **Translate...**, as indicated in the figure below:



And

Setup → **Models** → **Viscous** 🖱 **Model** → **Realizable k-epsilon**

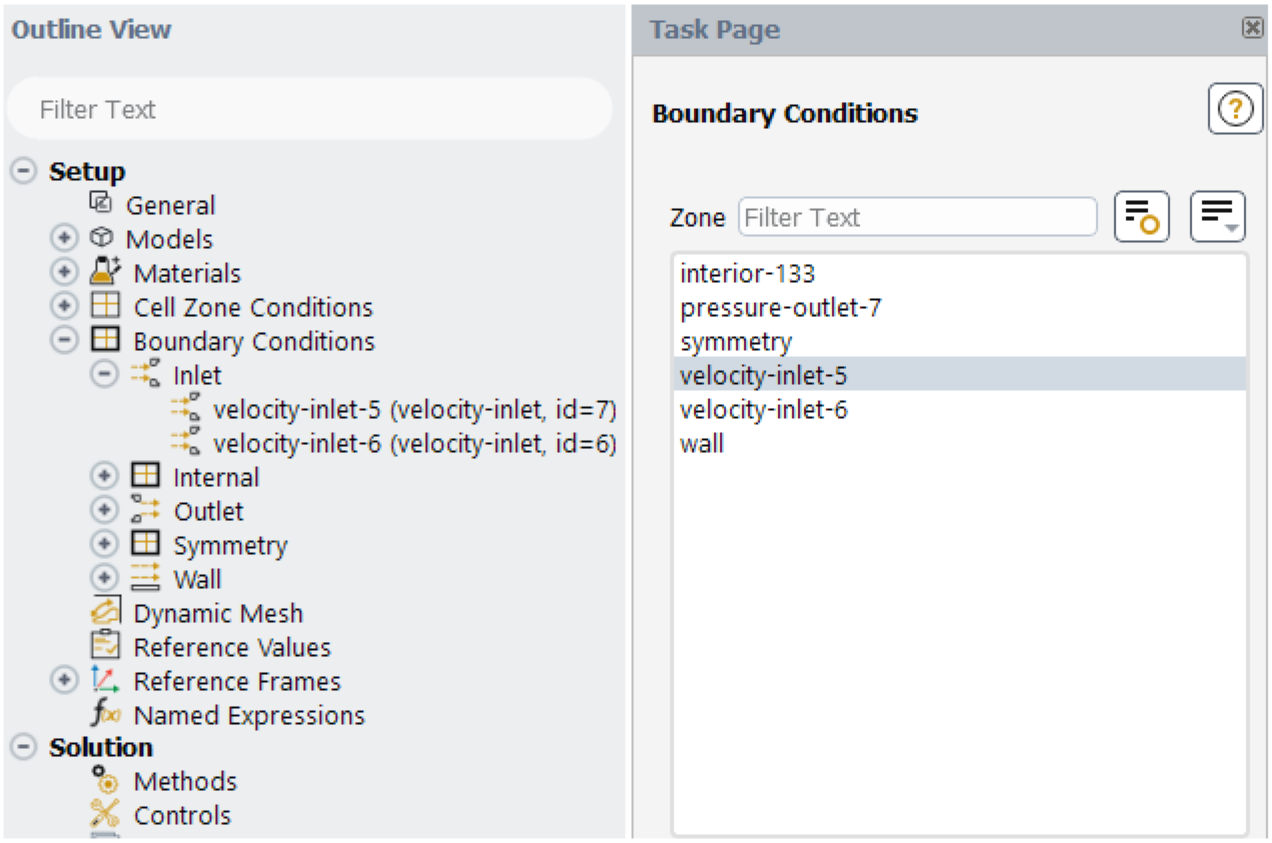
indicates expanding the **Setup** and **Models** branches, right-clicking **Viscous**, and selecting **Realizable k-epsilon** from the **Model** sub-menu, as shown in the following figure:



And

Setup → **Boundary Conditions** → **velocity-inlet-5**

indicates opening the task page as shown below:



In this manual, mini flow charts usually accompany a description of a dialog box or command, or a screen illustration showing how to use the dialog box or command. They show you how to quickly access a command or dialog box without having to search the surrounding material.

- In-text references to **File** ribbon tab selections can be indicated using a "/". For example **File/Write/Case...** indicates clicking the **File** ribbon tab and selecting **Case...** from the **Write** submenu (which opens the **Select File** dialog box).

3. Mathematical Conventions

- Where possible, vector quantities are displayed with a raised arrow (for example, $\vec{a}$, $\vec{A}$). Boldfaced characters are reserved for vectors and matrices as they apply to linear algebra (for example, the identity matrix, $\mathbf{I}$).

- The operator ∇, referred to as grad, nabla, or del, represents the partial derivative of a quantity with respect to all directions in the chosen coordinate system. In Cartesian coordinates, ∇ is defined to be

$$\frac{\partial}{\partial x}\vec{\imath}+\frac{\partial}{\partial y}\vec{\jmath}+\frac{\partial}{\partial z}\vec{k} \tag{1}$$

  ∇ appears in several ways:

– The gradient of a scalar quantity is the vector whose components are the partial derivatives; for example,

$$\nabla p = \frac{\partial p}{\partial x}\vec{i} + \frac{\partial p}{\partial y}\vec{j} + \frac{\partial p}{\partial z}\vec{k} \tag{2}$$

– The gradient of a vector quantity is a second-order tensor; for example, in Cartesian coordinates,

$$\nabla\left(\vec{v}\right) = \left(\frac{\partial}{\partial x}\vec{i} + \frac{\partial}{\partial y}\vec{j} + \frac{\partial}{\partial z}\vec{k}\right)\left(v_x\vec{i} + v_j\vec{j} + v_z\vec{k}\right) \tag{3}$$

This tensor is usually written as

$$\begin{pmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} & \frac{\partial v_x}{\partial z} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} & \frac{\partial v_y}{\partial z} \\ \frac{\partial v_z}{\partial x} & \frac{\partial v_z}{\partial y} & \frac{\partial v_z}{\partial z} \end{pmatrix} \tag{4}$$

– The divergence of a vector quantity, which is the inner product between ∇ and a vector; for example,

$$\nabla\cdot\vec{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \tag{5}$$

– The operator $\nabla\cdot\nabla$, which is usually written as ∇^2 and is known as the Laplacian; for example,

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \tag{6}$$

$\nabla^2 T$ is different from the expression $\left(\nabla T\right)^2$, which is defined as

$$\left(\nabla T\right)^2 = \left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial y}\right)^2 + \left(\frac{\partial T}{\partial z}\right)^2 \tag{7}$$

- An exception to the use of ∇ is found in the discussion of Reynolds stresses in Turbulence in the *Fluent Theory Guide*, where convention dictates the use of Cartesian tensor notation. In this chapter, you will also find that some velocity vector components are written as u, v, and w instead of the conventional v with directional subscripts.

Chapter 1: Introduction

ANSYS Fluent As A Server is a set of tools and functionality that allows local or remote client applications to access the full power of the ANSYS Fluent solver. A client application can perform case setup, initialization, iteration, and result reporting using the Fluent as a Server interface. Note that this capability is different from batch mode operation in that commands can be issued to the running session at any time rather than just from a predefined journal file. This allows solution steering and other manipulations without exiting from the Fluent session. The client application can be either:

- The Fluent Remote Console, Fluent Remote Console (p. 23)
- A custom application built using the Fluent as a Server Software Development Kit (SDK), Fluent as a Server Software Development Kit (SDK) (p. 49)

Following are some examples of how Fluent as a Server functionality might be used to run Fluent remotely and provide information of interest from a simulation:

Sample Use Cases

- A user performing a Fluent simulation on a local workstation requires assistance with a simulation from a Subject Matter Expert (SME) at another location. The user's Fluent session is running with the Fluent as a Server functionality enabled. The SME uses Fluent Remote Console to connect to the user's interactive session and use TUI commands to examine results, change under-relaxation or model settings, and assist the user in obtaining a solution.
- A custom application for design and layout of aircraft passenger seating configuration needs to accurately predict cooling air velocity around occupants. By incorporating interface methods provided in the Fluent as a Server SDK, the custom application can connect to Fluent as a Server and run a simulation to provide contour plots of air velocity.
- An engineer uses process modeling software to predict the output of an entire facility. A schematic of the overall process is built by linking individual pieces of equipment such as filter presses, reactors, or flash tanks. Standard engineering calculations are performed for each unit operation. However, problems arise when a tank reactor cannot be characterized using common assumptions such as "fully mixed". Using the Fluent as a Server SDK, the process modeling software can be connected to Fluent as a Server which performs a highly accurate CFD solution in place of the less accurate engineering calculation.

1.1. Overview

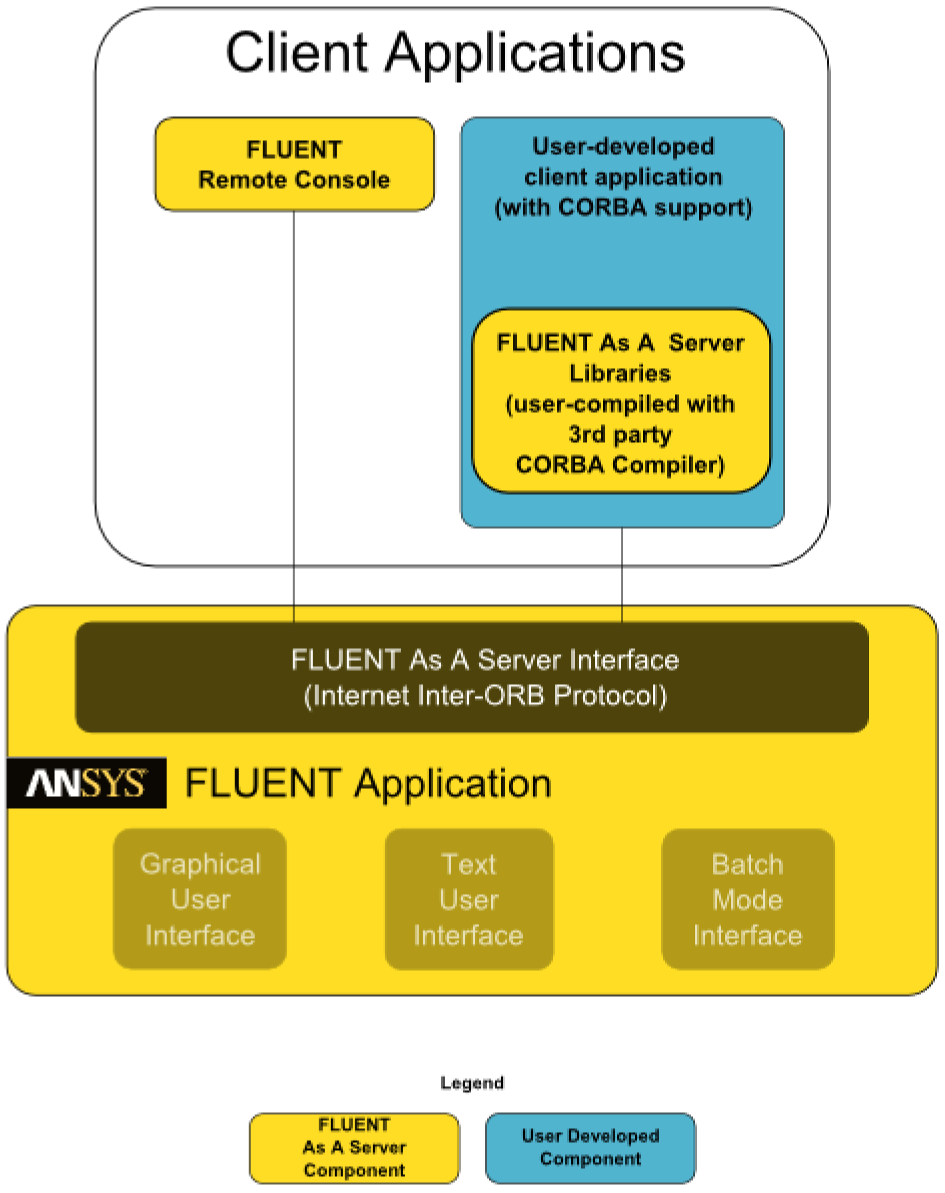
The following elements, provided with ANSYS Fluent, make up the Fluent as a Server capability:

- The ANSYS Fluent application with built-in Internet Inter-ORB Protocol (IIOP) interface
- The Fluent Remote Console client

- The Fluent as a Server Software Development Kit (SDK) which enables you to build customized client applications

Figure 1.1: Fluent as a Server System Schematic (p. 14) shows how these elements are used together to provide client access to the Fluent solver.

Figure 1.1: Fluent as a Server System Schematic


Client Applications
FLUENT Remote Console
User-developed client application (with CORBA support)
FLUENT As A Server Libraries (user-compiled with 3rd party CORBA Compiler)
FLUENT As A Server Interface (Internet Inter-ORB Protocol)
ANSYS
FLUENT Application
Graphical User Interface
Text User Interface
Batch Mode Interface
Legend
FLUENT As A Server Component
User Developed Component

The Fluent as a Server interface in ANSYS Fluent exists alongside the Graphical User Interface, the Text User Interface, and the Batch Mode Interface. It is made available by starting ANSYS Fluent in Server Mode as described in Fluent as a Server Session Management (p. 17). A client connected to the Fluent as a Server interface can then be used to issue commands remotely to the running Fluent session.

The information necessary to connect to the Fluent as a Server session is stored as an Interoperable Object Reference (IOR) string in a text file on the server machine when the session is started. This IOR string is unique to the session and can be used by a client application to read all necessary information to connect to the Fluent as a Server session (for example, hostname, port number, protocol). For further details on the Fluent as a Server interface and session management, see Fluent as a Server Session Management (p. 17).

The Fluent Remote Console is a precompiled client application that you can use to provide Text User Interface (TUI) commands to a remote Fluent as a Server session. It also supports bi-directional file transfer so you can transfer case/data files, user-defined functions, output files, and so on to or from the solver session as required. For further details on the Fluent Remote Console, see Fluent Remote Console (p. 23).

Alternatively, you can use the Fluent as a Server Software Development Kit (SDK) to build your own customized client application in any development environment that supports the CORBA protocol. The SDK is composed of a set of Interface Definition Language (IDL) files that, when compiled with a 3rd party CORBA compiler, provide a set of libraries that you can include in your application to enable communication with Fluent as a Server. For details about using the Fluent as a Server Software Development Kit, see Fluent as a Server Software Development Kit (SDK) (p. 49).

1.2. Limitations

Note the following limitation:

- All Fluent as a Server commands and most aaS commands cannot be executed until Fluent is idle or at a stable point. For example, Fluent is not stable while solving for an iteration, but at the end of the iteration there is a stable point. This means that there may be a lag between when you enter a command and when you receive a response.

  The minimum lag during a solve operation is one iteration, though you may choose settings that have the effect of increasing it. See Fluent as a Server Commands (`fluent.`) (p. 28) and Collaborative Commands (p. 56) for strategies for minimizing this lag for Fluent as a Server commands and aaS commands, respectively. For a list of aaS commands that can always provide an instant response (that is, without the possibility of a lag), see Instant Commands (p. 54).

1.3. Installation Requirements

1.3.1. Server Machine

A standard ANSYS Fluent installation is all that is required on the server machine to start a Fluent as a Server solver session that can listen for connections from clients (see Fluent as a Server Session Management (p. 17)).

1.3.2. Client Machine

You can connect from a client machine using either the Fluent Remote Console, Fluent Remote Console (p. 23), or your own custom client application built using the Fluent as a Server SDK, Fluent as a Server Software Development Kit (SDK) (p. 49). Both the Fluent Remote Console and the Fluent as a Server SDK are included in the standard Fluent installation.

Chapter 2: Fluent as a Server Session Management

This chapter provides information on starting and managing Fluent as a Server solver sessions that can be accessed by remote client applications.

2.1. Fluent as a Server Sessions

A Fluent as a Server session is very similar to a conventional stand-alone Fluent session, but with the addition of an Internet Inter-ORB Protocol (IIOP) interface exposed that accepts connections from suitable client applications. This capability is different from batch mode operation in that commands can be issued to the running session at any time rather than just from a predefined journal file. This allows solution steering and other manipulations without exiting from the Fluent session.

2.1.1. Starting Fluent In Server Mode

You can start Fluent in Server Mode in either of two ways:

- from the Fluent Launcher by specifying particular environment variables.
- from the command line by passing specific options to the Fluent executable. This is useful if you intend to create a script or batch file to launch Fluent in Server mode.

In either case, the Fluent as a Server session creates a keyfile in the session working directory with the information necessary for a client to connect to the session. See Fluent as a Server Keyfile (p. 20) and Connecting to Fluent In Server Mode (p. 21) for details.

Important:

While it is technically possible to start multiple Fluent as a Server sessions in the same working directory, this is not recommended as it could lead to inadvertent user file conflicts or corruption. For instance, two users, each running their own simulations, could both attempt to write out contour plots titled `pressure_contours.png` leading to one user inadvertently overwriting the other's file.

2.1.1.1. Startup Options

The startup and behavior of Fluent as a Server can be controlled by several command line options and environment variables.

Environment Variables

FLUENT_AAS
controls whether to start Fluent in Server mode. If FLUENT_AAS is set to `1`, the Fluent session will be started in Server mode.

> **Note:**
>
> `FLUENT_AAS` is observed only in the Environment tab of the Fluent Launcher.

AAS_HOST
can be set to an IP address on the server machine to allow connection only from clients with network access to the specified IP. Connections can be restricted to being from the local machine only by setting `AAS_HOST=localhost`. If `AAS_HOST` is not set, any client with access to at least one IP address can access the Fluent as a Server session (this is equivalent to having all possible IPs in the DNS of the server in the `AAS_HOST` variable). This behavior is useful when starting Fluent as a Server session using SGE where the IP address of the host machine is not known in advance.

AAS_PORTS
can be set to a port number or a range of port numbers on which to listen for connections. If set to a single port number, Fluent will attempt to use that port to listen for connections. You can specify a range of ports by using the format:

`<startport>/portspan=<num>`

In this case, Fluent will use the first available in the range of `<num>` ports beginning with `<startport>`. For example, to specify that Fluent should use a port in the range of 1000–1009 you would use:

```
AAS_PORTS=1000/portspan=10
```

FLUENT_AAS_KEY_FILE
specifies a name for the keyfile generated when Fluent as a Server is started. This name will only be used if no name is specified with the `–aas_key_file` command line option. See Fluent as a Server Keyfile (p. 20) for details.

Command Line Options

—aas
instructs Fluent to start in server mode. See Steps to Start Fluent in Server Mode From the Command Line (p. 20).

—aas_key_file
can be used to specify the name of the key file generated by Fluent as a Server containing connection information for the session. See Fluent as a Server Keyfile (p. 20).

2.1.1.2. Steps To Start Server Mode From Fluent Launcher

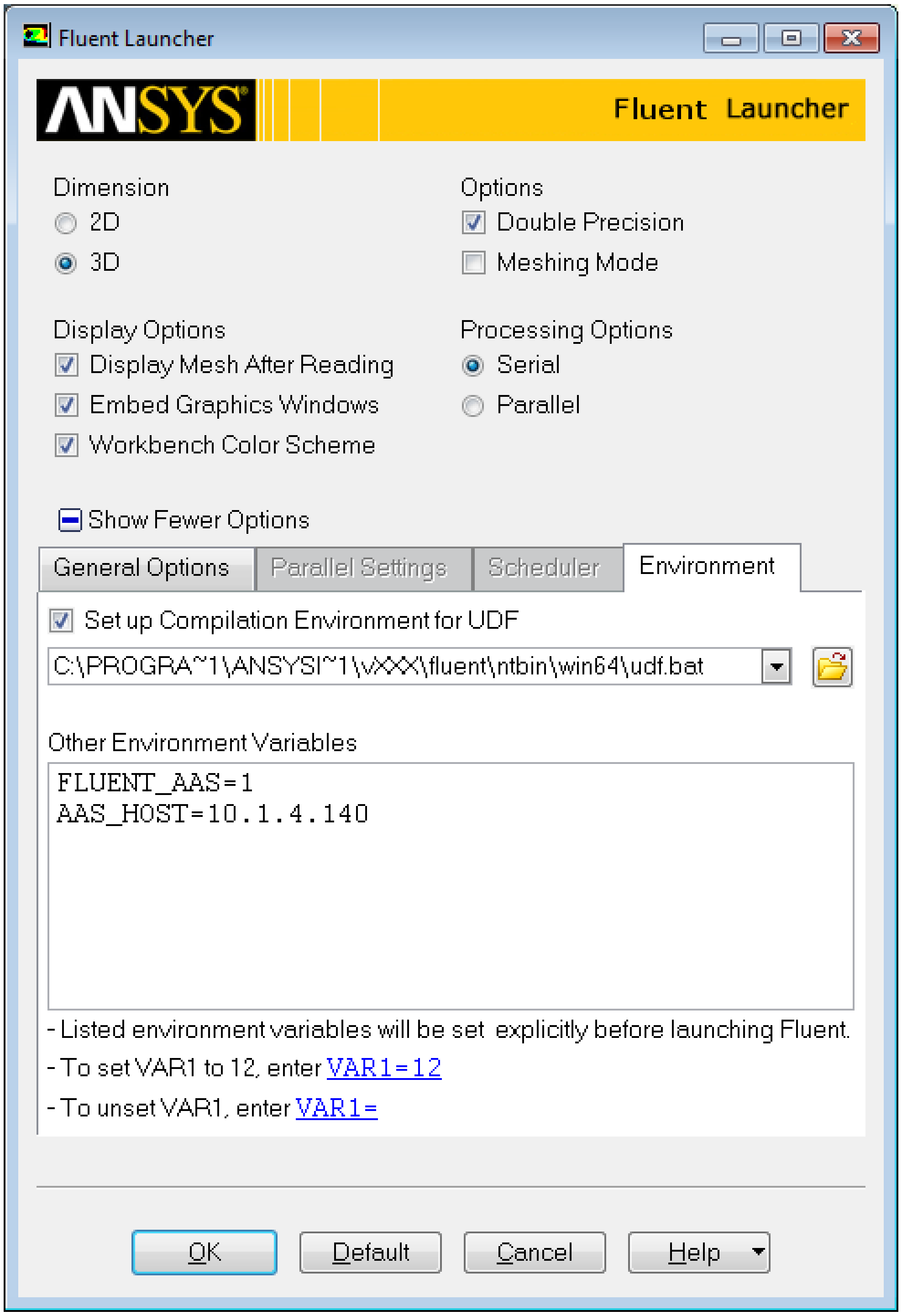
To start Fluent as a Server from the Fluent Launcher, perform the following steps:

1. Start Fluent Launcher and configure dimensionality and options as you normally would (see *Starting and Executing ANSYS Fluent* in the *Getting Started Guide*).

2. Click the **Show More Options** button to expand the Launcher settings (if necessary).

3. Open the **Environment** tab in the expanded Fluent Launcher window.

4. Set the FLUENT_AAS and, optionally, AAS_HOST and FLUENT_AAS_KEY_FILE environment variables.



```
FLUENT_AAS=1
AAS_HOST=10.1.4.140
```

- Listed environment variables will be set explicitly before launching Fluent.
- To set VAR1 to 12, enter VAR1=12
- To unset VAR1, enter VAR1=

a. Enter FLUENT_AAS=1 in the **Other Environment Variables** field.

Setting `FLUENT_AAS=1` instructs Fluent to start in Server Mode. If `FLUENT_AAS` is omitted or is set to `0`, Fluent will start without entering Server Mode and the Fluent as a Server IIOP interface will not be available.

> **Important:**
>
> The `FLUENT_AAS` environment variable is only respected when specified inside Fluent Launcher. Setting `FLUENT_AAS` as a shell environment variable will have no effect.

b. Optionally enter `AAS_HOST=`**`<IP Address>`** in the **Other Environment Variables** field (see Startup Options (p. 17)).

c. Optionally enter `FLUENT_AAS_KEY_FILE=`**`<keyfile-name>`** in the **Other Environment Variables** field (see Startup Options (p. 17)).

5. Click **OK**.

2.1.1.3. Steps to Start Fluent in Server Mode From the Command Line

To start Fluent as a Server from the command line, perform the following steps:

1. Optionally set the shell environment variable `AAS_HOST` to an IP address on the server machine on which Fluent should listen for client connections (see Startup Options (p. 17)).

2. Use the following commands to start Fluent in Server Mode:

   - On Windows platforms:

     ```
     C:\> fluent.exe <options> -aas
     ```

   - On Linux platforms:

     ```
     > fluent <options> -aas
     ```

   These command lines assume that the `fluent` executable is in your path. You may use any **`<options>`** you normally would to specify which solver to load (for example `2d`, `2ddp`, `3d`, `3ddp`), the number of processors to use, and so on. For details on available command line options, see Command Line Startup Options in the Fluent Getting Started Guide.

2.1.2. Fluent as a Server Keyfile

As described in Starting Fluent In Server Mode (p. 17), when you start a Fluent as a Server session, a keyfile is created in the session working directory that contains the information required for a client application to connect to the session. The keyfile is simply a text file containing an Interoperable Object Reference (IOR) string that contains encoded information about the host, port number, and protocol for connecting to the Fluent as a Server session.

The keyfile will be named according to the first value from the following hierarchy:

1. the name specified with the Fluent command line option, `–aas_key_file=<keyfilename>`.
2. the value of the `FLUENT_AAS_KEY_FILE` environment variable.
3. the default name, `aaS_FluentId.txt`.

If the desired keyfile name already exists and refers to a running Fluent as a Server session, the new Fluent as a Server session will fail to start. This is to prevent inadvertently overwriting a keyfile that may be needed by another user.

Important:

While it is technically possible to start multiple Fluent as a Server sessions in the same working directory by specifying different keyfile names, this is not recommended as it could lead to inadvertent user file conflicts or corruption. For instance, two users, each running their own simulations, could both attempt to write out contour plots titled `pressure_contours.png` leading to one user inadvertently overwriting the other's file.

Note that spaces are allowed in the filepath of the keyfile, even though typically Fluent text commands cannot access such filepaths. For example, the following File Session Manager command would not return an error: `fsm.connect_to_session "C:/Path With Spaces/aaS_FluentId.txt"`.

2.2. Connecting to Fluent In Server Mode

Once a Fluent Server Mode session is running it is available for connections from local or remote client applications that have network access to the server machine.

In order to connect to the Fluent session, you will need access to the IOR string contained in the keyfile for the session (see Fluent as a Server Keyfile (p. 20)). Because the keyfile is simply a text file, any convenient means of copying the file, or its contents, to a location accessible to the client machine can be used.

The Fluent Remote Console, Fluent Remote Console (p. 23), is a pre-built client provided with ANSYS Fluent that can be used to connect to Fluent Server sessions and fully control the simulation using TUI commands. It includes a command to read the IOR string from a specified Fluent as a Server keyfile.

Alternatively, you can connect using your own client that has been built with the Fluent as a Server CORBA interfaces, `ICoFluentUnit` and `ICoFluentSchemeController`, that are supplied as part of the Fluent as a Server SDK (see Fluent as a Server Software Development Kit (SDK) (p. 49) for details on creating a client application and connecting to Fluent as a Server).

Chapter 3: Fluent Remote Console

The Fluent Remote Console is a client application that allows you to connect to a Fluent as a Server session on a local or remote machine, potentially running a different operating system. Once connected to a Fluent session the Remote Console provides an interface similar to the interactive Fluent TUI interface. The user can send TUI commands from the Remote Console and receive the text output from the Fluent session in an output window as well as load and retrieve input and output files.

This chapter includes the following sections.

3.1. Using the Fluent Remote Console

The Fluent Remote Console executable is installed in the Fluent install tree.

It can be run either interactively or in batch mode. You can start it from the command line using the following commands:

On Windows Platforms

```
C:\> \Program Files\ANSYS Inc\V202\fluent\fluent20.2.0\launcher\<arch>\flconsole.exe
```

On Linux Platforms

```
> /ansys_inc/v202/fluent/fluent20.2.0/launcher/<arch>/flconsole
```

Command Line Options

On both Windows and Linux platforms, the following command line options are available:

-i *<input_file>.fcj*

specifies a journal file, *<input_file>.fcj*, that contains Fluent Remote Console and TUI commands to be executed in batch mode.

Important:

Due to the additional commands available in Fluent Remote Console, this journal file is not generally of the same format as a Fluent journal file.

-o ***<output_file>.fct***
specifies an output transcript file, *<output_file>.fct*, to which output from Fluent Remote Console will be written.

Important:

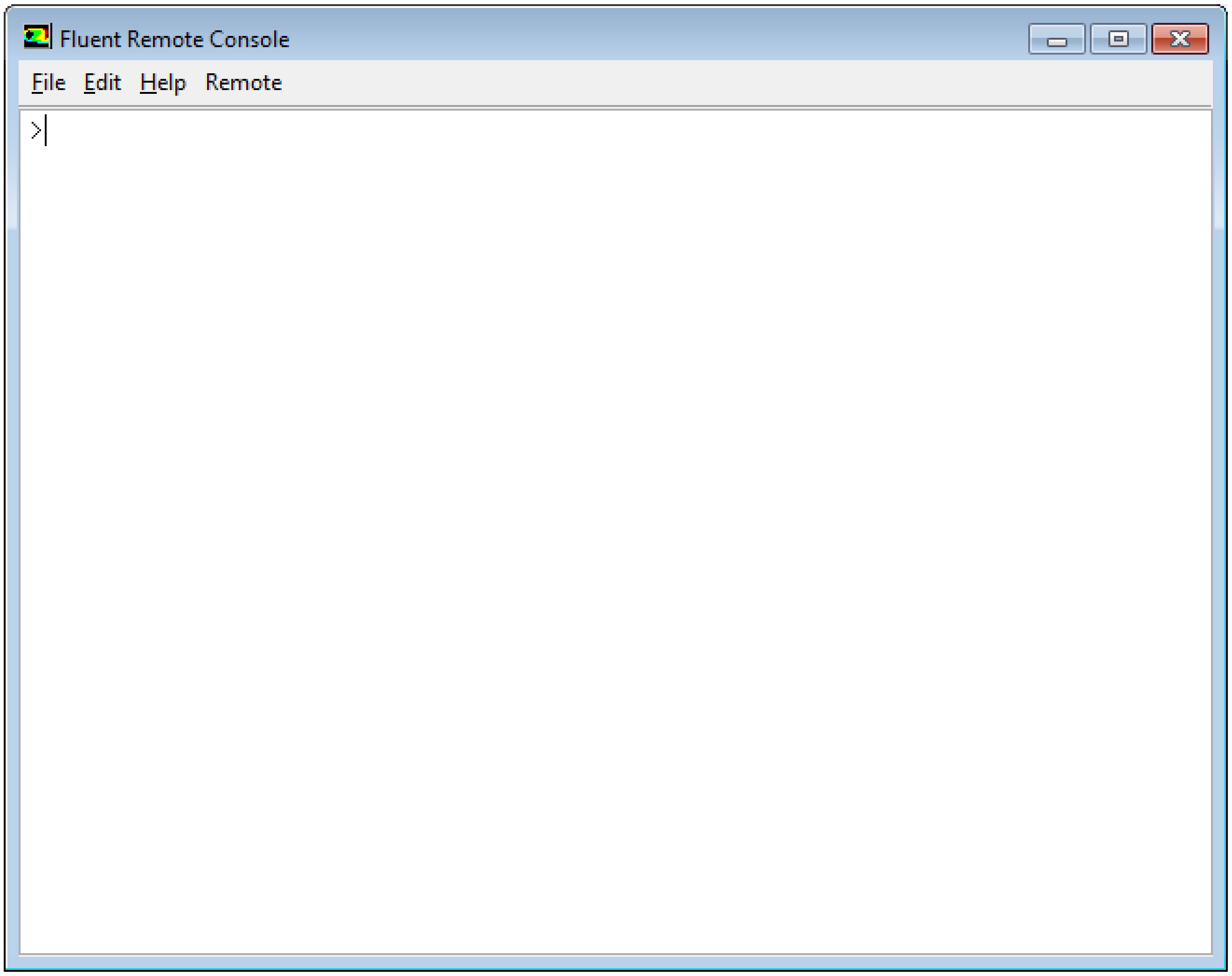
This transcript file is not generally of the same format as a Fluent transcript file.

-nogui
used in batch mode to specify that Fluent Remote Console should run without displaying the Fluent Remote Console or Fluent Output windows.

3.1.1. The Fluent Remote Console Window

When started interactively (that is, without the `-nogui` command line option), the Fluent Remote Console window appears, Figure 3.1: Fluent Remote Console Window (p. 24).

Figure 3.1: Fluent Remote Console Window



Commands are entered at the Fluent Remote Console command prompt, >. Session management output is displayed in the Fluent Remote Console. Fluent as a Server simulation output may be displayed either in the Fluent Remote Console window or in the Fluent Output window (The Fluent Output Window (p. 27)) depending on verbosity settings (Fluent as a Server Commands (`fluent.`) (p. 28)).

Typically, the first step after launching Fluent Remote Console is to connect to a running Fluent as a Server session. You can do this using the `fsm.connect_to_session` command:

```
>fsm.connect_to_session <path to keyfile>
```

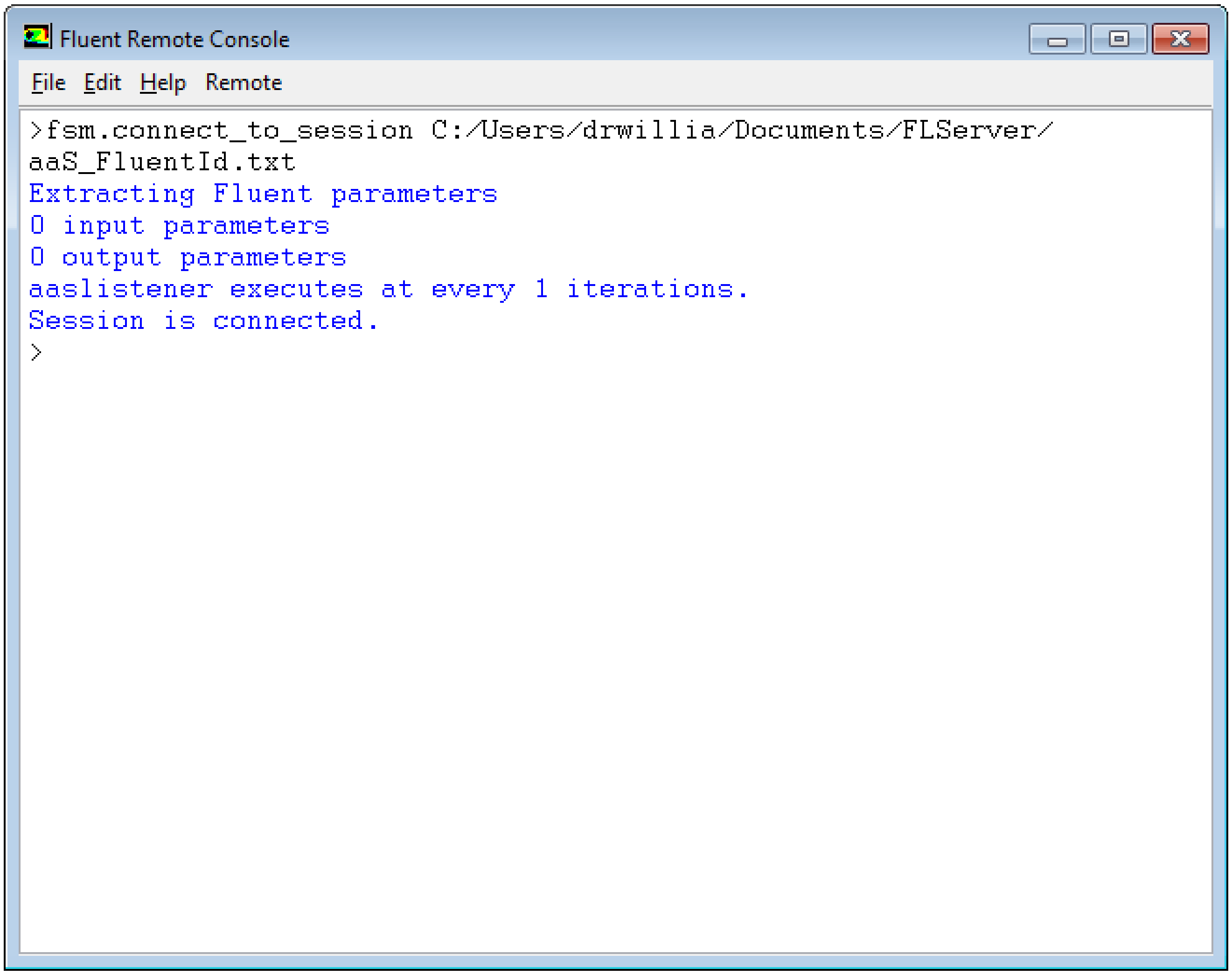
As discussed in Fluent as a Server Keyfile (p. 20), each Fluent as a Server session creates a unique keyfile, by default named `aaS_FluentId.txt`, which contains connection information for clients to use. The `fsm.connect_to_session` command uses the information in this file to initiate a connection to Fluent as a Server. You can also choose a session keyfile using the **File** → **Open...** menu command.

Note:

The session keyfile can be found in the working directory of the Fluent as a Server session and can be copied to another location (perhaps on the client machine) if convenient.

Once connected, Fluent Remote Console will display a connection message and, if a case file is already loaded on the server, the status of any input/output parameters that exist, Figure 3.2: Fluent Remote Console Connected to Fluent Session (p. 26). The Fluent Output Window will also open and will display output from the remote server TUI console.

Figure 3.2: Fluent Remote Console Connected to Fluent Session



```
>fsm.connect_to_session C:/Users/drwillia/Documents/FLServer/
aaS_FluentId.txt
Extracting Fluent parameters
0 input parameters
0 output parameters
aaslistener executes at every 1 iterations.
Session is connected.
>
```

Commands are entered at the prompt in the Fluent Remote Console and, by default, output will be directed back to the Fluent Remote Console. You can optionally direct output to the Fluent Output Window using the `fluent.set_silent` command in the Fluent Remote Console. See The Fluent Output Window (p. 27) for details.

The available command set in the Fluent Remote Console depends on the connection status. At any time you may press `<Tab>` to see a context-sensitive list of currently available commands (or command completions if you have already entered some characters).

Commands in Fluent Remote Console fall into the following categories:

- File Session Management commands (prefixed by `fsm.`)
- Fluent as a Server commands (prefixed by `fluent.`)
- Fluent TUI commands (prefixed by `tui.`)
- Remote File and Shell Commands

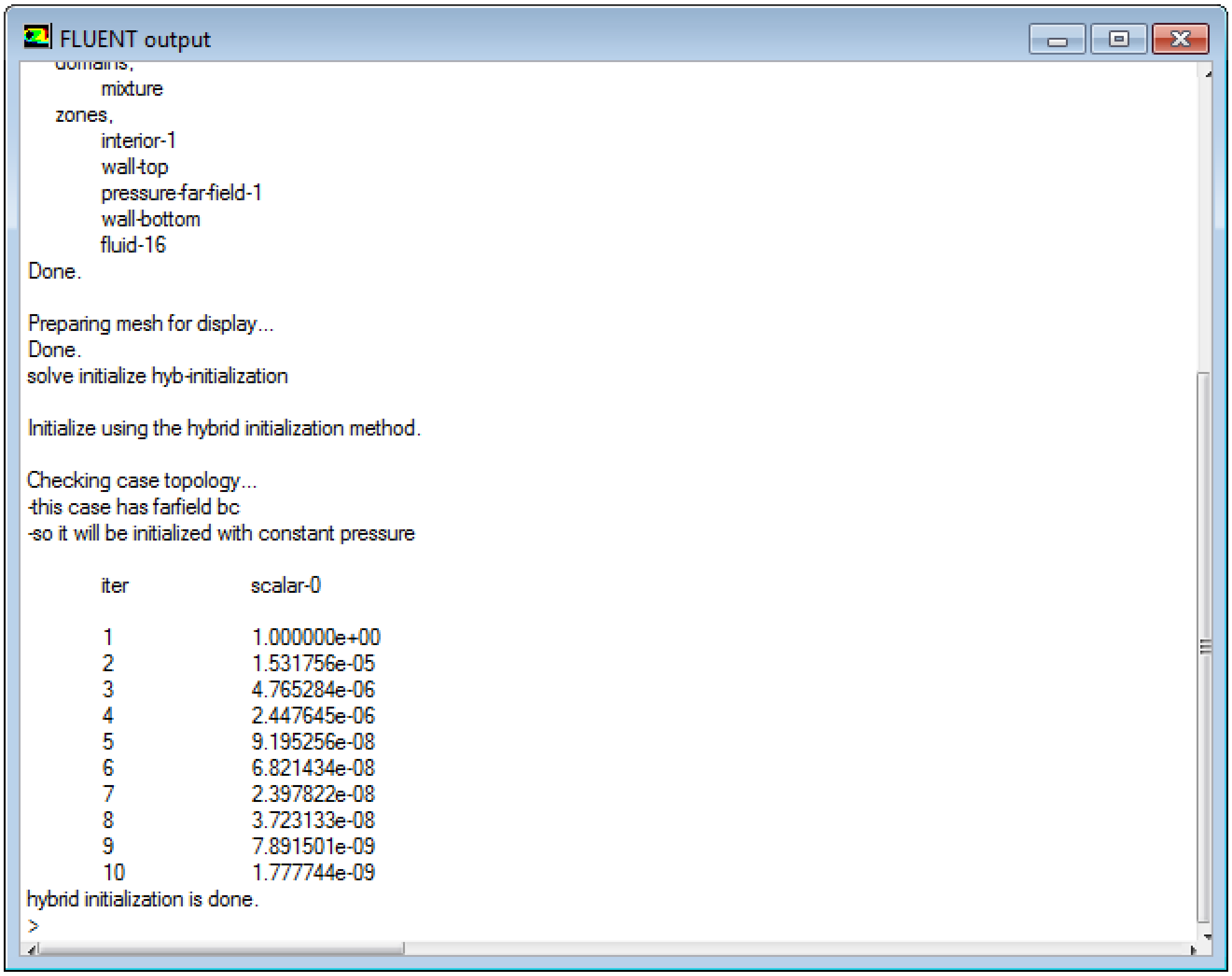
For detailed descriptions of all the commands available in the Fluent Remote Console, see Fluent Remote Console Commands (p. 28). To exit the Fluent Remote Console you can type `exit` at the prompt. Note that this will *not* terminate any connected Fluent as a Server instances.

For a complete example of using Fluent Remote Console to perform a simulation see Fluent Remote Console Example (p. 35).

3.1.2. The Fluent Output Window

The Fluent Output Window, Figure 3.3: Fluent Output Window (p. 27), displays the contents of the remote Fluent as a Server session TUI console. Based on the verbosity settings of the Remote Console, this may include output from commands issued from Fluent Remote Console.

Figure 3.3: Fluent Output Window



```
  domains,
         mixture
  zones,
         interior-1
         wall-top
         pressure-far-field-1
         wall-bottom
         fluid-16
Done.

Preparing mesh for display...
Done.
solve initialize hyb-initialization

Initialize using the hybrid initialization method.

Checking case topology...
-this case has farfield bc
-so it will be initialized with constant pressure

         iter            scalar-0

         1               1.000000e+00
         2               1.531756e-05
         3               4.765284e-06
         4               2.447645e-06
         5               9.195256e-08
         6               6.821434e-08
         7               2.397822e-08
         8               3.723133e-08
         9               7.891501e-09
         10              1.777744e-09
hybrid initialization is done.
>
```

By default, TUI output from remote commands issued to the Fluent as a Server session from Fluent Remote Console are displayed in the Fluent Remote Console window. If you prefer to have the output displayed in the Fluent Output window you can issue the `fluent.set_silent` command in the Fluent Remote Console. This will prevent output from `tui.*` and `fluent.*` commands from appearing in the Fluent Remote Console window. The output will instead be directed to the remote server TUI console and mirrored in the Fluent Output Window on the local client machine. You can direct output back to the Fluent Remote Console using `fluent.set_verbose`. For further information on verbosity settings and output commands, see Fluent as a Server Commands (`fluent.`) (p. 28)).

3.2. Fluent Remote Console Commands

Commands in Fluent Remote Console are divided into the following categories and namespaces:

3.2.1. File Session Manager Commands (fsm.)

3.2.2. Fluent as a Server Commands (fluent.)

3.2.3. Fluent TUI Commands (tui.)

3.2.4. Remote File and Shell Commands

3.2.1. File Session Manager Commands (`fsm.`)

File Session Manager commands are used to manage aspects of the local Fluent Remote Console session and file system. You can connect to Fluent sessions that are listening for connections on the Fluent as a Server interface by specifying a keyfile (see Fluent as a Server Session Management (p. 17)).

File Session Manager commands begin with the prefix `fsm`.

fsm.connect_to_session <filename>
connect to a running Fluent as a Server session. **`<filename>`** is the path to the Fluent as a Server keyfile corresponding to the solver session. If **`<filename>`** is omitted, a dialog box will open which allows you to navigate to a file. You can also choose **File** → **Open...** from the menu bar.

fsm.cd <directory>
change the current local directory to **`<directory>`**. If **`<directory>`** is omitted, a dialog box will open, allowing you to navigate to a directory.

fsm.ls
lists the contents of the current local directory.

fsm.pwd
print the current local directory.

fsm.display_users_guide
opens the ANSYS Help. You can also press **F1** or choose **Help** → **User's Guide...** from the menu bar.

3.2.2. Fluent as a Server Commands (`fluent.`)

Fluent as a Server commands are used to perform client/server interactions with a connected Fluent as a Server session. These include commands to upload files, set/get rpvar values, alter verbosity, and so on. Fluent as a Server commands begin with the prefix `fluent`. Note that for the sake of convenience and accuracy, you do not have to type the whole command, but can begin typing and use the **Tab** key to automatically complete it or display a list of options.

The following commands collaborate with Fluent processes and therefore cannot be executed until Fluent is idle or at a stable point. This means that you are not guaranteed to have an instant response when you enter the command.

Tip:

The following strategies can be used to minimize the lag between when you enter a command and when it is executed:

- Set the `fluent.set_aaslistening_step_at <number> iteration|timestep` command to a minimum value, based on your objectives. Note that by decreasing the iterations / timesteps between stable points, you are favoring cosimulation over simulation (that is, you are prioritizing your ability to manipulate the settings over the time it will take to calculate the solution).
- You can nest a number of commands between `fluent.pause` and `fluent.continue` commands, so that they are all executed during the same stable point; otherwise, each command will have to wait for a separate stable point.

---

fluent.continue
continues remote execution of a journal file on the Fluent server.

fluent.display_menu
displays a message of the remote TUI menu.

fluent.download_file [-f] <filename>
downloads a file, **`<filename>`**, from the remote Fluent session working directory to the local machine. The file will be downloaded to the local directory from which Fluent Remote Console was started. By default, `fluent.download_file` will not overwrite an existing local file. You can force an overwrite by including the `–f` flag.

fluent.get_aaslistening_step
provides information about the aaslistener that listens for commands from Fluent as a Server clients.

fluent.get_current_flow_time
provides the current flow time.

fluent.get_current_iteration
provides the current iteration.

fluent.get_current_time_step
provides the current time step.

fluent.get_dimensionality
provides the dimensionality of the current case.

fluent.get_last_casefile_name
provides the name of the last case file accessed before the current case. This can be useful if you have written the case file with a new name.

fluent.get_list_input_parameters
provides a list of the input parameters available in the current case.

fluent.get_list_output_parameters
provides a list of the output parameters available in the current case.

fluent.get_nr_input_parameters
provides the number of input parameters available in the current case.

fluent.get_nr_output_parameters
provides the number of output parameters available in the current case.

fluent.get_parameter <parametername>
reports the current value of **`<parametername>`**.

fluent.get_paused_index
returns an index indicating the "depth" of pause. When Fluent is not paused, it will return `0`. It is incremented with each successive pause instruction. This can be useful to determine the state of a session if the connection is lost while Fluent is paused.

fluent.get_precision
specifies whether the solution calculations for the current case will be performed in single- or double-precision mode.

fluent.get_release
provides the name of the solver and the release number.

fluent.get_status
provides information about the status of the Fluent session (for example, `journaling`, `iterate`).

fluent.get_rpvar <rpvarname>
reports the current value of **`<rpvarname>`**.

fluent.get_status
provides information about the status of the Fluent session (for example, `journaling`, `iterate`).

fluent.get_verbosity
displays the current setting of verbosity.

fluent.interrupt
interrupts remote execution of a journal file on the Fluent server. Note that this terminates reading of the journal file. If you want to be able to resume execution where it was stopped, use `fluent.pause`.

fluent.list_commands
returns a list of solver calculation activities (including the listening process for Fluent as a Server commands).

fluent.list_monitors <monitor_type> <monitor_name>
returns a JSON-formatted listing of the monitor definitions in the current case. **`<monitor_type>`** may be one of the following types: `residuals`, `lift`, `drag`, `moment`, `surface`, or `volume`. If no arguments are given, Fluent returns all defined monitors. If only **`<monitor_type>`** is given, all monitors of the specified type are returned.

fluent.list_monitors_with_persisted_data <monitor_type> <monitor_name> <filter> <filter_value>
: returns a JSON-formatted listing of the monitor definitions in the current case, including the data from the monitor history file (if any). Options for **`<monitor_type>`** are the same as those listed for `fluent.list_monitors`. **`<filter>`** allows you to return incremental monitor data, by removing the data calculated before the specified **`<filter_value>`**; a single command can include one or more of the following **`<filter>`** options: `start-flow-time`, `start-iteration`, and `start-time-step`.

fluent.list_parameters
: reports the names and values of the input and output parameters defined in the current case.

fluent.list_report_definitions
: returns a JSON-formatted listing of the report definitions in the current case. For more information about report definitions, see Creating Report Definitions in the *Fluent User's Guide*.

fluent.list_report_files
: returns a JSON-formatted listing of the report file definitions in the current case. If no arguments are given, Fluent returns all defined report files. For more information about report files, see Report Files and Report Plots in the *Fluent User's Guide*.

fluent.list_report_files_with_persisted_data <filter> <filter_value>
: returns a JSON-formatted listing of the report file definitions in the current case, including the data from the report file `.out` file (if any). For more information about report files, see Report Files and Report Plots in the *Fluent User's Guide*. **`<filter>`** allows you to return incremental report file data, by removing the data calculated before the specified **`<filter_value>`**; a single command can include one or more of the following **`<filter>`** options: `start-iteration` and `start-time-step`.

fluent.list_report_plots
: returns a JSON-formatted listing of the report plot definitions in the current case. For more information about report plots, see Report Files and Report Plots in the *Fluent User's Guide*.

fluent.list_rpvars
: lists all rpvars by name and their current values. This command will not return anything until after `fluent.retrieve_rpvars_info` has been issued.

fluent.list_rpvars_regexp '<key>'
: lists all rpvars matching the specified regular expression, **`<key>`**, and their current values. Note that **`<key>`** must be enclosed in single quotes. This command will not return anything until after `fluent.retrieve_rpvars_info` has been issued.

fluent.list_rpvars_wildcards '<key>'
: lists all rpvars matching **`<key>`** with wildcard expansion. Use **`*`** to match zero or more characters. Use ? to match exactly one character. Note that **`<key>`** must be enclosed in single quotes. This command will not return anything until after `fluent.retrieve_rpvars_info` has been issued.

fluent.load_case <filename>
: loads a case file over the network connection to the Fluent session. Note that this is different from the `read-case` TUI command because it allows you to load a local case file to the remote session. If **`<filename>`** is omitted, a dialog box will open, allowing you to navigate to a file.

fluent.load_data <filename>

loads a data file over the network connection to the Fluent session. Note that this is different from the `read-data` TUI command because it allows you to load a local data file to the remote session. If **`<filename>`** is omitted, a dialog box will open, allowing you to navigate to a file.

fluent.pause

pauses remote execution of a journal file on the Fluent server. Execution can be resumed using the `fluent.continue` command.

fluent.read_journal

loads a journal and executes it asynchronously (control is immediately returned to the user). The status of the journal execution can be checked using `fluent.get_status`.

fluent.retrieve_rpvars_info

retrieves the list of rpvars from the connected Fluent as a Server session. This command must be issued before the `fluent.list_rpvars` commands or completion suggestions for `fluent.get_rpvar`/`fluent.set_rpvar` will work.

fluent.save_case <filename>

saves a case file from the remote Fluent session on the local machine. Note that this is different from the `write-case` TUI command because it allows you to save the case file to the local machine. If **`<filename>`** is omitted, a dialog box will open, allowing you to navigate to a file location.

fluent.save_data <filename>

saves a data file from the remote Fluent session on the local machine. Note that this is different from the `write-data` TUI command because it allows you to save the data file to the local machine. If **`<filename>`** is omitted, a dialog box will open, allowing you to navigate to a file location.

fluent.save_monitors_with_persisted_data <monitor_file_name>

saves a JSON-formatted file on the remote Fluent machine with the data (if it exists) from all monitors defined in the current case.

fluent.set_aaslistening_step_at <number> iteration|timestep

configures the aaslistener to listen for commands from Fluent as a Server clients every **`<number>`** iterations or timesteps.

fluent.set_auto_save_period_json_monitors <period_in_minutes>

allows you to change the period for when monitor history data is automatically saved as a JSON-formatted file (named `monitor.json`) on the remote Fluent machine. By default, the period is set to 60 minutes. Note that it is recommended that you do not have multiple sessions in a single folder, as they will all have the same name for the monitor history file.

fluent.set_parameter <parametername> <value>

sets the value of **`<paramtername>`** to **`<value>`**.

fluent.set_rpvar <rpvarname> <value>

sets **`<rpvarname>`** to **`<value>`**.

fluent.set_server_questions_silent

disables prompts for TUI commands that are entered without all arguments. Fluent will automatically use the default values for any arguments that are not specified.

fluent.set_server_questions_verbose
enables prompts for TUI commands that are entered without all arguments.

fluent.set_silent
sets the output of TUI commands issued in the Fluent Remote Console to display in the Fluent Output window instead of in the Fluent Remote Console.

fluent.set_verbose
sets the output of TUI commands issued in the Fluent Remote Console to display in the Fluent Remote Console instead of in the Fluent Output window. This is the default behavior.

fluent.terminate
terminates the connected Fluent solver session.

fluent.upload_file [-f] <filename>
uploads a file from the local client machine to the remote Fluent session working directory. If **`<filename>`** is omitted, a dialog box will open, allowing you to navigate to a file. By default, `fluent.upload_file` will not overwrite an existing remote file. You can force an overwrite by including the `–f` flag.

3.2.3. Fluent TUI Commands (`tui.`)

Once you have connected to a Fluent session, you will be able to execute the full set of TUI commands in the Fluent Remote Console as if you are running Fluent locally. For details on the available TUI commands, refer to the *Fluent Text Command List*. When issued from the Fluent Remote Console, these commands are formally prepended with the `tui.` namespace prefix. However, Fluent Remote Console treats `tui.` as the default namespace so it can be omitted. For example, the following commands in Fluent Remote Console are equivalent:

```
tui.report summary
report summary
```

3.2.4. Remote File and Shell Commands

For reasons of convenience and security, shell and file system commands behave differently in Fluent Remote Console than in an interactive or batch Fluent session. Specifically:

- Execution of arbitrary shell commands (using the `!` prefix in the Fluent Text User Interface) is not supported from the Fluent Remote Console.
- File system commands (for example, `ls`, `chdir`, and so on) in the Text User Interface are overloaded in the Fluent Remote Console with similar commands that are intended to provide a degree of security against unwanted remote file system access.
- Several additional file system commands are available to manage files in the Fluent as a Server working directory on the server machine.

The remote file system commands provided in the Fluent Remote Console are restricted to operations within the Fluent as a Server session working directory and its subdirectories. These commands will return an error if you attempt to access a file or directory outside of the Fluent as a Server directory.

Warning:

Although the restrictions placed on shell and file system commands in the Remote Console provide a measure of protection against unwanted remote file system access and program execution, they cannot guard against every threat and must not be relied upon for complete security. For example, commands issued from a journal file in the remote Fluent as a Server session will not be subject to the same restrictions. Therefore, it is important that the Fluent as a Server session is started with appropriate permissions on the remote machine to maximize remote system security.

The available shell and file system commands in Fluent Remote Console are as follows:

Note:

Unless otherwise noted, wildcards are not supported.

tui.ls `<name>`
list the contents of directory **`<name>`**. If **`<name>`** refers to a single file, list that file. If **`<name>`** is omitted, list the contents of the current directory.

tui.dir `<name>`
same as `ls`

tui.pwd
print the current directory on the server machine. The path is returned relative to the base working directory of the Fluent as a Server session, which appears as `$home`. Note that `$home` is merely a symbol for the base working directory in the path display. You cannot use `$home` in a path specifier with `tui.chdir`.

tui.chdir `<directory>`
change to **`<directory>`**. If **`<directory>`** is omitted, return to the base working directory of the Fluent as a Server session.

tui.cd `<directory>`
same as `chdir`

tui.mkdir `<name>`
create a new directory called **`<name>.`**

tui.rmdir `<directory>`
delete the directory called **`<directory>`**. Note that **`<directory>`** must be empty before it can be deleted.

tui.cp `<source>` `<dest>`
copy the file **`<source>`** to **`<dest>`**. If **`<dest>`** exists, it is overwritten. If **`<dest>`** is an existing directory, **`<source>`** is copied into **`<dest>`**.

tui.copy `<source>` `<dest>`
same as `cp`.

tui.rename `<source>` `<dest>`
rename the file **`<source>`** to **`<dest>`**. If **`<dest>`** exists, it is overwritten. Note that you can move a file into a directory by forming **`<dest>`** as **`<directory>/<filename>`**.

tui.delete `<file>`
delete the file named **`<file>`**.

3.3. Fluent Remote Console Example

This example illustrates a typical sequence in which Fluent Remote Console is used to connect to a running Fluent Server session and perform a remote simulation. It will demonstrate the following basic tasks:

- Start Fluent Remote Console
- Connect to a Fluent Server session using a keyfile, such as `aaS_FluentId.txt` file
- Upload a case file from the client machine to the Fluent Server
- Get and set parameter and rpvar values for the simulation
- Use TUI commands to initialize and iterate the solution
- Save the case and data files to the client machine
- End the Fluent session

Note:

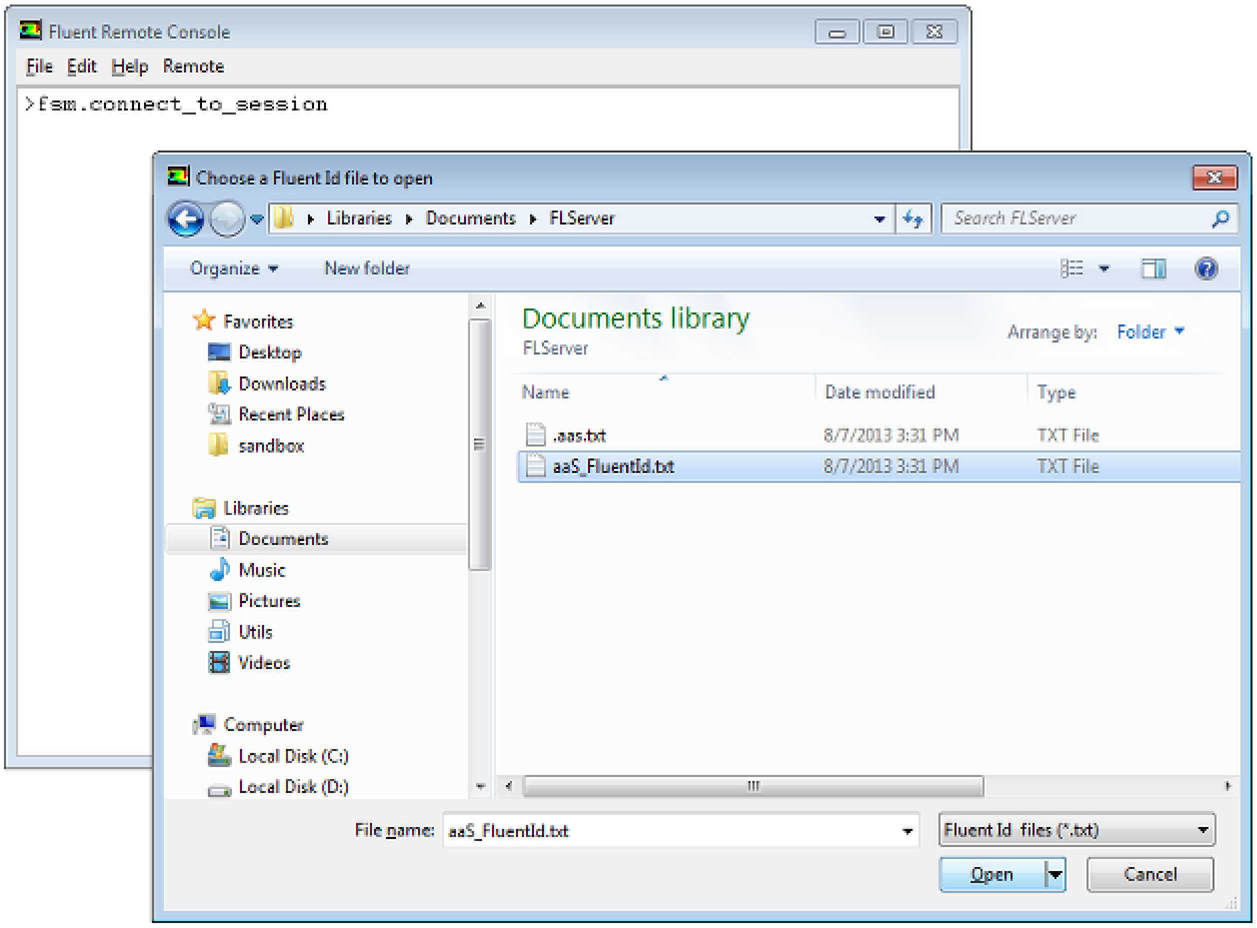
This example presumes that a Fluent Server session is running on a local or remote machine. For details on starting a Fluent Server session, see Starting Fluent In Server Mode (p. 17).

Example Procedure

1. Start Fluent Remote Console using one of the methods in Using the Fluent Remote Console (p. 23).

```
C:\>"\Program Files\ANSYS Inc\v202\fluent\fluent20.2.0\launcher\win64\flconsole"
```

2. Connect to a running Fluent Server session (`fsm.connect_to_session`).



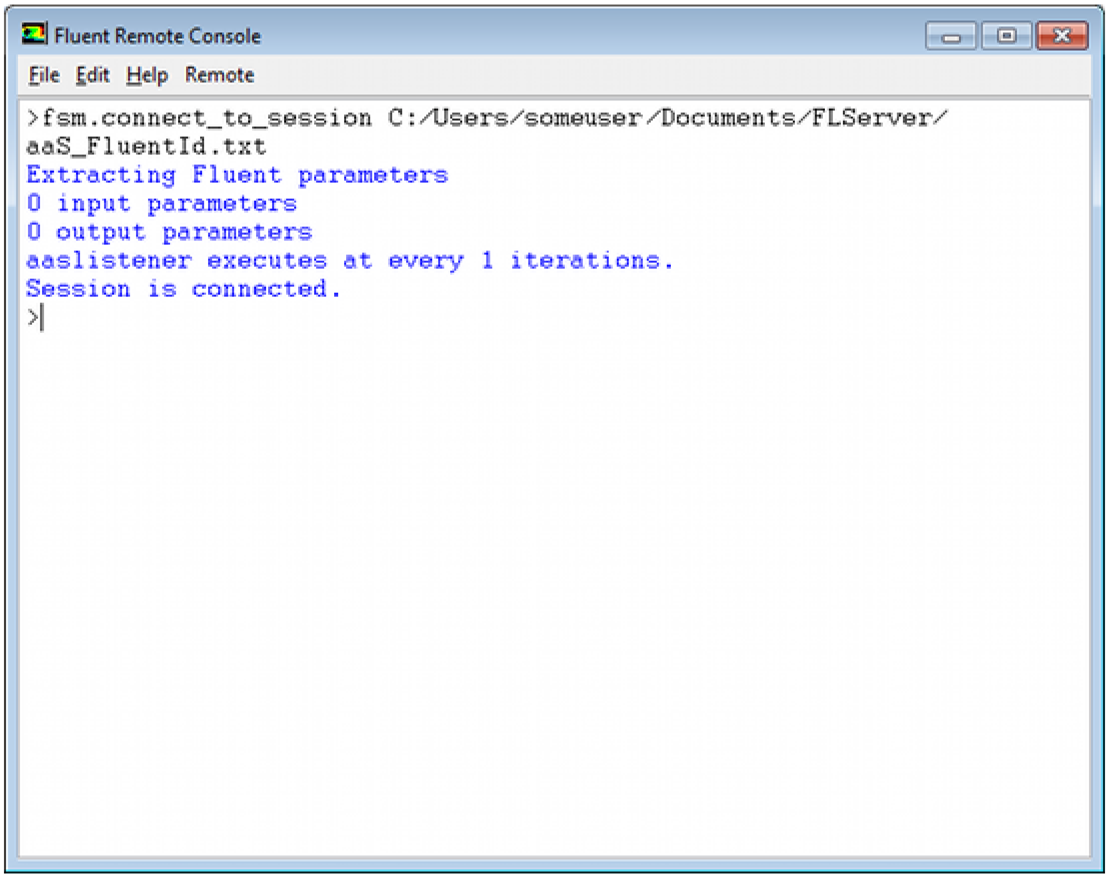
```
>fsm.connect_to_session
```

Name	Date modified	Type
.aas.txt	8/7/2013 3:31 PM	TXT File
aaS_FluentId.txt	8/7/2013 3:31 PM	TXT File

File name: aaS_FluentId.txt

Fluent Id files (*.txt)

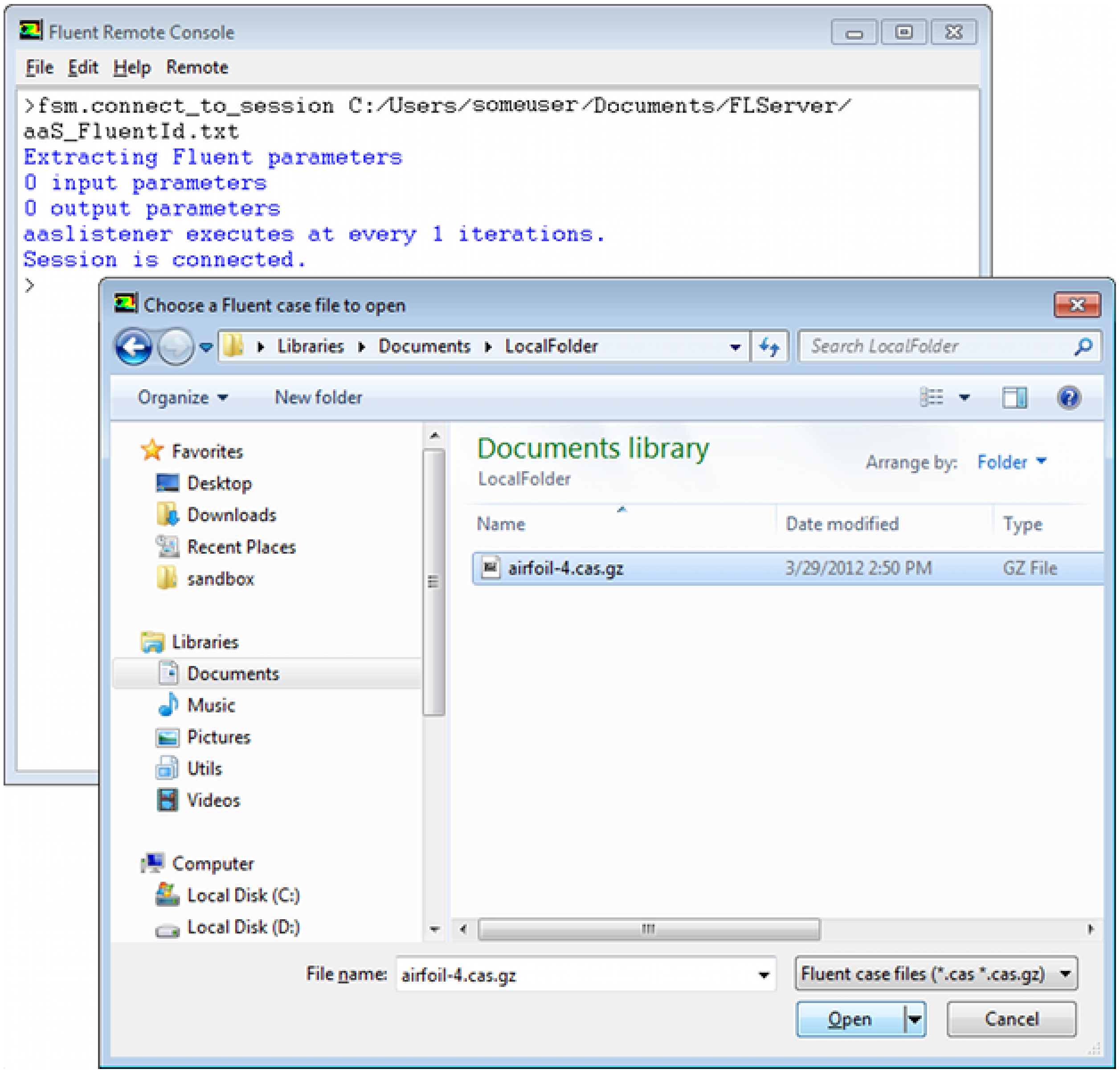
Because a filename was not specified with the `fsm.connect_to_session` *command, a dialog box opens to select the* `aaS_FluentId.txt` *file.*



```
>fsm.connect_to_session C:/Users/someuser/Documents/FLServer/
aaS_FluentId.txt
Extracting Fluent parameters
0 input parameters
0 output parameters
aaslistener executes at every 1 iterations.
Session is connected.
>
```

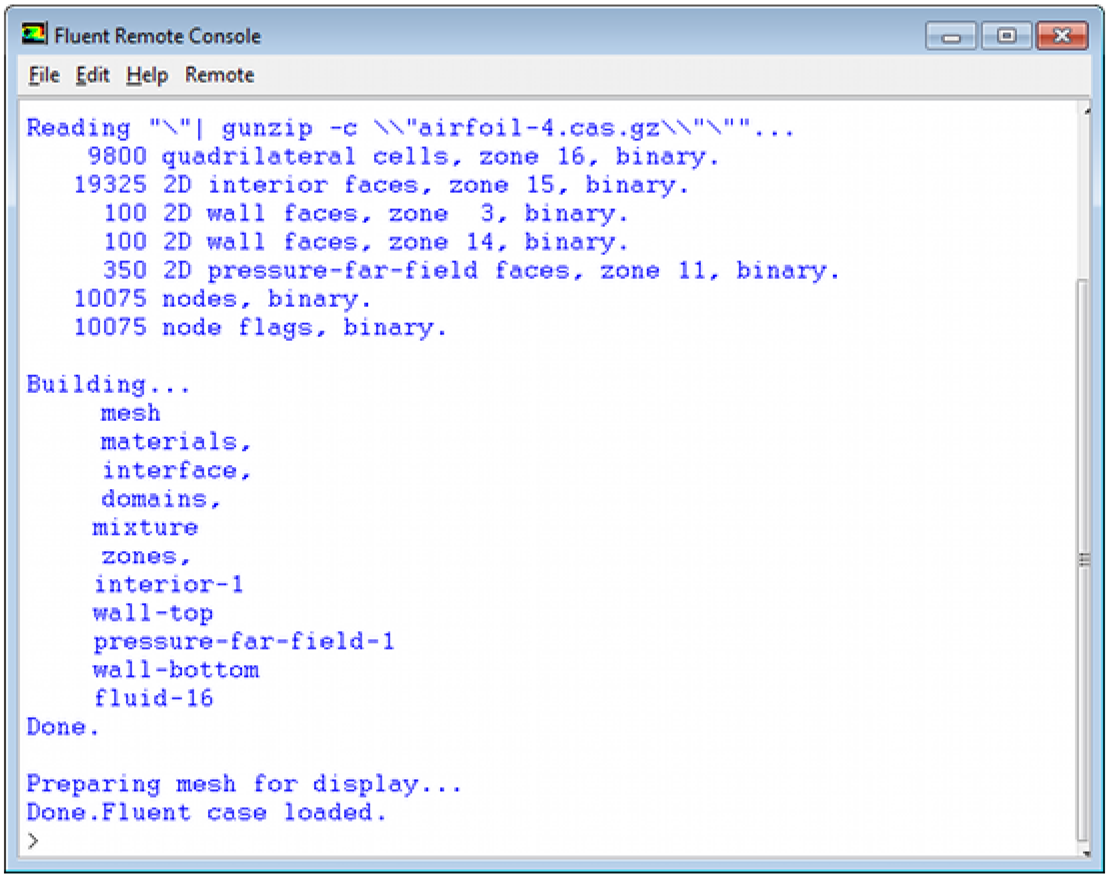
*After clicking **Open**, Fluent Remote Console reports the connection status and lists available parameters. In this example, no case file is loaded in the Fluent session, so no parameters are available.*

3. Load a case file from the local (client) machine (`fluent.load_case`).



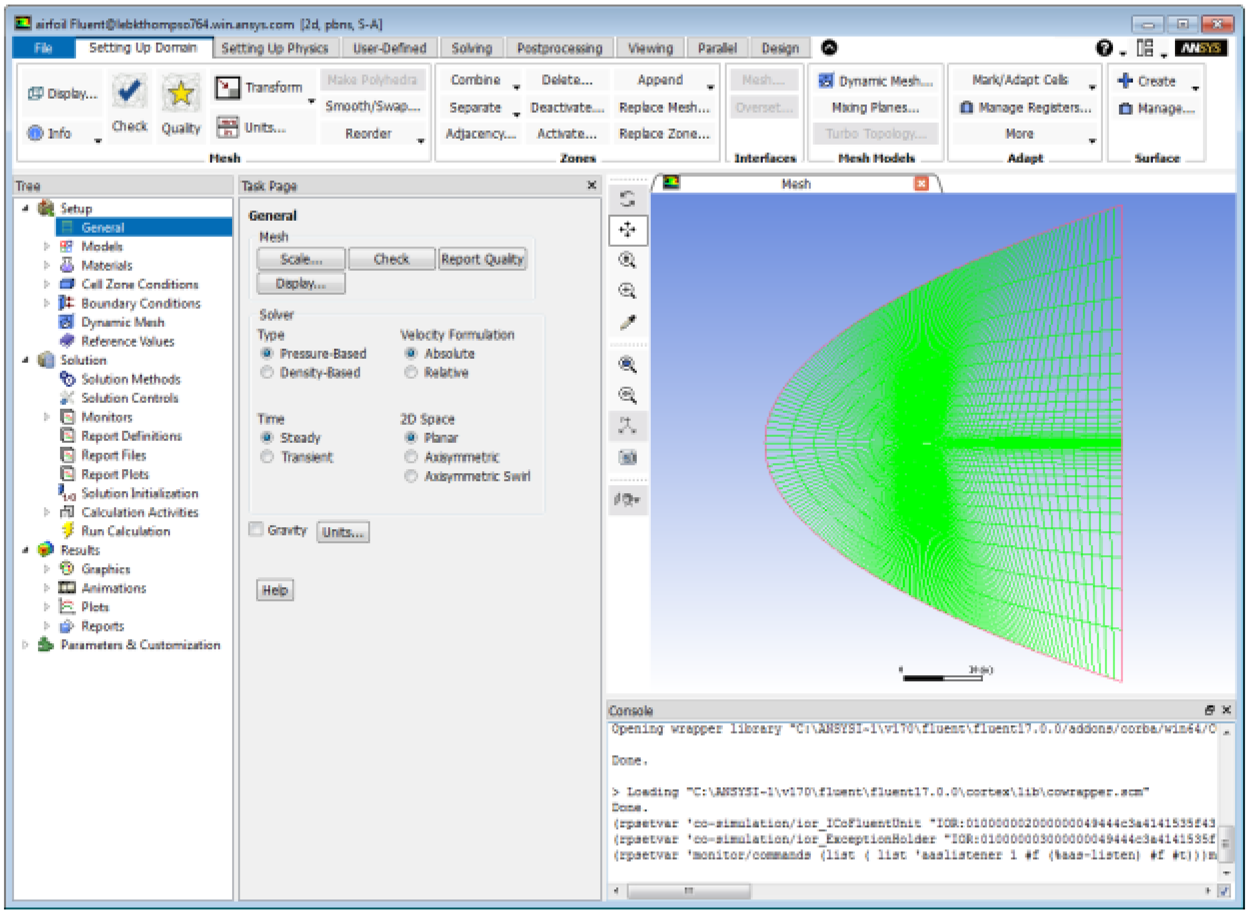
```
>fsm.connect_to_session C:/Users/someuser/Documents/FLServer/
aaS_FluentId.txt
Extracting Fluent parameters
0 input parameters
0 output parameters
aaslistener executes at every 1 iterations.
Session is connected.
>
```

Because a filename was not specified with the `fluent.load_case` command, a dialog box opens to select the case file.



```
Reading "\"| gunzip -c \\"airfoil-4.cas.gz\\"\""...
    9800 quadrilateral cells, zone 16, binary.
   19325 2D interior faces, zone 15, binary.
     100 2D wall faces, zone  3, binary.
     100 2D wall faces, zone 14, binary.
     350 2D pressure-far-field faces, zone 11, binary.
   10075 nodes, binary.
   10075 node flags, binary.

Building...
     mesh
     materials,
     interface,
     domains,
    mixture
     zones,
    interior-1
    wall-top
    pressure-far-field-1
    wall-bottom
    fluid-16
Done.

Preparing mesh for display...
Done.Fluent case loaded.
>
```

Fluent Remote Console reports that the case has been loaded. Note that we use the `fluent.load_case` *command which uploads the case file to the server machine prior to it being read into Fluent.*


airfoil Fluent@lebkthompso764.win.ansys.com [2d, pbns, S-A]

General

Solver Type: [✓ Pressure-Based] Density-Based

Velocity Formulation: [✓ Absolute] Relative

Time: [✓ Steady] Transient

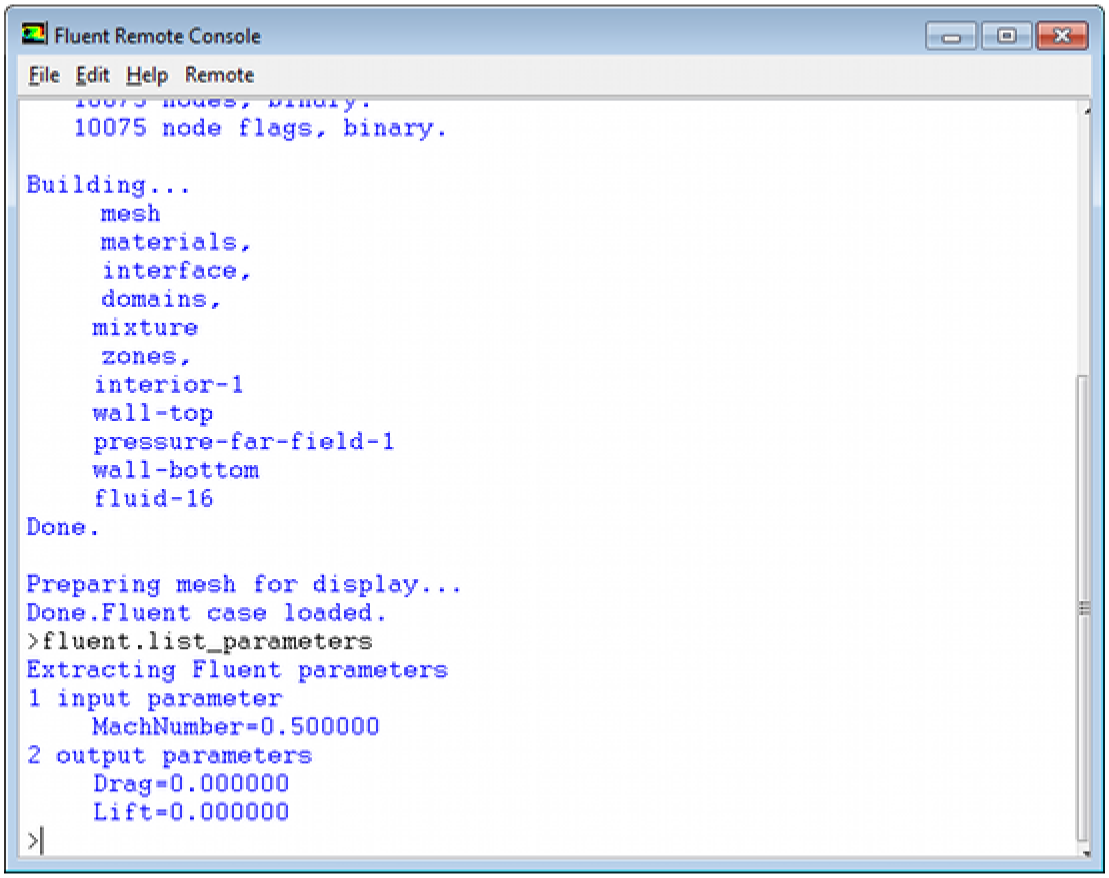
2D Space: [✓ Planar] Axisymmetric Axisymmetric Swirl

```
Opening wrapper library "C:\ANSYSI~1\v170\fluent\fluent17.0.0/addons/corba/win64/C

Done.

> Loading "C:\ANSYSI~1\v170\fluent\fluent17.0.0\cortex\lib\cowrapper.scm"
Done.
(rpsetvar 'co-simulation/ior_ICoFluentUnit "IOR:010000002000000049444c3a41415355f43
(rpsetvar 'co-simulation/ior_ExceptionHolder "IOR:010000003000000049444c3a4141535f
(rpsetvar 'monitor/commands (list ( list 'aaslistener 1 #f (%aas-listen) #f #t)))m
```

The case is loaded into the Fluent session. In this case, the Fluent session was started with graphics enabled so it displays the mesh as it would if running interactively.

4. Retrieve the current parameter values (`fluent.list_parameters`).

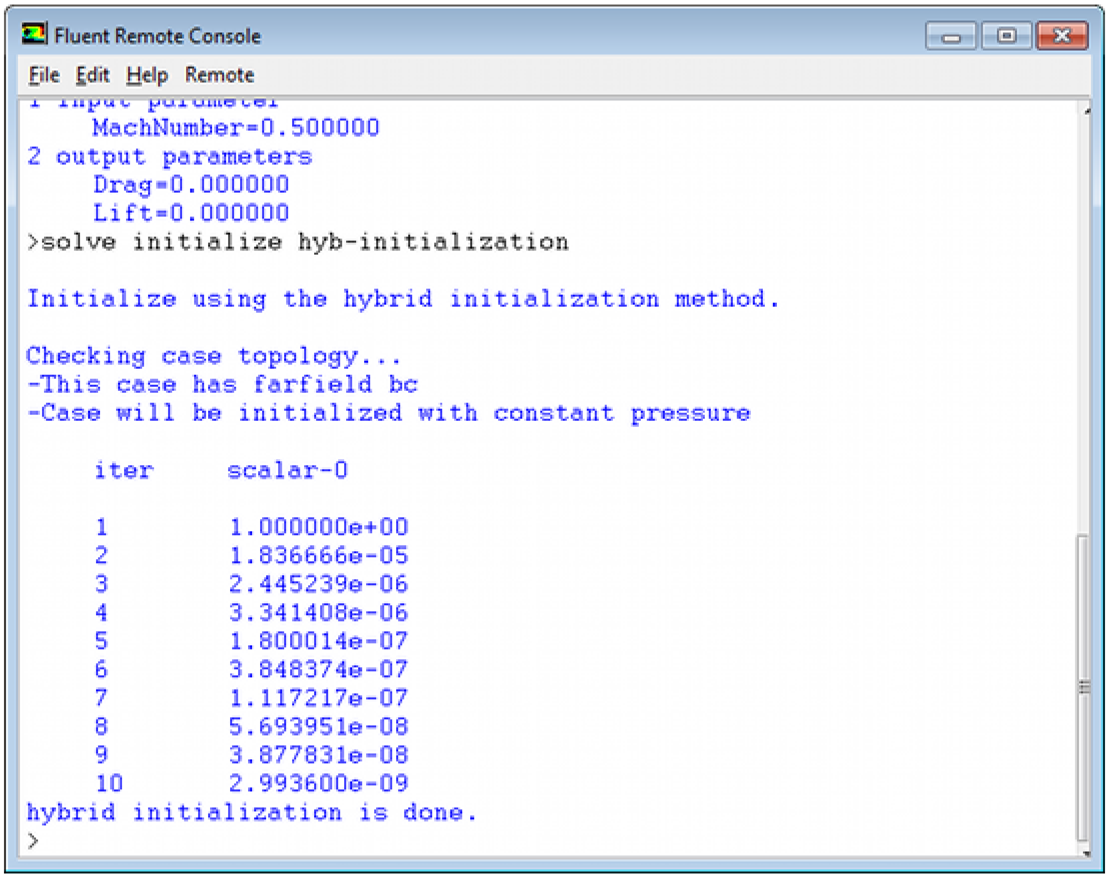
   This case has 3 parameters defined: input `MachNumber`, and outputs `Drag` and `Lift`.



```
    10075 node flags, binary.

Building...
     mesh
     materials,
     interface,
     domains,
    mixture
     zones,
    interior-1
    wall-top
    pressure-far-field-1
    wall-bottom
    fluid-16
Done.

Preparing mesh for display...
Done.Fluent case loaded.
>fluent.list_parameters
Extracting Fluent parameters
1 input parameter
    MachNumber=0.500000
2 output parameters
    Drag=0.000000
    Lift=0.000000
>
```

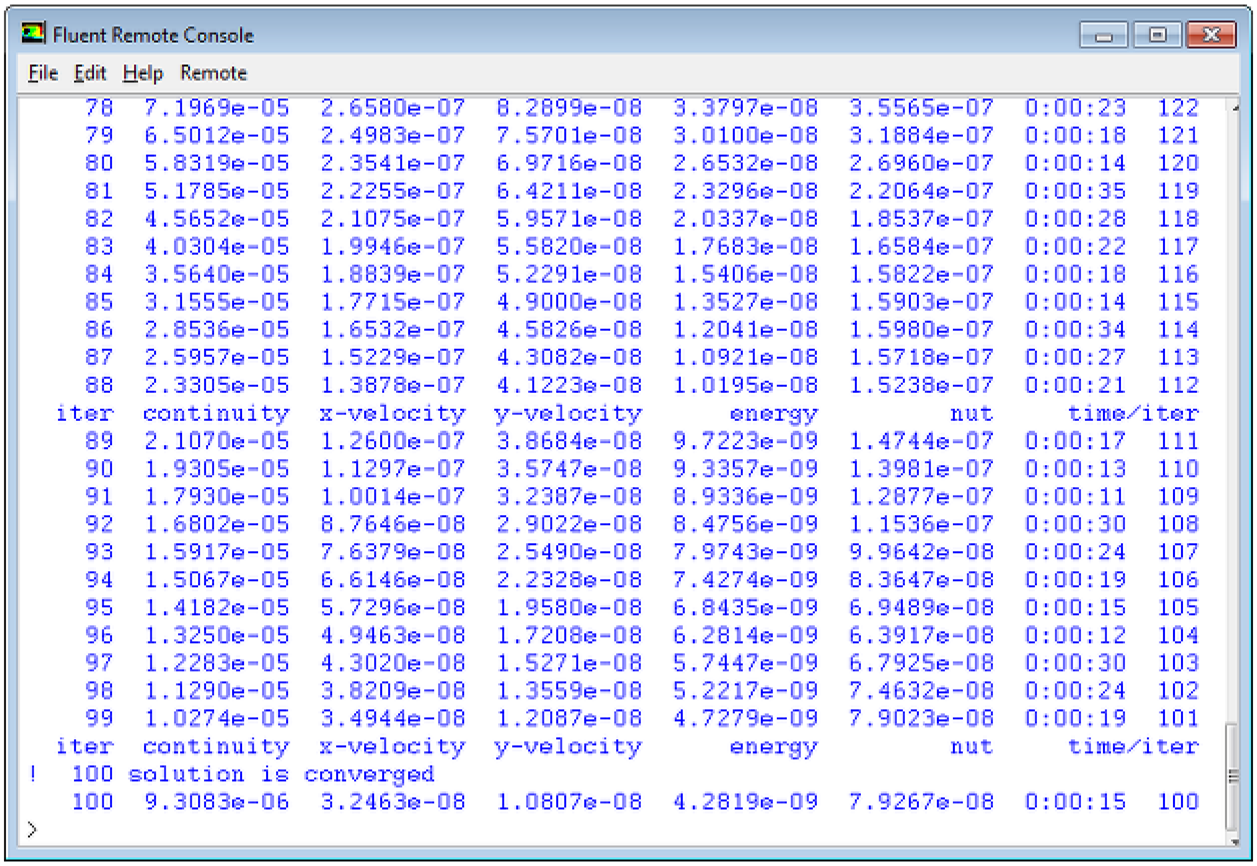
There is not yet any solution data, so `Drag` *and* `Lift` *values are reported as* `0.000000`*.*

5. Initialize the solution (TUI Command `solve initialize hyb-initialization`).



```
     MachNumber=0.500000
2 output parameters
     Drag=0.000000
     Lift=0.000000
>solve initialize hyb-initialization

Initialize using the hybrid initialization method.

Checking case topology...
-This case has farfield bc
-Case will be initialized with constant pressure

     iter      scalar-0

     1         1.000000e+00
     2         1.836666e-05
     3         2.445239e-06
     4         3.341408e-06
     5         1.800014e-07
     6         3.848374e-07
     7         1.117217e-07
     8         5.693951e-08
     9         3.877831e-08
     10        2.993600e-09
hybrid initialization is done.
>
```

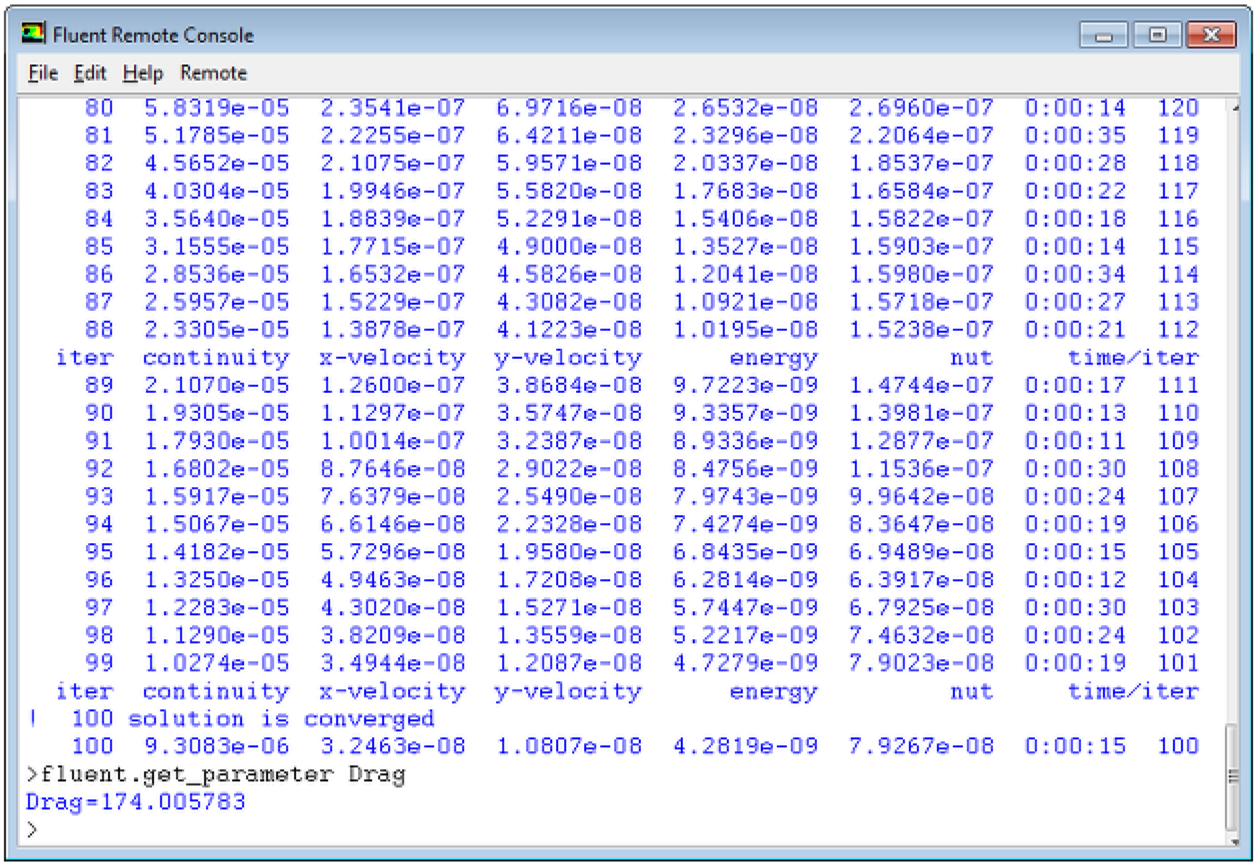
6. Iterate the solution (TUI Command `solve iterate 200`).



```
    78  7.1969e-05  2.6580e-07  8.2899e-08  3.3797e-08  3.5565e-07  0:00:23  122
    79  6.5012e-05  2.4983e-07  7.5701e-08  3.0100e-08  3.1884e-07  0:00:18  121
    80  5.8319e-05  2.3541e-07  6.9716e-08  2.6532e-08  2.6960e-07  0:00:14  120
    81  5.1785e-05  2.2255e-07  6.4211e-08  2.3296e-08  2.2064e-07  0:00:35  119
    82  4.5652e-05  2.1075e-07  5.9571e-08  2.0337e-08  1.8537e-07  0:00:28  118
    83  4.0304e-05  1.9946e-07  5.5820e-08  1.7683e-08  1.6584e-07  0:00:22  117
    84  3.5640e-05  1.8839e-07  5.2291e-08  1.5406e-08  1.5822e-07  0:00:18  116
    85  3.1555e-05  1.7715e-07  4.9000e-08  1.3527e-08  1.5903e-07  0:00:14  115
    86  2.8536e-05  1.6532e-07  4.5826e-08  1.2041e-08  1.5980e-07  0:00:34  114
    87  2.5957e-05  1.5229e-07  4.3082e-08  1.0921e-08  1.5718e-07  0:00:27  113
    88  2.3305e-05  1.3878e-07  4.1223e-08  1.0195e-08  1.5238e-07  0:00:21  112
  iter  continuity  x-velocity  y-velocity      energy         nut    time/iter
    89  2.1070e-05  1.2600e-07  3.8684e-08  9.7223e-09  1.4744e-07  0:00:17  111
    90  1.9305e-05  1.1297e-07  3.5747e-08  9.3357e-09  1.3981e-07  0:00:13  110
    91  1.7930e-05  1.0014e-07  3.2387e-08  8.9336e-09  1.2877e-07  0:00:11  109
    92  1.6802e-05  8.7646e-08  2.9022e-08  8.4756e-09  1.1536e-07  0:00:30  108
    93  1.5917e-05  7.6379e-08  2.5490e-08  7.9743e-09  9.9642e-08  0:00:24  107
    94  1.5067e-05  6.6146e-08  2.2328e-08  7.4274e-09  8.3647e-08  0:00:19  106
    95  1.4182e-05  5.7296e-08  1.9580e-08  6.8435e-09  6.9489e-08  0:00:15  105
    96  1.3250e-05  4.9463e-08  1.7208e-08  6.2814e-09  6.3917e-08  0:00:12  104
    97  1.2283e-05  4.3020e-08  1.5271e-08  5.7447e-09  6.7925e-08  0:00:30  103
    98  1.1290e-05  3.8209e-08  1.3559e-08  5.2217e-09  7.4632e-08  0:00:24  102
    99  1.0274e-05  3.4944e-08  1.2087e-08  4.7279e-09  7.9023e-08  0:00:19  101
  iter  continuity  x-velocity  y-velocity      energy         nut    time/iter
!  100 solution is converged
   100  9.3083e-06  3.2463e-08  1.0807e-08  4.2819e-09  7.9267e-08  0:00:15  100
>
```

Fluent iterates until convergence is reached and prints the iteration information to the Fluent output window.

7. Get the values of `Drag` and `Lift` (`fluent.get_parameter`).

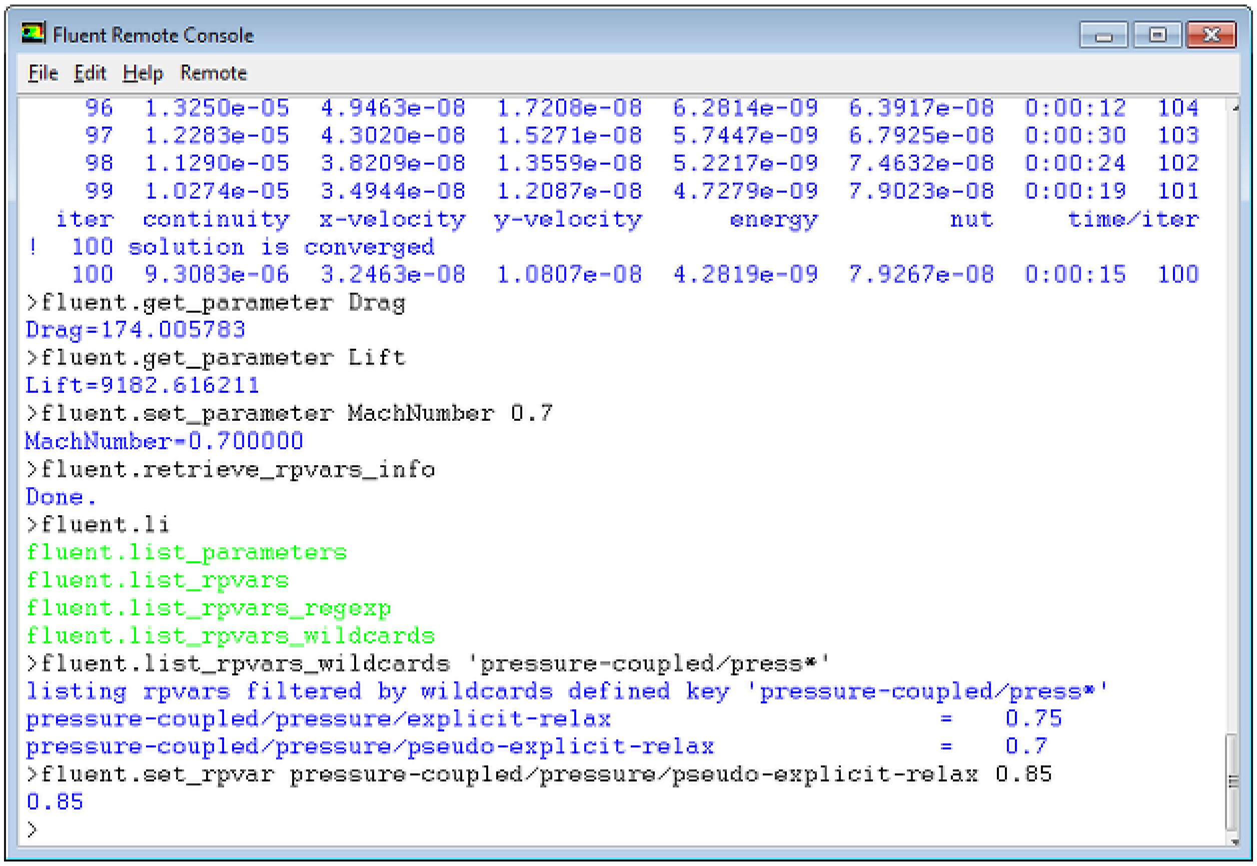


```
    80  5.8319e-05  2.3541e-07  6.9716e-08  2.6532e-08  2.6960e-07  0:00:14  120
    81  5.1785e-05  2.2255e-07  6.4211e-08  2.3296e-08  2.2064e-07  0:00:35  119
    82  4.5652e-05  2.1075e-07  5.9571e-08  2.0337e-08  1.8537e-07  0:00:28  118
    83  4.0304e-05  1.9946e-07  5.5820e-08  1.7683e-08  1.6584e-07  0:00:22  117
    84  3.5640e-05  1.8839e-07  5.2291e-08  1.5406e-08  1.5822e-07  0:00:18  116
    85  3.1555e-05  1.7715e-07  4.9000e-08  1.3527e-08  1.5903e-07  0:00:14  115
    86  2.8536e-05  1.6532e-07  4.5826e-08  1.2041e-08  1.5980e-07  0:00:34  114
    87  2.5957e-05  1.5229e-07  4.3082e-08  1.0921e-08  1.5718e-07  0:00:27  113
    88  2.3305e-05  1.3878e-07  4.1223e-08  1.0195e-08  1.5238e-07  0:00:21  112
  iter  continuity  x-velocity  y-velocity      energy         nut     time/iter
    89  2.1070e-05  1.2600e-07  3.8684e-08  9.7223e-09  1.4744e-07  0:00:17  111
    90  1.9305e-05  1.1297e-07  3.5747e-08  9.3357e-09  1.3981e-07  0:00:13  110
    91  1.7930e-05  1.0014e-07  3.2387e-08  8.9336e-09  1.2877e-07  0:00:11  109
    92  1.6802e-05  8.7646e-08  2.9022e-08  8.4756e-09  1.1536e-07  0:00:30  108
    93  1.5917e-05  7.6379e-08  2.5490e-08  7.9743e-09  9.9642e-08  0:00:24  107
    94  1.5067e-05  6.6146e-08  2.2328e-08  7.4274e-09  8.3647e-08  0:00:19  106
    95  1.4182e-05  5.7296e-08  1.9580e-08  6.8435e-09  6.9489e-08  0:00:15  105
    96  1.3250e-05  4.9463e-08  1.7208e-08  6.2814e-09  6.3917e-08  0:00:12  104
    97  1.2283e-05  4.3020e-08  1.5271e-08  5.7447e-09  6.7925e-08  0:00:30  103
    98  1.1290e-05  3.8209e-08  1.3559e-08  5.2217e-09  7.4632e-08  0:00:24  102
    99  1.0274e-05  3.4944e-08  1.2087e-08  4.7279e-09  7.9023e-08  0:00:19  101
  iter  continuity  x-velocity  y-velocity      energy         nut     time/iter
!  100 solution is converged
   100  9.3083e-06  3.2463e-08  1.0807e-08  4.2819e-09  7.9267e-08  0:00:15  100
>fluent.get_parameter Drag
Drag=174.005783
>
```

8. Change the value of MachNumber and Pressure relaxation (`fluent.set_parameter`, `fluent.retrieve_rpvars_info`, `fluent.list_rpvars_wildcards`, `fluent.set_rpvar`).

Note:

You must use `fluent.retrieve_rpvars_info` to update the available rpvars from Fluent before listing the rpvars for the first time, and any time that new rpvars are created.

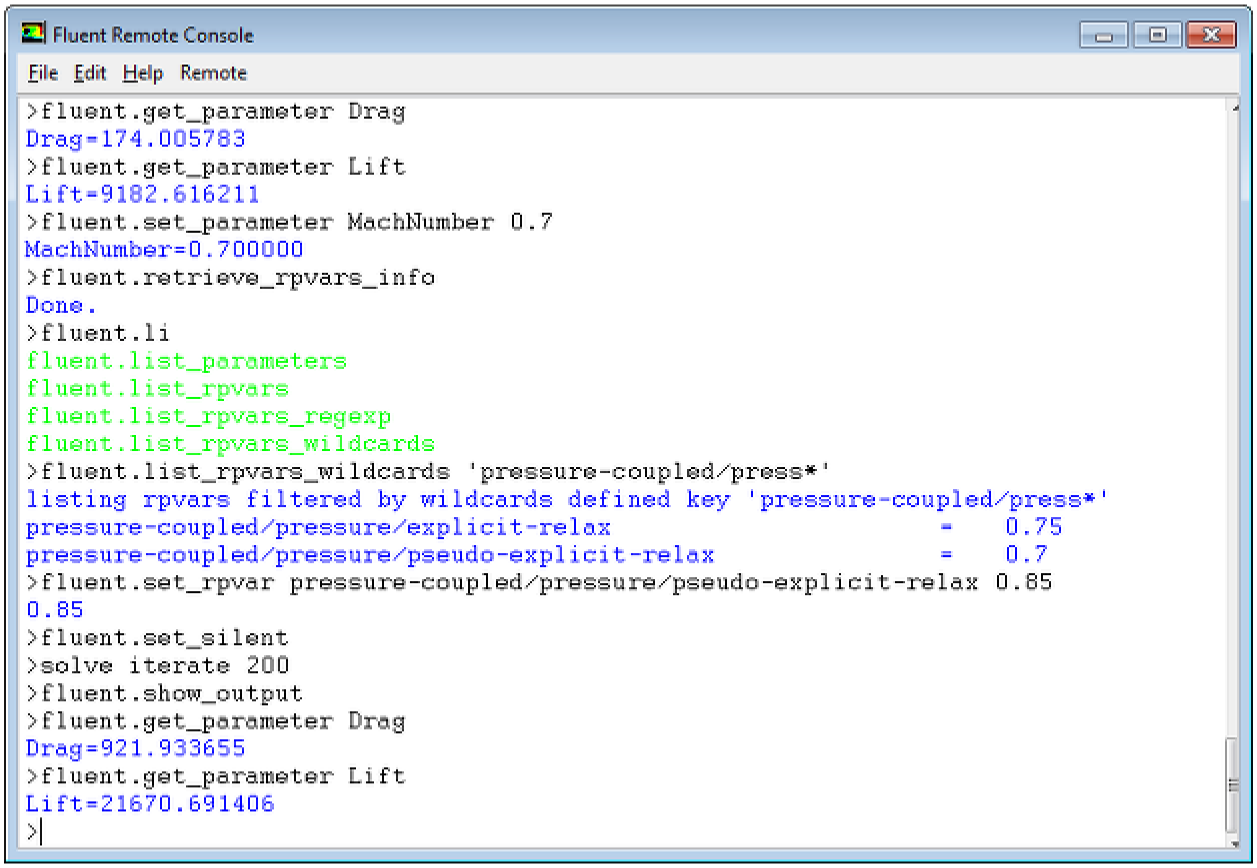


```
     96  1.3250e-05  4.9463e-08  1.7208e-08  6.2814e-09  6.3917e-08  0:00:12  104
     97  1.2283e-05  4.3020e-08  1.5271e-08  5.7447e-09  6.7925e-08  0:00:30  103
     98  1.1290e-05  3.8209e-08  1.3559e-08  5.2217e-09  7.4632e-08  0:00:24  102
     99  1.0274e-05  3.4944e-08  1.2087e-08  4.7279e-09  7.9023e-08  0:00:19  101
   iter  continuity  x-velocity  y-velocity      energy         nut     time/iter
!   100 solution is converged
    100  9.3083e-06  3.2463e-08  1.0807e-08  4.2819e-09  7.9267e-08  0:00:15  100
>fluent.get_parameter Drag
Drag=174.005783
>fluent.get_parameter Lift
Lift=9182.616211
>fluent.set_parameter MachNumber 0.7
MachNumber=0.700000
>fluent.retrieve_rpvars_info
Done.
>fluent.li
fluent.list_parameters
fluent.list_rpvars
fluent.list_rpvars_regexp
fluent.list_rpvars_wildcards
>fluent.list_rpvars_wildcards 'pressure-coupled/press*'
listing rpvars filtered by wildcards defined key 'pressure-coupled/press*'
pressure-coupled/pressure/explicit-relax                      =    0.75
pressure-coupled/pressure/pseudo-explicit-relax               =    0.7
>fluent.set_rpvar pressure-coupled/pressure/pseudo-explicit-relax 0.85
0.85
>
```

This case uses the Coupled Pressure-Based solver with Pseudo Transient relaxation, so the rpvar `pressure-coupled/pressure/pseudo-explicit-relax` is chosen.

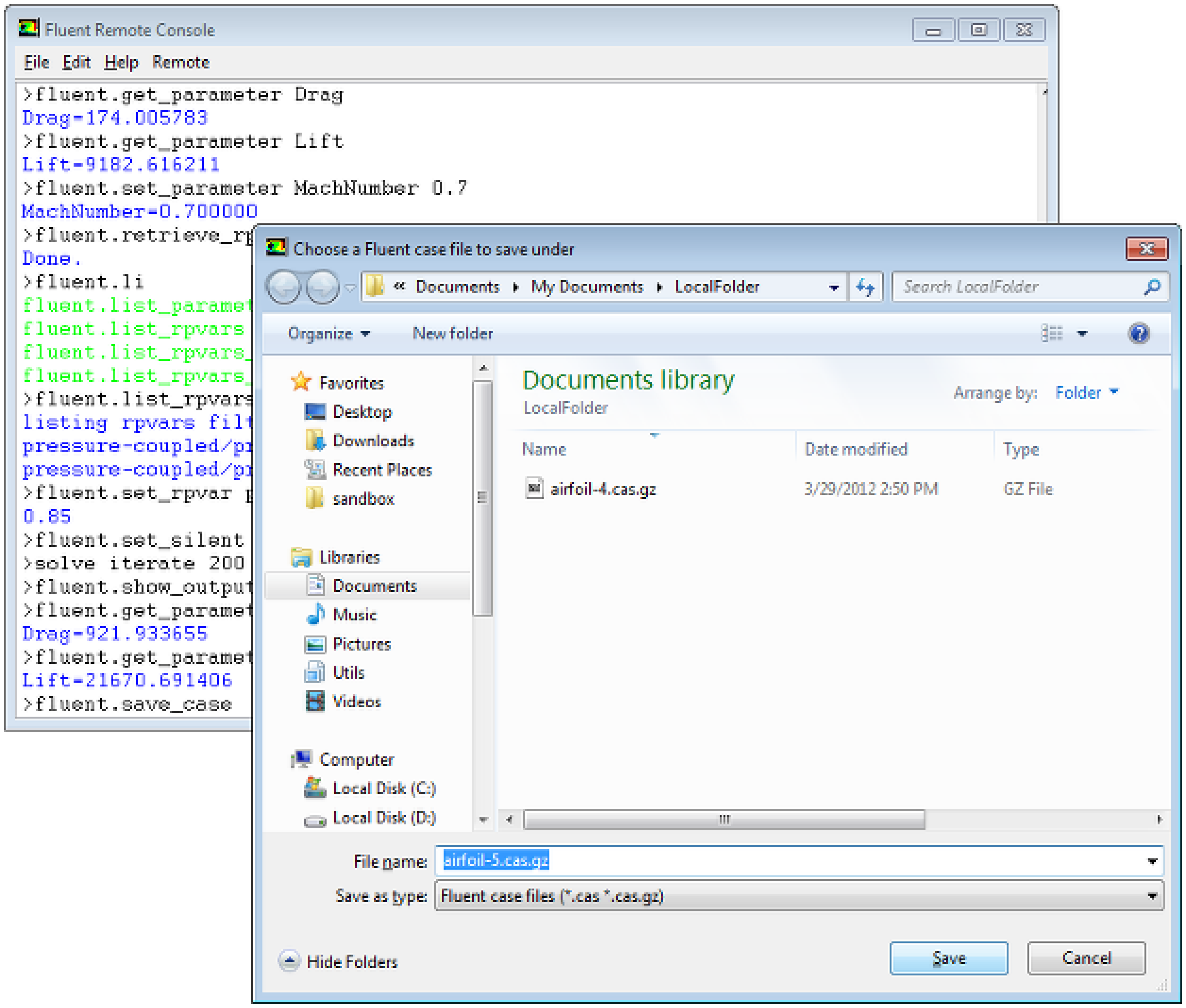
9. Direct output to the Fluent Output window and update the solution (`fluent.set_silent`, `solve iterate 200`).

10. Report the values of `Drag` and `Lift` at `MachNumber` = `0.7`.



```
>fluent.get_parameter Drag
Drag=174.005783
>fluent.get_parameter Lift
Lift=9182.616211
>fluent.set_parameter MachNumber 0.7
MachNumber=0.700000
>fluent.retrieve_rpvars_info
Done.
>fluent.li
fluent.list_parameters
fluent.list_rpvars
fluent.list_rpvars_regexp
fluent.list_rpvars_wildcards
>fluent.list_rpvars_wildcards 'pressure-coupled/press*'
listing rpvars filtered by wildcards defined key 'pressure-coupled/press*'
pressure-coupled/pressure/explicit-relax                    =    0.75
pressure-coupled/pressure/pseudo-explicit-relax             =    0.7
>fluent.set_rpvar pressure-coupled/pressure/pseudo-explicit-relax 0.85
0.85
>fluent.set_silent
>solve iterate 200
>fluent.show_output
>fluent.get_parameter Drag
Drag=921.933655
>fluent.get_parameter Lift
Lift=21670.691406
>
```

11. Save the case and data files to the client machine (`fluent.save_case`, `fluent.save_data`).



```
>fluent.get_parameter Drag
Drag=174.005783
>fluent.get_parameter Lift
Lift=9182.616211
>fluent.set_parameter MachNumber 0.7
MachNumber=0.700000
>fluent.retrieve_r
Done.
>fluent.li
fluent.list_paramet
fluent.list_rpvars
fluent.list_rpvars_
fluent.list_rpvars_
>fluent.list_rpvars
listing rpvars filt
pressure-coupled/pr
pressure-coupled/pr
>fluent.set_rpvar p
0.85
>fluent.set_silent
>solve iterate 200
>fluent.show_output
>fluent.get_paramet
Drag=921.933655
>fluent.get_paramet
Lift=21670.691406
>fluent.save_case
```

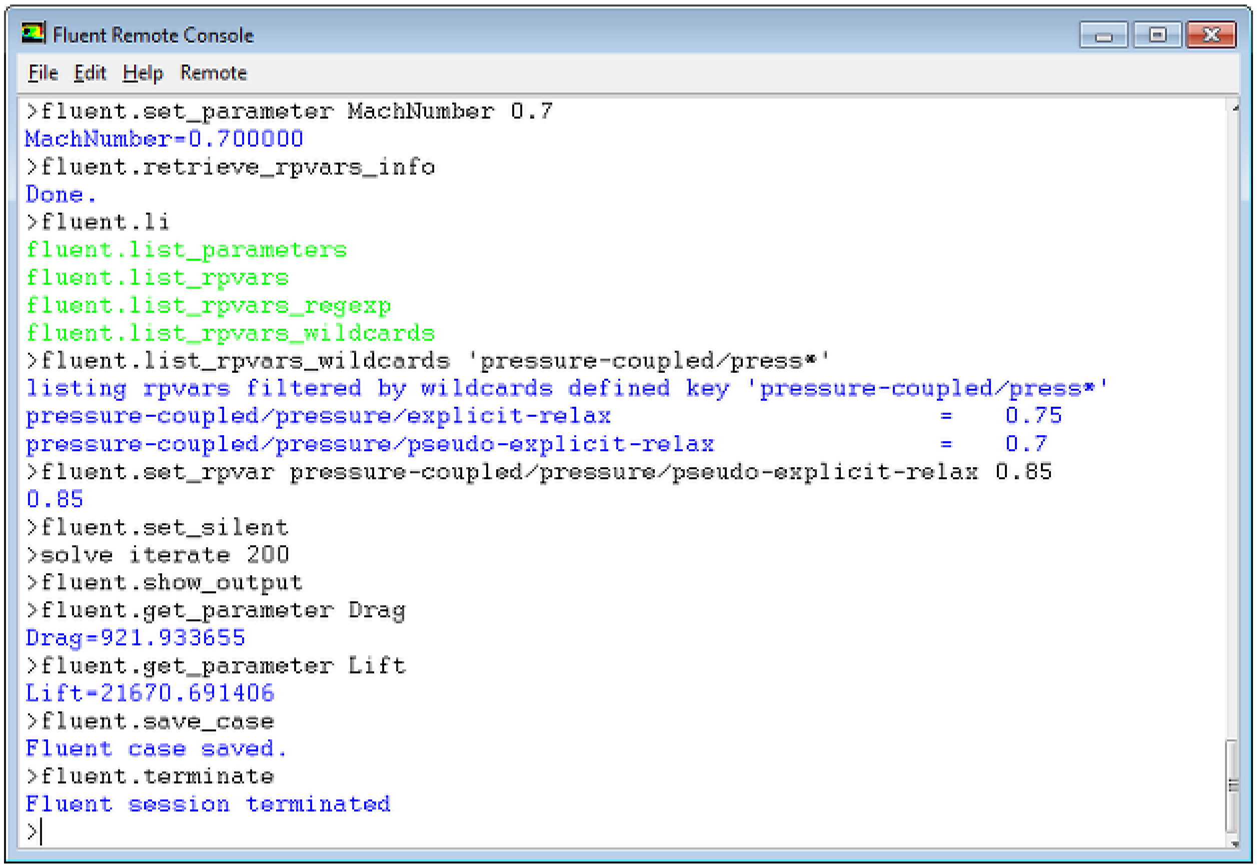
Name	Date modified	Type
airfoil-4.cas.gz	3/29/2012 2:50 PM	GZ File

File name: airfoil-5.cas.gz

Save as type: Fluent case files (*.cas *.cas.gz)

Since no filename was specified to `fluent.save_case` *a dialog box opens for you to select a location. The* `fluent.save_data` *command is performed the same way. Note that* `fluent.save_case` *and* `fluent.save_data` *allow you to save the case/data files to the local (client) machine.*

12. Terminate the Fluent session (`fluent.terminate`).



```
>fluent.set_parameter MachNumber 0.7
MachNumber=0.700000
>fluent.retrieve_rpvars_info
Done.
>fluent.li
fluent.list_parameters
fluent.list_rpvars
fluent.list_rpvars_regexp
fluent.list_rpvars_wildcards
>fluent.list_rpvars_wildcards 'pressure-coupled/press*'
listing rpvars filtered by wildcards defined key 'pressure-coupled/press*'
pressure-coupled/pressure/explicit-relax                    =    0.75
pressure-coupled/pressure/pseudo-explicit-relax             =    0.7
>fluent.set_rpvar pressure-coupled/pressure/pseudo-explicit-relax 0.85
0.85
>fluent.set_silent
>solve iterate 200
>fluent.show_output
>fluent.get_parameter Drag
Drag=921.933655
>fluent.get_parameter Lift
Lift=21670.691406
>fluent.save_case
Fluent case saved.
>fluent.terminate
Fluent session terminated
>
```

Chapter 4: Fluent as a Server Software Development Kit (SDK)

The Fluent as a Server Software Development Kit (SDK) is included with an installation of ANSYS Fluent or the Fluent as a Server Client Package and allows you to enable your own client application to connect to and communicate with Fluent as a Server sessions. Using the SDK you can extend or create a client application to provide whatever level of control over the Fluent as a Server session that your application requires.

This chapter is divided into the following sections.

4.1. Requirements

In order to use the Fluent as a Server SDK to create client applications that connect to Fluent as a Server sessions, you will need to ensure you have the following additional tools not supplied with ANSYS Fluent:

- A development environment with native or 3rd party support for Common Object Request Broker Architecture (CORBA) and a suitable CORBA IDL compiler.

Tip:

3rd-party CORBA implementations are available for a wide variety of common languages including:

C
C++
Python
Java

OR

- A development environment based on .NET Framework version 3.5 or 4.0.

4.2. Fluent as a Server CORBA Interfaces

The central element of the Fluent as a Server SDK is a pair of CORBA interfaces that you can use in building a client application. These are included with an installation of Fluent.

The following sections describe the interfaces and their use:

4.2.1. ICoFluentUnit

The `ICoFluentUnit` interface provides a set of functions that perform typical commands in ANSYS Fluent for solving a CFD simulation case. For example you can load or save case and data files, set the number of computational iterations, initiate a calculation, get or set defined parameter values, and so on. An example of a simple client that uses the interface is given in A Fluent Client Example (p. 60).

interface `ICoFluentUnit`

string `getComponentName`();
returns the name of the connected component

void `setComponentName`(in string *`p_szName`*);
sets the name of the component to *`p_szName`*

string `getComponentDescription`();
returns the description of the connected component

void `setComponentDescription`(in string *`p_szDescription`*);
sets the description of the connected component to *`p_szDescription`*

void `calculate`();
iterates the solution for the number of iterations specified with `setNrIterations`. This is mainly for use with steady simulations as it does not perform dual-time iteration. For transient cases you can issue the solve/dual-time-iterate TUI command using doMenuCommand (see ICoFluentSchemeController (p. 51))

void `setNrIterations`(in long *`p_lNrIterations`*);
sets the number of iterations that `calculate` will perform to *`p_lNrIterations`*.

long `getNrIterations`();
returns the number of iterations currently set for a `calculate` command to perform

void `loadCase`(in string *`p_szCaseFileName`*);
load the case file *`p_szCaseFileName`* from the Fluent working directory into Fluent

void `loadData`(in string *`p_szDataFileName`*);
load the data file *`p_szDataFileName`* from the Fluent working directory into Fluent

void `saveCase`(in string *`p_szCaseFileName`*);
save the current Fluent case to *`p_szCaseFileName`* in the Fluent working directory

void `saveData`(in string *`p_szDataFileName`*);
save the current Fluent data to *`p_szDataFileName`* in the Fluent working directory

long `getNrInputParameters`();
returns the number of input parameters defined in the current case

string `getInputParameterNameByIndex`(in long *`lInputParameterIndex`*);
returns a string containing the name of the input parameter with index *`lInputParameterIndex`*

void `setInputParameterValueByIndex`(in long *`p_lInputParameterIndex`*, in float *`p_lfInputParameterValue`*);
sets the value of the input parameter with index *`p_lInputParameterIndex`* to *`p_lfInput-ParameterValue`*

void `setInputParameterValueByName`(in string *`p_lInputParameterName`*, in float *`p_lfInputParameterValue`*);
sets the value of the input parameter with name *`p_lInputParameterName`* to *`p_lfInput-ParameterValue`*

long `getNrOutputParameters`();
returns the number of output parameters defined in the current case

string `getOutputParameterNameByIndex`(in long *`lOutputParameterIndex`*);
returns a string containing the name of the output parameter with index *`IOutputParameter-Index`*

float `getOutputParameterValueByIndex`(in long *`p_lOutputParameterIndex`*);
returns the value of the output parameter with index *`IOutputParameterIndex`*

float `getOutputParameterValueByName`(in string *`p_lOutputParameterName`*);
returns the value of the output parameter with name *`p_IOutputParameterName`*

void `terminate`();
terminate the connected Fluent as a Server session

Object `getSchemeControllerInstance`();
returns an object that can be used to send TUI or scheme commands to the Fluent session and perform more advanced functions using the ICoFluentSchemeController Interface (see ICoFluentS-chemeController (p. 51))

4.2.2. ICoFluentSchemeController

The ICoSchemeController interface includes functions for more advanced control of a Fluent session, such as issuing direct Scheme or menu commands and manipulating rpvars. An example of a simple client that uses the interface is given in A Fluent Client Example (p. 60).

ICoFluentSchemeController

void `execScheme`(in string *`p_szSchemeCommand`*);
: issues a scheme command to the connected Fluent session. Output from the command is not returned.

string `execSchemeToString`(in string *`p_szSchemeCommand`*);
: issues a scheme command to the connected Fluent session and returns the output as a string.

void `doMenuCommand`(in string *`p_szMenuCommand`*);
: issues a TUI or aaS command to the connected Fluent session (for a list of available commands, see Fluent Text Command List and aaS Commands (`aaS.`) (p. 53)). Output from the command is not returned.

string `doMenuCommandToString`(in string *`p_szMenuCommand`*);
: issues a TUI or aaS command to the connected Fluent session and returns the output as a string. For a list of available commands, see Fluent Text Command List and aaS Commands (`aaS.`) (p. 53).

void `setRpVar`(in string *`p_szRpVar`*, in string *`p_szRpVarValue`*);
: sets the value of the rpvar *`p_szRpVar`* to *`p_szRpVarValue`*.

string `getRpVar`(in string *`p_szRpVar`*);
: returns a string with the value of the rpvar *`p_szRpVar`*.

void `uploadFileFromBuffer`(in string *`p_szFileName`*, in CoOctetBuffer *`p_pFileContent`*);
: writes a file named *`p_szFileName`* in the remote Fluent session working directory with the contents of *`p_pFileContent`*. If *`p_szFileName`* exists, it is overwritten.

CoOctetBuffer `downloadFileFromBuffer`(in string *`p_szFileName`*);
: returns the contents of the file named *`p_szFileName`* in the remote Fluent session working directory.

4.2.3. Exceptions

Interactive Prompting Exceptions

The following exception structures are defined to handle the various interactive questions that Fluent may respond with if an incomplete command is issued using `doMenuCommand()` or `doMenuCommandToString()`. For a description of how to use these in your client application, see Using Interactive Prompting (p. 58).

```
exception EYesNoQuestion
{
 long  questionType;                        # An integer for the type of question:
                                            # 1 for this type
 string questionPromptWithDefaultAnswer;    # The prompt as it would be given in Fluent
                                            # with the default given in square brackets
 string defaultAnswer;                      # The default answer by itself
 string questionHelp;                       # A help message if applicable
 string questionMessage;                    # The message Fluent would present if the answer
                                            # given was invalid
}
```

```
exception EReadUnquotedString
{
 long  questionType;                  # An integer for the type of question:
                                      # 2 for this type
 string questionPrompt;               # The prompt as it would be given in Fluent
                                      # (without the default answer)
 string defaultAnswer;                # The default answer
 string questionHelp;                 # A help message if applicable
 string questionMessage;              # The message Fluent would present
                                      # if the answer given was invalid
};
```

```
exception EReadQGenericQuestion
{
 long  questionType;                  # An integer for the type of question:
                                      # 3 for this type
 string questionPromptWithDefaultAnswer;   # The prompt as it would be given in Fluent
                                      # with the default given in square brackets
 string rawDefaultAnswer;             # The default answer
 string questionHelp;                 # A help message if applicable
 string questionMessage;              # The message Fluent would present
                                      # if the answer given was invalid
};
```

```
exception EMenuGetQuestion
{
 long  questionType;                  # An integer for the type of question:
                                      # 4 for this type
 string questionPrompt;               # The prompt as it would be given in Fluent
                                      # with the default given in square brackets
 string questionHelp;                 # A help message if applicable
 string questionMenu;                 # A list of items that can be chosen from
};
```

Error Handling

```
exception EFluentGenericError
{
 long  errorType;                     # The type of error being returned
 string questionPrompt;               # A prompt for user input
 string questionHelp;                 # A help message if applicable
 string questionMenu;                 # A list of items that can be chosen from
};
```

4.2.4. aaS Commands (`aaS.`)

aaS commands are used to perform client/server interactions with a connected Fluent as a Server session. These include commands to set/get rpvar values, alter verbosity, and so on. aaS commands begin with the prefix `aaS`.

The aaS commands are described in the following sections, and are grouped based on whether they can provide an “instant” response (because they only require existing data) or whether they must “collaborate” with Fluent processes and therefore may require a waiting period to respond. Note that instant commands can be called from a collaborative object, but will not be executed until Fluent is idle or at a stable point.

4.2.4.1. Instant Commands

The following commands can always provide an instant response, as they use existing data and can be executed even if Fluent is not currently idle or at a stable point.

aaS.get_current_flow_time
provides the current flow time.

aaS.get_current_iteration
provides the current iteration.

aaS.get_current_time_step
provides the current time step.

aaS.get_dimensionality
provides the dimensionality of the current case.

aaS.get_last_casefile_name
provides the name of the last case file accessed before the current case. This can be useful if you have written the case file with a new name.

aaS.get_list_input_parameters
provides a list of the input parameters available in the current case.

aaS.get_list_output_parameters
provides a list of the output parameters available in the current case.

aaS.get_nr_input_parameters
provides the number of input parameters available in the current case.

aaS.get_nr_output_parameters
provides the number of output parameters available in the current case.

aaS.get_paused_index
returns an index indicating the “depth” of pause. When Fluent is not paused, it will return `0`. It is incremented with each successive pause instruction. This can be useful to determine the state of a session if the connection is lost while Fluent is paused.

aaS.get_precision
specifies whether the solution calculations for the current case will be performed in single- or double-precision mode.

aaS.get_release
provides the name of the solver and the release number.

aaS.get_status
provides information about the status of the Fluent session (for example, `journaling`, `iterate`).

aaS.list_monitors <monitor_type> <monitor_name>
returns a JSON-formatted listing of the monitor definitions in the current case. **`<monitor_type>`** may be one of the following types: `residuals`, `lift`, `drag`, `moment`, `surface`, or `volume`. If no arguments are given, Fluent returns all defined monitors. If only **`<monitor_type>`** is given, all monitors of the specified type are returned.

aaS.list_monitors_with_persisted_data <monitor_type> <monitor_name> <filter> <filter_value>
returns a JSON-formatted listing of the monitor definitions in the current case, including the data from the monitor history file (if any). Options for **`<monitor_type>`** are the same as those listed for `aaS.list_monitors`. **`<filter>`** allows you to return incremental monitor data, by removing the data calculated before the specified **`<filter_value>`**; a single command can include one or more of the following **`<filter>`** options: `start-flow-time`, `start-iteration`, and `start-time-step`.

aaS.list_report_definitions
returns a JSON-formatted listing of the report definitions in the current case. For more information about report definitions, see Creating Report Definitions in the *Fluent User's Guide*.

aaS.list_report_files
returns a JSON-formatted listing of the report file definitions in the current case. If no arguments are given, Fluent returns all defined report files. For more information about report files, see Report Files and Report Plots in the *Fluent User's Guide*.

aaS.list_report_files_with_persisted_data <filter> <filter_value>
returns a JSON-formatted listing of the report file definitions in the current case, including the data from the report file `.out` file (if any). For more information about report files, see Report Files and Report Plots in the *Fluent User's Guide*. **`<filter>`** allows you to return incremental report file data, by removing the data calculated before the specified **`<filter_value>`**; a single command can include one or more of the following **`<filter>`** options: `start-iteration` and `start-time-step`.

aaS.list_report_plots
returns a JSON-formatted listing of the report plot definitions in the current case. For more information about report plots, see Report Files and Report Plots in the *Fluent User's Guide*.

aaS.save_monitors_with_persisted_data <monitor_file_name>
saves a JSON-formatted file on the remote Fluent machine with the data (if it exists) from all monitors defined in the current case.

aaS.set_auto_save_period_json_monitors <period_in_minutes>
allows you to change the period for when monitor history data is automatically saved as a JSON-formatted file (named `monitor.json`) on the remote Fluent machine. By default, the period is set to 60 minutes. Note that it is recommended that you do not have multiple sessions in a single folder, as they will all have the same name for the monitor history file.

4.2.4.2. Collaborative Commands

The following commands collaborate with Fluent processes and therefore cannot be executed until Fluent is idle or at a stable point. This means that you are not guaranteed to have an instant response when you enter the command.

Tip:

The following strategies can be used to minimize the lag between when you enter a command and when it is executed:

- Set the `aaS.set_aaslistening_step_at <number> iteration|timestep` command to a minimum value, based on your objectives. Note that by decreasing the iterations / timesteps between stable points, you are favoring cosimulation over simulation (that is, you are prioritizing your ability to manipulate the settings over the time it will take to calculate the solution).
- You can nest a number of commands between `aaS.pause` and `aaS.continue` commands, so that they are all executed during the same stable point; otherwise, each command will have to wait for a separate stable point.
- For `aaS.` commands that have an equivalent `fluent.` command (see Fluent as a Server Commands (`fluent.`) (p. 28)), it may be preferable to use the latter: `fluent.` commands undergo lexical and semantic checks that ensure you are not needlessly waiting on a poorly formed command, and allow you to use the **Tab** key to automatically complete partially typed commands for the sake of convenience and accuracy.

---

aaS.continue
continues remote execution of a journal file on the Fluent server.

aaS.get_aaslistening_step
provides information about the aaslistener that listens for commands from Fluent as a Server clients.

aaS.interrupt
interrupts remote execution of a journal file on the Fluent server. Note that this terminates reading of the journal file. If you want to be able to resume execution where it was stopped, use `aaS.pause`.

aaS.pause
pauses remote execution of a journal file on the Fluent server. Execution can be resumed using the `aaS.continue` command.

aaS.set_aaslistening_step_at <number> iteration|timestep
configures the aaslistener to listen for commands from Fluent as a Server clients every **`<number>`** iterations or timesteps.

aaS.read_journal
loads a journal and executes it asynchronously (control is immediately returned to the user). The status of the journal execution can be checked using `aaS.get_status`.

4.2.5. Using the CORBA Interfaces

There are two ways to include the CORBA interfaces in your client application depending on your needs and development platform.

Building From Interface Definition Language (IDL) Files

You can build the interface modules for any CORBA-enabled language by compiling the supplied Interface Definition Language (IDL) file using a suitable 3rd-party CORBA compiler. The IDL file is provided in the following location in the ANSYS install tree:

```
v202\fluent\fluent20.2.0\addons\corba\<ARCH>\CoFluentUnit.idl
```

Refer to the documentation for your CORBA implementation and development environment for details on how to compile this IDL file and include the resulting modules in your application.

Using the Pre-Built .NET Connector

If you are using .NET for your client application, you can load a pre-built DLL that contains wrappers for the CORBA interfaces. Two DLLs are available, depending on your version of .NET.

```
v202\fluent\fluent20.2.0\addons\corba\DotNetFramework35\DotNetCoFluen-
tUnit.dll
v202\fluent\fluent20.2.0\addons\corba\DotNetFramework40\DotNetCoFluen-
tUnit.dll
```

Including `DotNetCoFluentUnit.dll` makes available the following classes in the AAS_CORBA namespace.

`DotNetCoFluentUnit`
has as methods wrappers for each of the functions listed in ICoFluentUnit (p. 50).

The `DotNetCoFluentUnit` class also makes available two additional methods.

void `ConnectToServerFromIorFile`(string *`p_stringIorFile`*);
takes as input the name of a Fluent as a Server key file and uses it to instantiate a CORBA connection to the Fluent server.

DotNetCoFluentSchemeController `getDotNetCoFluentSchemeControllerInstance`();
returns a Scheme controller object that makes available (through wrappers) the methods listed in ICoFluentSchemeController (p. 51). When using the .NET connector, this method should be used rather than directly using the getSchemeControllerInstance method as it handles typing that may otherwise not be handled correctly.

`DotNetCoFluentSchemeController`
has as methods wrappers for the functions listed in ICoFluentSchemeController (p. 51).

`DotNetCoFluentUnitError`
defines an exception for errors returned by the .NET wrappers.

```
DotNetCoFluentUnitError
{
 long long m_iCode;
 string m_stringDescription;
 string m_stringInterfaceName;
 string m_stringMoreInfo;
 string m_stringName;
 string m_stringOperation;
 string m_stringScope
};
```

4.2.6. Using Interactive Prompting

Normally, when your client application issues a TUI / aaS command to Fluent using `doMenuCommand()` or `doMenuCommandToString()` Fluent will use default values for any unspecified arguments. You can instruct Fluent to instead prompt for missing arguments by appending the character sequence '`<space>-?`' to the command string. For example, the following line would send the `report summary` command to Fluent through the CORBA scheme interface and request that Fluent prompt with questions:

```
fluentSchemeInterface.doMenuCommandToString("report summary -?")
```

If Fluent requires additional information to execute the command it will throw one of the CORBA exceptions in Exceptions (p. 52) depending on the type of question. The client can catch the exception and process it as necessary to respond with the additional information.

Thus, a typical sequence for a client application conducting a dialog with Fluent proceeds as follows:

1. Client sends a command to Fluent with the '`<space>-?`' sequence appended.
2. Fluent checks whether an argument is required and if so returns a CORBA exception.
3. Client catches the exception and determines how to respond to the question from Fluent.
4. Client reformulates the command with the answer to the question now appended as an argument (along with '`<space>-?`') and sends the new command to Fluent.
5. Fluent checks whether an additional argument is required and, if so, returns a CORBA exception.
6. Steps 3–5 are repeated until Fluent determines that it has a complete command.

Example Code Block

The following is a sample Python code block showing a basic implementation of exception handling to receive prompts from a remote Fluent as a Server session and present them to a user operating the client application. For each prompt with a default answer, if the user presses **Enter** without entering any text, the default value is used.

```
success=False
while not success:
 try:
  output=scheme_interface.doMenuCommandToString(''.join([command," -?"]))
  success=True
 except AAS_CORBA.EYesNoQuestion as e:
  moreinfo=raw_input(''.join([e.questionPromptWithDefaultAnswer," "]))
  if not moreinfo:
   moreinfo=e.defaultAnswer
  command=''.join([command," ",moreinfo])
 except AAS_CORBA.EReadUnquotedString as e:
  moreinfo=raw_input(''.join([e.questionPrompt," ","[",e.defaultAnswer,"]"]))
  if not moreinfo:
   moreinfo=e.defaultAnswer
  command=''.join([command," ",moreinfo])
 except AAS_CORBA.EReadQGenericQuestion as e:
  moreinfo=raw_input(''.join([e.questionPromptWithDefaultAnswer," "]))
  if not moreinfo:
   moreinfo=e.rawDefaultAnswer
  command=''.join([command," ",moreinfo])
 except AAS_CORBA.EMenuGetQuestion as e:
  print(e.questionMenu)
  moreinfo=raw_input(''.join([e.questionPrompt," "]))
  command=''.join([command," ",moreinfo])
 except:
  e=sys.exc_info()
  print( "Error: %s" % e[0])
  print(e[1])
  sys.exit(1)
```

4.3. Procedure for a Creating a Simple Fluent as a Server Client Application

The basic steps to create or enable an application to connect to Fluent as a Server are:

Using a CORBA-Aware Language

1. Install a CORBA-compliant ORB implementation suitable for the development environment in which you will be creating your client application.

2. Use the compiler included with your CORBA implementation to compile the Fluent as a Server interface modules from the Interface Definition Language (IDL) file. The IDL file is located in the following location in the ANSYS install tree.

   `v202\fluent\fluent20.2.0\addons\corba\`**`<ARCH>`**`\CoFluentUnit.idl`

3. Create your client project and include the AAS_CORBA module created in the previous step. Refer to Fluent as a Server CORBA Interfaces (p. 50) for details about the interfaces contained in the AAS_CORBA module.

   The specific source code to use the provided interfaces will depend on your development language and requirements. In general, a useful client application will:

   a. instantiate a CORBA object

   b. pass the IOR string contained in the Fluent as a Server session's keyfile to an ORB method to create an ICoFluentUnit object

   c. use the methods provided in ICoFluentUnit and/or ICoFluentSchemeController to load a case file, send commands, or retrieve output from the remote Fluent session.

      See Fluent as a Server CORBA Interfaces (p. 50) for details about these interfaces and the provided methods.

4. Start a Fluent as a Server session on a remote machine that is accessible from the client machine (see Fluent as a Server Session Management (p. 17)).

5. Use the keyfile created by the remote Fluent as a Server session to connect your client application to Fluent.

Using .NET Framework

1. Create your client project and include `DotNetCoFluentUnit.dll`. Refer to Using the Pre-Built .NET Connector (p. 57) for details.

   The specific source code to use the provided interfaces will depend on your development language and requirements. In general, a useful client application will:

   a. Create a DotNetCoFluentUnit object.

   ```
   dotNetCoFluentUnit=DotNetCoFluentUnit()
   dotNetCoFluentUnit.ConnectToServerFromIorFile('aaS_FluentId.txt')
   ```

   b. use the methods provided in `DotNetCoFluentUnit` and `DotNetCoFluentSchemeController` to load a case file, send commands, or retrieve output from the remote Fluent session.

      See Using the Pre-Built .NET Connector (p. 57) for details about these classes the provided methods.

2. Start a Fluent as a Server session on a remote machine that is accessible from the client machine (see Fluent as a Server Session Management (p. 17)).

3. Use the keyfile created by the remote Fluent as a Server session to connect your client application to Fluent.

4.4. A Fluent Client Example

The following example demonstrates the procedure to create and use a simple client program. While very basic, this example illustrates the essential elements of a client application and some of the fundamental capabilities offered by the SDK, including:

- compiling the IDL file into an includable module.
- creating a CORBA object and using the IOR string in a keyfile named `aaS_FluentId.txt` to connect to a remote session.
- obtaining a scheme controller object from the Fluent as a Server session.
- loading a case file from the client machine into the Fluent as a Server session.
- send TUI commands to the remote session and retrieve the output.
- download files from the remote Fluent as a Server machine to the client machine.
- Exit when no more files are requested. The remote Fluent as a Server session is left running.

For reference, the example is performed on a 64–bit Windows 7 and makes use of the following 3rd party software:

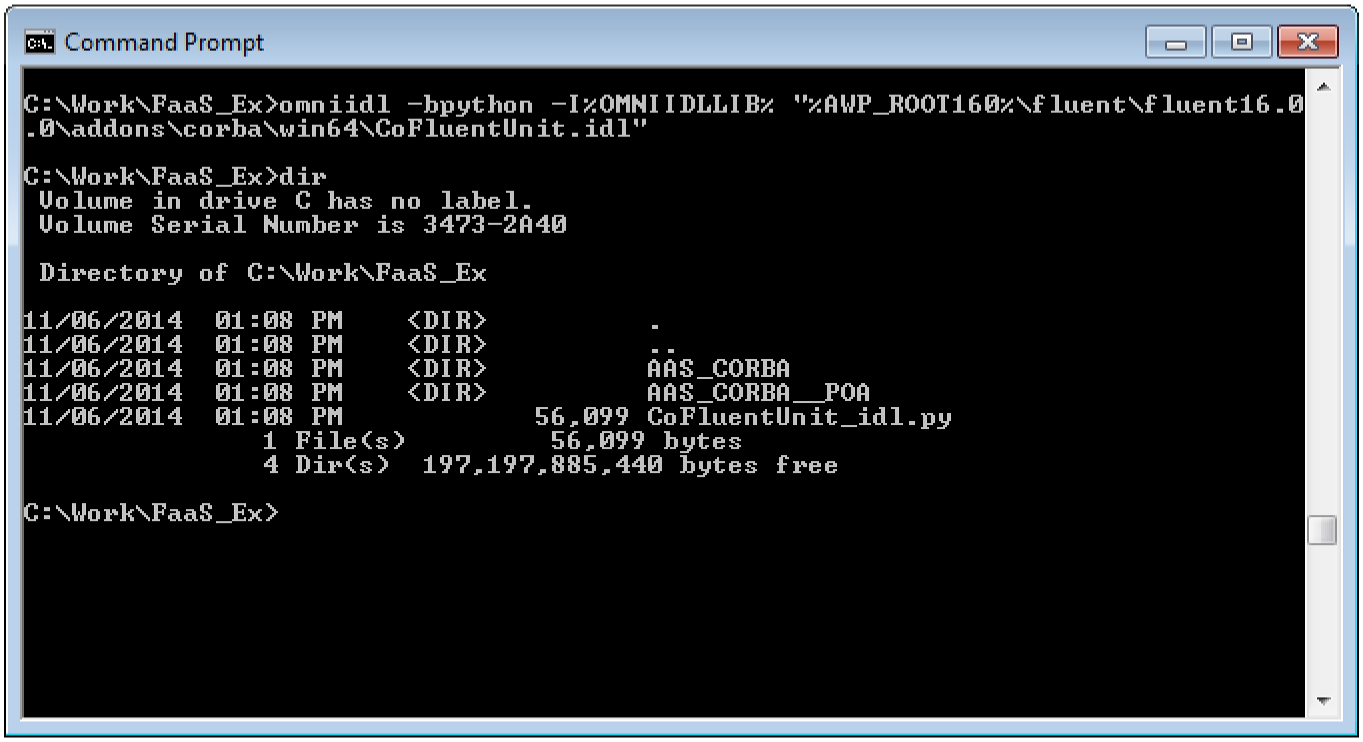
- Python 2.6 64–bit
- omniORBpy 3.5

4.4.1. Procedure

1. Compile `CoFluentUnit.idl` to create the AAS_CORBA module using the IDL compiler from the 3rd party CORBA implementation.

   Refer to the documentation for your CORBA compiler for details about how to compile the IDL file.

   ```
   >omniidl -bpython -I%OMNIIDLLIB% "%AWP_ROOT202%\fluent\fluent20.2.0\addons\corba\win64\CoFluentUnit.idl"
   ```



```
C:\Work\FaaS_Ex>omniidl -bpython -I%OMNIIDLLIB% "%AWP_ROOT160%\fluent\fluent16.0
.0\addons\corba\win64\CoFluentUnit.idl"

C:\Work\FaaS_Ex>dir
 Volume in drive C has no label.
 Volume Serial Number is 3473-2A40

 Directory of C:\Work\FaaS_Ex

11/06/2014  01:08 PM    <DIR>          .
11/06/2014  01:08 PM    <DIR>          ..
11/06/2014  01:08 PM    <DIR>          AAS_CORBA
11/06/2014  01:08 PM    <DIR>          AAS_CORBA__POA
11/06/2014  01:08 PM            56,099 CoFluentUnit_idl.py
               1 File(s)         56,099 bytes
               4 Dir(s)  197,197,885,440 bytes free

C:\Work\FaaS_Ex>
```

2. Create the client application, `client_ex.py`. This very simple example will perform the following tasks:

   - Prompt the user for the location of an aaS_FluentId.txt file with connection information a remote Fluent as a Server session.
   - Create a CORBA object that will be used to connect to the remote session.
   - Obtain a scheme controller object from the Fluent as a Server session.
   - Prompt the user for the name of a case file on the client machine to upload and read into Fluent.
   - Allow the user to interactively send TUI commands to the remote session until they indicate they are finished

- Ask the user for the names of any files on the remote Fluent as a Server machine they would like to download. These could include graphics or xy plots, case/data files, etc.

- Exit when no more files are requested. The remote Fluent as a Server session is left running.

Following are the major code blocks of the example program with the use of the SDK interface methods highlighted. The complete code listing appears at the end of the example (Code Listing (p. 68)).

a. Import AAS_CORBA and any other required modules.

```
# client_ex.py
# A simple example of a &pn091g client application
import os
import io
import CORBA
import AAS_CORBA    #import the AAS_CORBA module compiled from CoFluentUnit.idl

from functools import partial
```

b. Prompt the user for an aaS_FluentId.txt file that points to a Fluent as a Server session and extract the IOR string.

```
# Request that the user supply an aaS_FluentId.txt file with connection
# details for a Fluent session
iorstring=''

while iorstring=='':
    filename=str(raw_input('Please provide an aaS_FluentId.txt file:\n> '))
    try:
        filehdl=open(filename,'r')
    except IOError:
        print(filename+" doesn't appear to exist!")
        continue

    if filehdl.read(3)!="IOR":   #if the file doesn't begin with IOR
        print(filename+" doesn't appear to contain a valid IOR string!")
        continue
    else:
        filehdl.seek(0)
        iorstring=filehdl.read()
    filehdl.close()
```

c. Instantiate the ORB, connect to the Fluent as a Server session, and get a scheme interface object using `ICoFluentUnit.getSchemeControllerInstance()`.

```
# Connect to the Fluent session and open the scheme interface
print "\nConnecting to Fluent sesssion:\n" + iorstring +"\n"
orb=CORBA.ORB_init()
fluent_session=orb.string_to_object(iorstring)
scheme_interface=fluent_session.getSchemeControllerInstance()
```

d. Load a local case file into the remote Fluent as a Server session.

   - Ask the user for the name of a local case file and read it into a local buffer.

   - Use `ICoFluentUnit.uploadFileFromBuffer()` to upload it to the remote working directory.

   - Use `ICoFluentSchemeController.loadCase()` to instruct the remote session to read the case file.

```
# Request the name of a local case file and upload it to the Fluent server
# machine and read it into Fluent. For simplicity in this example we assume
# here that it is a valid case file for the running session).
casefile=''
while not os.path.exists(casefile):
    casefile=str(raw_input('Please provide a case file to load:\n> '))

with io.FileIO(casefile) as f:
    fn_remote=os.path.basename(f.name)
    print 'Uploading file ' + fn_remote + '\n'
    f_buf_r=io.BufferedReader(f)
    scheme_interface.uploadFileFromBuffer(fn_remote,f_buf_r.read())
    fluent_session.loadCase(fn_remote)
```

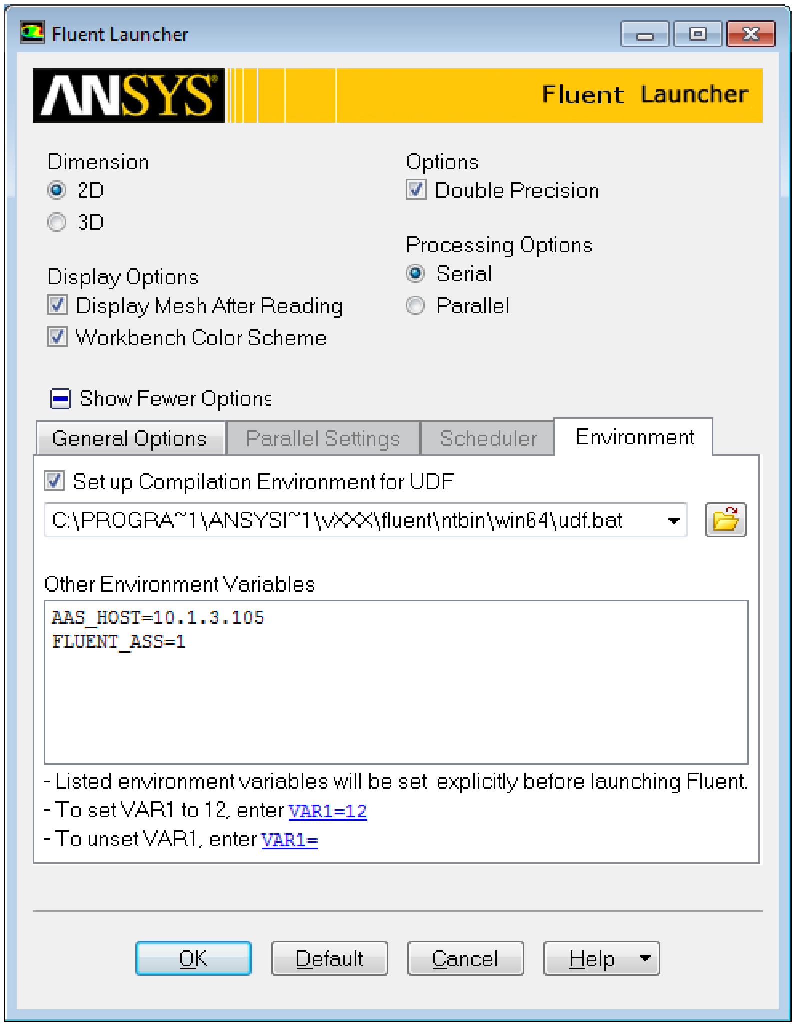
e. Loop through TUI / aaS commands provided by the user and use `ICoFluentSchemeController.doMenuCommandToString()` to execute them on the remote Fluent session and return the output to the user.

```
# Once the case file is loaded, the user may enter conventional TUI
# commands to perform simulation, write files, etc
while True:
    try:
        command=raw_input("Please enter a TUI command (END-COSIM to finish):\n> ")
    except:
        break
    if command=='END-COSIM':
        break
    output=scheme_interface.doMenuCommandToString(command)
    print output
```

f. Prompt the user for the names of files created on the remote system and use `ICoFluentSchemeController.downloadFileToBuffer()` to download them to the client machine.

```
# Ask the user for any remote files they would like to retrieve, download
# them into a buffer and write them to local files
files=scheme_interface.doMenuCommandToString('ls')
for fn_retrieve in iter(partial(raw_input, 'Filename to retrieve (CR when done): '), ''):
    f_local=io.FileIO(fn_retrieve,'w')
    f_buf_w=io.BufferedWriter(f_local)
    f_buf_w.write(scheme_interface.downloadFileToBuffer(fn_retrieve))
    f_buf_w.flush()
    f_buf_w.close()
```

3. Start a remote Fluent as a Server session and copy the resulting `aaS_FluentId.txt` file to a convenient location on the client machine. For this example, a remote 2D Fluent session is started on a Windows machine. For illustration purposes it is made interactive with the GUI displayed, but in general this is not necessary. (see Starting Fluent In Server Mode (p. 17) for details on starting Fluent as a Server sessions)



Dimension
[✓ 2D] 3D

Options
[✓] Double Precision

Processing Options
[✓ Serial] Parallel

Display Options
[✓] Display Mesh After Reading
[✓] Workbench Color Scheme

[✓] Set up Compilation Environment for UDF

C:\PROGRA~1\ANSYSI~1\vXXX\fluent\ntbin\win64\udf.bat

Other Environment Variables

```
AAS_HOST=10.1.3.105
FLUENT_ASS=1
```

- Listed environment variables will be set explicitly before launching Fluent.
- To set VAR1 to 12, enter VAR1=12
- To unset VAR1, enter VAR1=

4. Run the client application, `client_ex.py`

```
C:\Example>python client_ex.py
Please provide an aaS_FluentId.txt file:
>
```

5. Provide the path to the `aaS_FluentId.txt` file

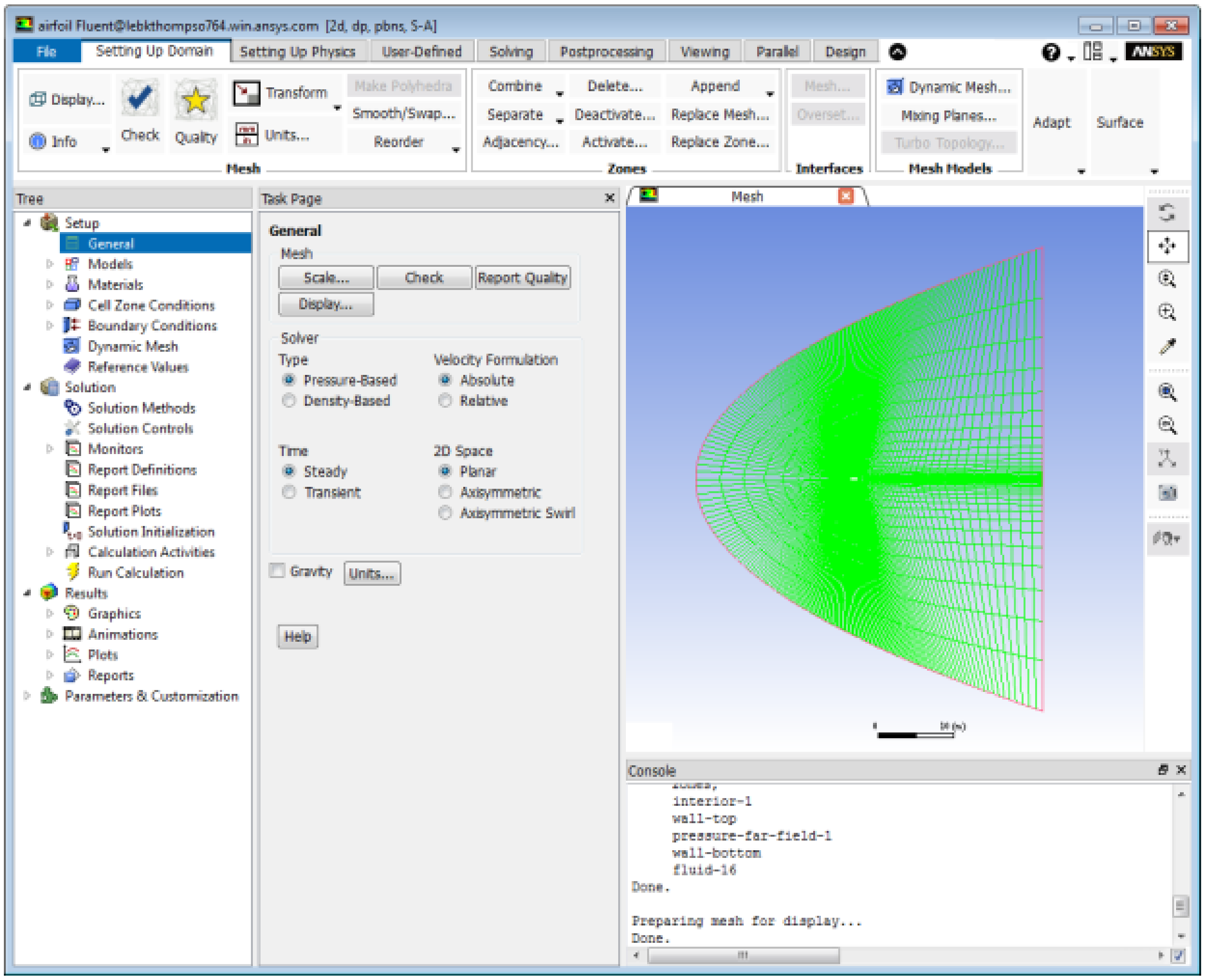
```
Please provide an aaS_FluentId.txt file:
> h:\FLServer\aaS_FluentId.txt

Connecting to Fluent sesssion:
IOR:010000002000000049444c3a4141535f434f5242412f49436f466c75656e74556e69743a312e
30000100000000000000068000000010102 7f0a00000031302e312e332e383300e8e21b0000001401
0f005253546cd1854f8c660d000000000001000000010000007f0200000000000000080000000016e
2aa4004f41540100000018000000001762aa4010001000100000000100010509010100000000000
```

```
Please provide a case file to load:
>
```

6. Provide the local path to a case file. In this case we are loading a 2D airfoil case

```
Please provide a case file to load:
> airfoil.cas.gz
Uploading file airfoil.cas.gz
```

```
Please enter a TUI command (END-COSIM to finish):
>
```



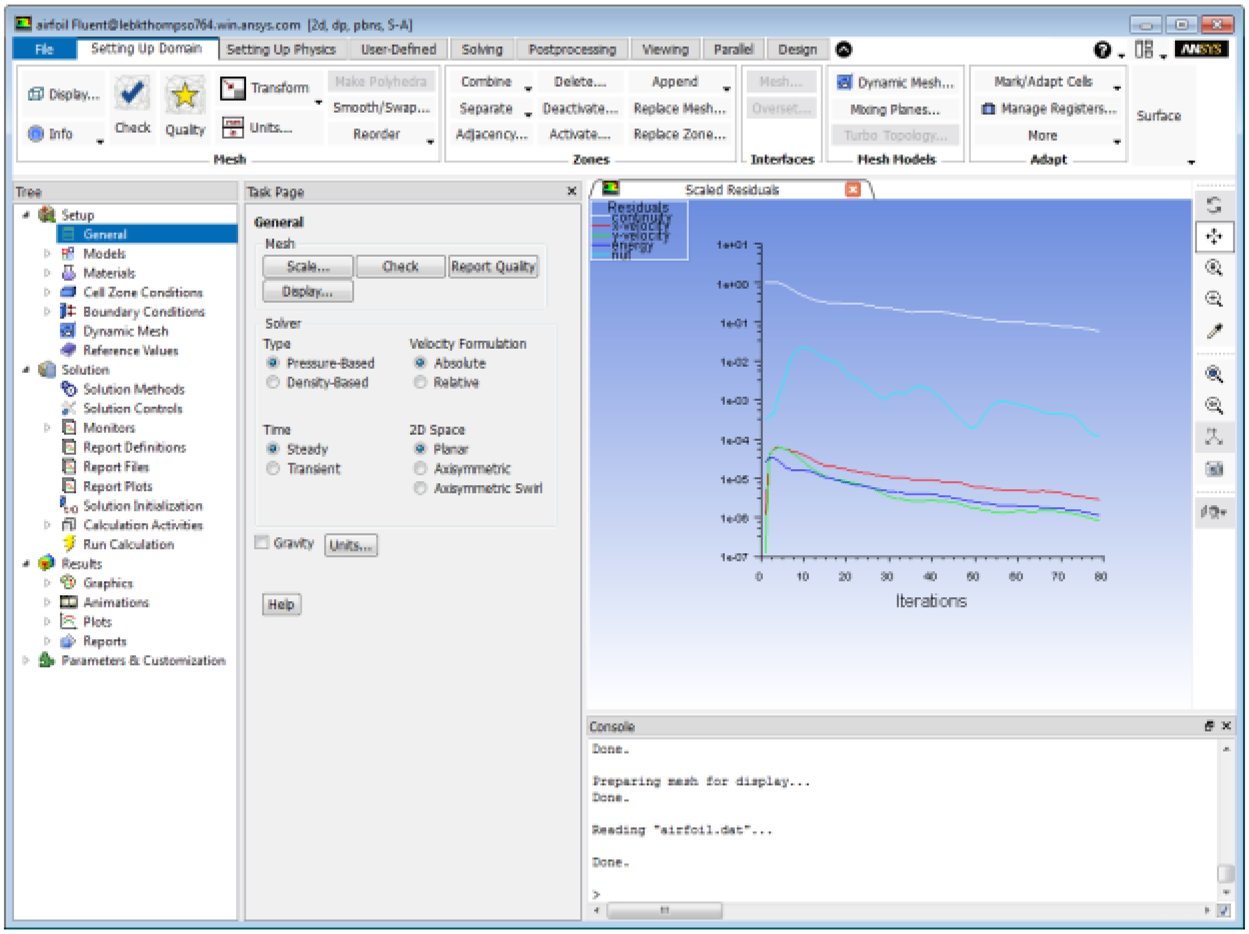
Solver
Type: [✓ Pressure-Based] Density-Based
Velocity Formulation: [✓ Absolute] Relative
Time: [✓ Steady] Transient
2D Space: [✓ Planar] Axisymmetric Axisymmetric Swirl
Gravity

```
        interior-1
        wall-top
        pressure-far-field-1
        wall-bottom
        fluid-16
Done.

Preparing mesh for display...
Done.
```

The case is uploaded to the remote machine and read into Fluent

7. Execute TUI commands to initialize the solution and iterate for up to 100 iterations. (some output has been omitted for clarity)

```
Please enter a TUI command (END-COSIM to finish):
> solve initialize hyb-initialization

Initialize using the hybrid initialization method.

Checking case topology...
-this case has farfield bc
-so it will be initialized with constant pressure

        iter             scalar-0

        1                1.000000e+00
        2                1.836666e-05
        3                1.327842e-06
        4                2.862682e-07
        5                9.451323e-07
        6                1.223471e-07
        7                2.501983e-07
        8                1.137657e-07
        9                4.959297e-08
        10               2.159624e-08
hybrid initialization is done.
Please enter a TUI command (END-COSIM to finish):
> solve iterate 100
  iter continuity x-velocity y-velocity     energy        nut     time/iter
     1 1.0000e+00 1.7946e-03 1.6266e-03 4.2495e-04 3.9276e-04  0:00:17   99
     2 6.3481e-01 2.0332e-03 1.3184e-03 1.1530e-04 2.9259e-04  0:00:25   98
     3 3.7028e-01 1.1406e-03 7.6986e-04 1.6216e-04 3.3093e-04  0:00:24   97
     .
     .
     .
    76 1.1063e-03 1.1319e-06 3.2180e-07 4.3823e-07 1.4217e-05  0:00:03   24
    77 1.0552e-03 1.0725e-06 2.9931e-07 4.1631e-07 1.4760e-05  0:00:03   23
  iter continuity x-velocity y-velocity     energy        nut     time/iter
    78 1.0072e-03 1.0196e-06 2.8048e-07 3.9612e-07 1.5606e-05  0:00:03   22
!   79 solution is converged
    79 9.6023e-04 9.6787e-07 2.6549e-07 3.7618e-07 1.6702e-05  0:00:03   21

Please enter a TUI command (END-COSIM to finish):
>
```



Type: [✓ Pressure-Based] Density-Based
Velocity Formulation: [✓ Absolute] Relative
Time: [✓ Steady] Transient
2D Space: [✓ Planar] Axisymmetric Axisymmetric Swirl

```
Done.

Preparing mesh for display...
Done.

Reading "airfoil.dat"...

Done.

>
```

8. Execute TUI commands to plot the pressure field, create an image file, and save the data file. Then enter END-COSIM to complete the simulation activities.

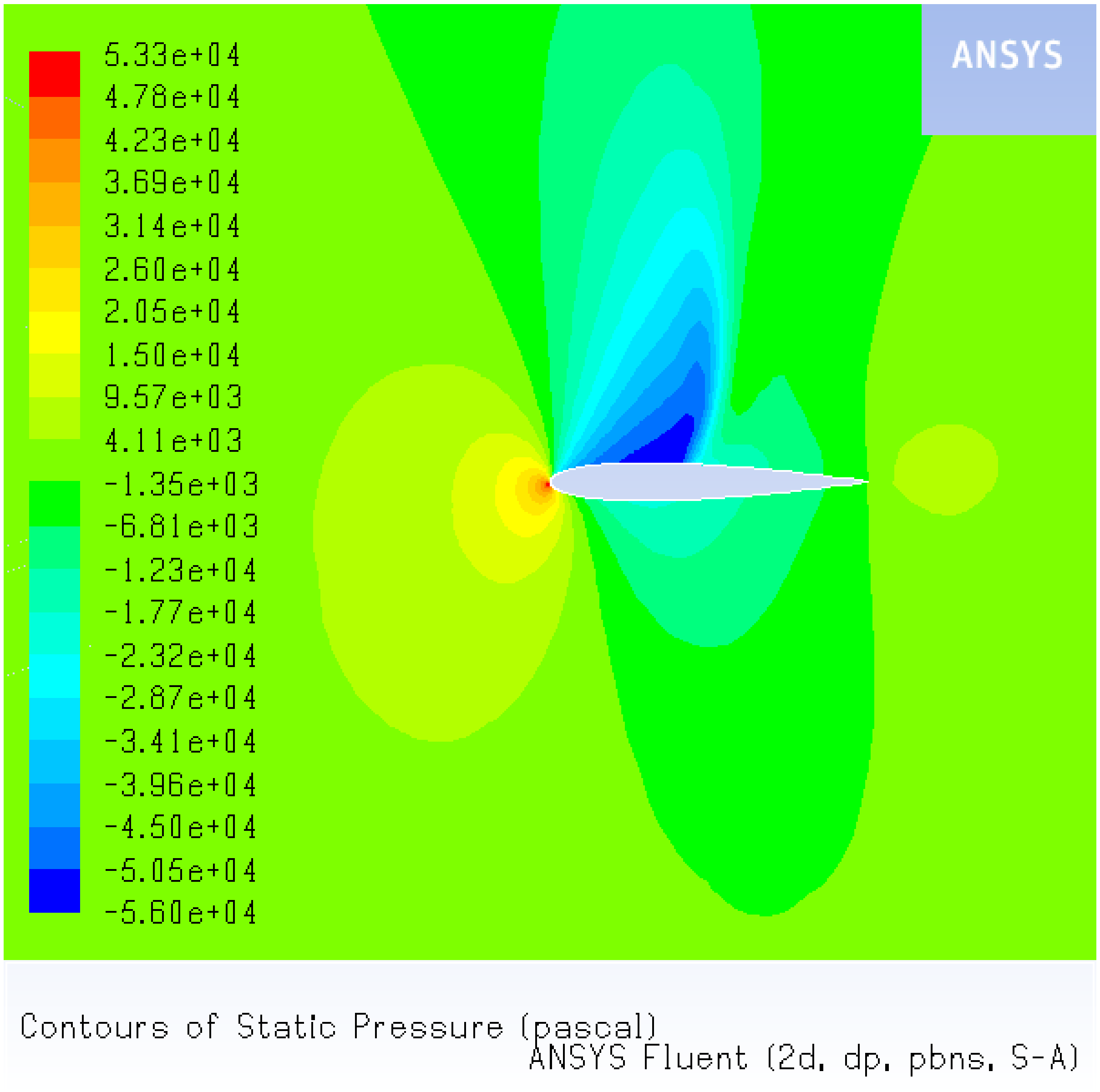
```
Please enter a TUI command (END-COSIM to finish):
> display views camera field 3 3

Please enter a TUI command (END-COSIM to finish):
> display views camera target 0 0 0
```

```
Please enter a TUI command (END-COSIM to finish):
> display set contours filled-contours? yes

Please enter a TUI command (END-COSIM to finish):
> display contour pressure , ,

Please enter a TUI command (END-COSIM to finish):
> display set picture driver png

Please enter a TUI command (END-COSIM to finish):
> display save-picture contour_pressure.png

Please enter a TUI command (END-COSIM to finish):
> file write-data airfoil.dat.gz

Writing "| gzip -2cfv > \"airfoil.dat.gz\""...
 27.0%
Done.

Please enter a TUI command (END-COSIM to finish):
> END-COSIM
Filename to retrieve (CR when done):
```

9. Retrieve the generated files from the remote machine and enter a Carriage Return when finished to exit.

```
Filename to retrieve (CR when done): airfoil.dat.gz
Filename to retrieve (CR when done): contour_pressure.png
Filename to retrieve (CR when done): <CR>
```

10. View the contour plot on the local machine



4.4.2. Summary

This example showed only the basic building blocks of a customized Fluent as a Server client application. By exploiting the full set of interface methods described in Fluent as a Server CORBA Interfaces (p. 50) you can build very powerful client applications tailored to the specific requirements of your application.

4.4.3. Code Listing

Below is the complete code listing for `client_ex.py`

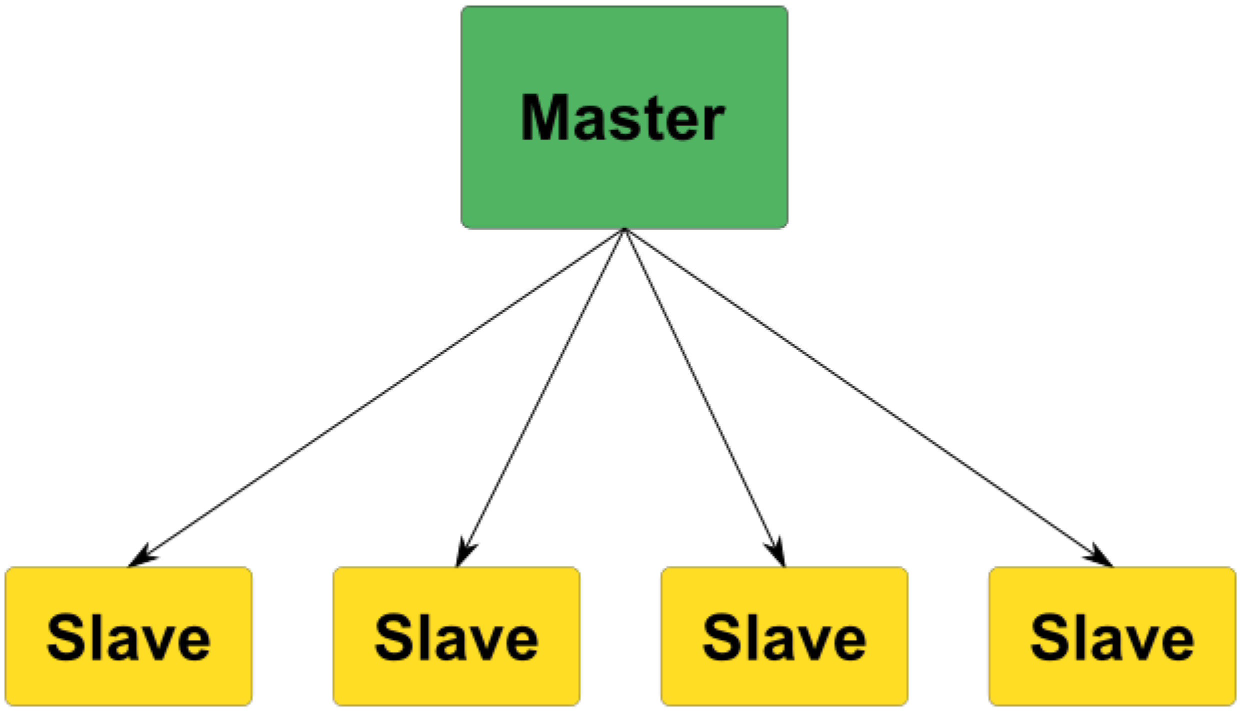
```
# client_ex.py
# A simple example of a Fluent As A Server client application
import os
import io
import CORBA
import AAS_CORBA    #import the AAS_CORBA module compiled from CoFluentUnit.idl
from functools import partial

# Request that the user supply an aaS_FluentId.txt file with connection
# details for a Fluent session
iorstring=''

while iorstring=='':
    filename=str(raw_input('Please provide an aaS_FluentId.txt file:\n> '))
    try:
        filehdl=open(filename,'r')
    except IOError:
        print(filename+" doesn't appear to exist!")
        continue

    if filehdl.read(3)!="IOR":   #if the file doesn't begin with IOR
        print(filename+" doesn't appear to contain a valid IOR string!")
        continue
    else:
        filehdl.seek(0)
        iorstring=filehdl.read()
    filehdl.close()

# Connect to the Fluent session and open the scheme interface
print "\nConnecting to Fluent sesssion:\n" + iorstring +"\n"
orb=CORBA.ORB_init()
fluent_session=orb.string_to_object(iorstring)
scheme_interface=fluent_session.getSchemeControllerInstance()

# Request the name of a local case file and upload it to the Fluent server
# machine and read it into Fluent. For simplicity in this example we assume
# here that it is a valid case file for the running session).
casefile=''
while not os.path.exists(casefile):
    casefile=str(raw_input('Please provide a case file to load:\n> '))

with io.FileIO(casefile) as f:
    fn_remote=os.path.basename(f.name)
    print 'Uploading file ' + fn_remote + '\n'
    f_buf_r=io.BufferedReader(f)
    scheme_interface.uploadFileFromBuffer(fn_remote,f_buf_r.read())
    fluent_session.loadCase(fn_remote)

# Once the case file is loaded, the user may enter conventional TUI
# commands to perform simulation, write files, etc
while True:
    try:
        command=raw_input("Please enter a TUI command (END-COSIM to finish):\n> ")
    except:
        break
    if command=='END-COSIM':
        break
    output=scheme_interface.doMenuCommandToString(command)
    print output

# Ask the user for any remote files they would like to retrieve, download
# them into a buffer and write them to local files
files=scheme_interface.doMenuCommandToString('ls')
for fn_retrieve in iter(partial(raw_input, 'Filename to retrieve (CR when done): '), ''):
    f_local=io.FileIO(fn_retrieve,'w')
    f_buf_w=io.BufferedWriter(f_local)
    f_buf_w.write(scheme_interface.downloadFileToBuffer(fn_retrieve))
    f_buf_w.flush()
    f_buf_w.close()
```

4.5. OLGA-Fluent Coupling

You can use the Fluent as a Server functionality to couple ANSYS Fluent with SPT-OLGA using the ANSYS Workbench or the ANSYS Twin Builder environment. The coupling is implemented through IronPython scripting that makes use of connectors/libraries that are supplied with Fluent. Both Workbench and Twin Builder incorporate IronPython scripting environments. So, while the examples in this manual will show the usage in Workbench, the same approach and techniques apply to coupling through Twin Builder.

The coupling is implemented as a Master/Slave model. In this model, a single process (the Master) has unilateral control over one or more other processes (the Slaves). The Slave processes do not communicate with each other and any synchronization between the Slave processes is the responsibility of the Master process. For example, in a transient simulation the Master process must manage the simulation time for all of the Slave processes.

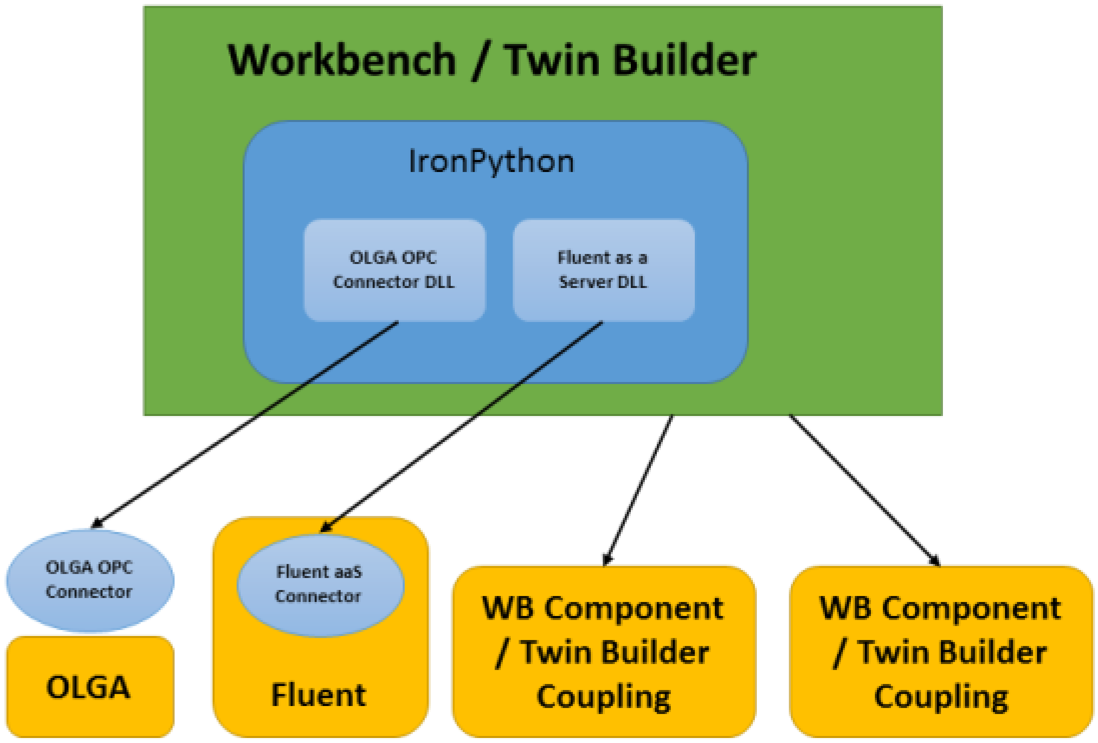
Figure 4.1: Master/Slave Model



In the OLGA-Fluent coupling implementation, the Master role can be filled by the IronPython interpreter that is available in either Workbench or Twin Builder. Fluent and OLGA are the Slaves (along with any other supported Workbench components or Twin Builder couplings). Several DLLs and helper applications are made available in the Software Development Kit to enable communication between the Master and Slave processes.

Figure 4.2: OLGA-Fluent Coupling Implementation (p. 71) shows how the Master/Slave model is employed for OLGA-Fluent coupling.

Figure 4.2: OLGA-Fluent Coupling Implementation



The important points to note are:

- The Master role is occupied by the IronPython interpreter that is present in both Workbench and Twin Builder.
- The OLGA OPC Connector DLL and Fluent as a Server DLL contain modules that are used to communicate with the OLGA and Fluent Slave processes. (Master-Side Libraries (p. 72))
- An OLGA OPC Connector application runs on the OLGA host machine and listens for connections from the Master process. (OLGA OPC Connector (p. 75))
- The Fluent as a Server Connector is embedded in the Fluent application and is enabled by starting Fluent in Server mode as described in Starting Fluent In Server Mode (p. 17).

4.5.1. Requirements

In addition to installations of Fluent and OLGA, you must have the following installed in order to use the OLGA-Fluent coupling capability: ANSYS Workbench or ANSYS Twin Builder.

4.5.2. Procedure

The basic steps to perform a coupled OLGA-Fluent simulation are as follows:

1. Start the OLGA application and start the OLGA OPC Server.

2. Start the OLGA OPC Connector on the OLGA machine with appropriate options (OLGA OPC Connector (p. 75)).

3. Start Fluent in server mode (Fluent as a Server Session Management (p. 17)).

4. Start Workbench or Twin Builder.

5. Run your IronPython co-simulation script in Workbench or Twin Builder. In order to perform co-simulation using the provided interfaces, your IronPython script must, at least:

   Load the OLGA OPC Connector DLL (OLGA OPC Connector DLL (p. 72)).

   Load the Fluent as a Server DLL (Fluent as a Server DLL (p. 72)).

4.5.3. Master-Side Libraries

The following libraries are included with the Fluent as a Server SDK and are used to communicate between the IronPython Master running in Workbench or Twin Builder and the Fluent and OLGA clients.

4.5.3.1. Fluent as a Server DLL

In order to communicate with a running Fluent as a Server session from the IronPython Master, it is necessary to load one of the pre-built .NET DLLs supplied with the Fluent as a Server SDK. Refer to Using the Pre-Built .NET Connector (p. 57) for additional information about the Fluent as a Server interfaces and the available DLLs.

4.5.3.2. OLGA OPC Connector DLL

IronPython clients running in either ANSYS Workbench or ANSYS Twin Builder connect to the OLGA OPC Connector through classes and methods defined in the file `DotNetCoOpcUnit.dll` and structures defined in `AasCoOpcUnit.dll`.

4.5.3.2.1. Libraries

DotNetCoOpcUnit.dll

public class `DotNetCoOpcServer`

public `DotNetCoOpcServer`();
constructor for DotNetCoOpcServer objects

public void `ConnectToServerFromIorFile`(string *`p_stringIorFile`*);
connect to the remote OLGA OPC server with OLGA aaS key *`p_stringIorFile`*

public string `getStatusReport`();
returns a status report from OLGA

public `DotNetOPCItemMgmt` `AddGroup`(string *`p_stringGroupName`*);
create an OPC group named *`p_stringGroupName`*

public void `RemoveGroup`(`DotNetOPCItemMgt` *`p_dot_netOPCItemMgt`*);
remove the OPC group named *`p_dot_netOPCItemMgt`*

public `DotNetOlgaBrowseServerCommands` `getOLGABrowseServerCommands`();
retrieve a browser of OLGA commands

public `DotNetOLGACommandProperties` `getOLGACommandProperties`();
retrieve a browser of OLGA command properties

public `DotNetOPCItemProperties` `getOPCItemProperties`();
retrieve a browser of OPC item properties

public `DotNetOPCBrowseServerAddressSpace` `getOPCBrowseServerAddressSpace`();
retrieve a browser of OPC Server address space

public class `DotNetOPCBrowseServerAddressSpace`

public string[] `BrowseOPCItemIDs`();
retrieve OPC item IDs from OLGA

public class `DotNetOlgaBrowseServerCommands`

public string[] `BrowseCommandIDs`();
retrieve a list of OLGA command IDs

public class `DotNetOLGACommandProperties`

string `GetCommandPropertiesReport`(string *`p_stringCommandID`*);
retrieve a command properties report for the OLGA command *`p_stringCommandID`*

public class `DotNetOPCItemProperties`

public string `GetItemPropertiesReport`(string *`p_stringItemID`*);
retrieve an item properties report for the OPC item *`p_stringItemID`*

public class `DotNetOPCItemMgt`

public AAS_CORBA.OPC.IOItem `AddItem`(string *`p_stringItemName`*);
add the OPC item *`p_stringItemName`* to an OPC group

public `DotNetIOPCSyncIO` `GetIOPCSyncIO`();
retrieve a synchronous access object

public class `DotNetIOPCSyncIO`

bool `ReadItemAsBool`(int *`p_iOPCHandle`*);

void `WriteItemAsBool`(int *`p_iOPCHandle`*, int *`p_iComType`*, bool *`p_Value`*);

bool[] `ReadItemAsBoolArray`(int *`p_iOPCHandle`*);

void `WriteItemAsBoolArray`(int *`p_iOPCHandle`*, int *`p_iComType`*, bool[] *`p_aValues`*);

double `ReadItemAsDouble`(int *`p_iOPCHandle`*);

void `WriteItemAsDouble`(int *`p_iOPCHandle`*, int *`p_iComType`*, double *`p_Value`*);

double[] `ReadItemAsBoolArray`(int *`p_iOPCHandle`*);

void `WriteItemAsDoubleArray`(int *`p_iOPCHandle`*, int *`p_iComType`*, double[] *`p_aValues`*);

int `ReadItemAsLong`(int *`p_iOPCHandle`*);

void `WriteItemAsLong`(int *`p_iOPCHandle`*, int *`p_iComType`*, int *`p_Value`*);

int[] `ReadItemAsLongArray`(int *`p_iOPCHandle`*);

void `WriteItemAsLongArray`(int *`p_iOPCHandle`*, int *`p_iComType`*, int[] *`p_aValues`*);

string `ReadItemAsString`(int *`p_iOPCHandle`*);

void `WriteItemAsString`(int *`p_iOPCHandle`*, int *`p_iComType`*, string *`p_Value`*);

string[] `ReadItemAsStringArray`(int *`p_iOPCHandle`*);

void `WriteItemAsStringArray`(int *`p_iOPCHandle`*, int *`p_iComType`*, string[] *`p_aValues`*);

string `ReadStringifiedItem`(int *`p_iOPCHandle`*);

void `WriteStringifiedItem`(int *`p_iOPCHandle`*, int *`p_iComType`*, string *`p_Value`*);

AasCoOpcUnit.dll

struct *`ICoOpcUnitError`*

Structure for exceptions returned from the OPC connection to OLGA

long *`m_iCode`*;

long *`m_stringDescription`*;

long *`m_stringInterfaceName`*;

long *`m_stringMoreInfo`*;

long *`m_stringName`*;

long *`m_stringOperation`*;

long *`m_stringScope`*;

struct *`IOItem`*

Structure for items in OLGA. To add items, see public AAS_CORBA.OPC.IOItem `AddItem`(string *`p_stringItemName`*); (p. 73). To read/write items, see public class `DotNetIOPCSyncIO` (p. 73).

long *`m_iCOMType`*

long *`m_iOPCHandle`*

string *`m_stringItemId`*

4.5.4. Slave-Side Connectors

4.5.4.1. Fluent as a Server

The Fluent as a Server connector is embedded in the Fluent installation. To enable it, start Fluent in server mode. For more information refer to Starting Fluent In Server Mode (p. 17).

4.5.4.2. OLGA OPC Connector

The OLGA OPC Connector provides a listening interface for connections to OLGA from an IronPython client running in either ANSYS Workbench or ANSYS Twin Builder. The connector interface implements a subset of OPC interfaces. Two versions are provided for use depending on whether the machine running OLGA is 32-bit or 64-bit architecture:

32–bit
`v202\fluent\fluent20.2.0\addons\corba\ntx86\AasCoOpUnit.exe`

64–bit
`v202\fluent\fluent20.2.0\addons\corba\win64\AasCoOpUnit.exe`

Usage

The connector accesses OLGA through a COM OPC interface. The COM OPC OLGA server has to be started before starting the connector. Once the OLGA OPC server is running, you can invoke the connector with the following command:

```
AasCoOpcUnit.exe -host<host> -port<port> -portspan<portspan> -iorFile<filename> -opcCLSID<CLSID>
```

`<host>`
The hostname or IP address on which the AasCoOpcUnit application will listen. This is the name that will be made available in the IOR file that is created and used to connect. If omitted, the connector will listen on all available IP addresses. If set to `localhost`, only local clients will be able to connect.

`<port>`
The port number on which to listen for connections. If no port is specified, a randomly chosen port will be used. If **`<portspan>`** is also specified, **`<port>`** is the base of the port range.

`<portspan>`
Specifies that a range of **`<portspan>`** ports should be used. This requires that <port> also be specified. The actual range used will be [**`<port>`**:**`<port>`**+**`<portspan>`**-1]

`<filename>`
specifies the name to use for the IOR text file containing the OLGA OPC connection key. By default, the name will be aaS_OPCId.txt.

`<CLSID>`
specifies the CLSID of the OPC server. The default value is "SPT.OlgaOPCServer.7'. Check the OLGA documentation for the correct value.

After starting the OLGA OPC Connector, the following session information will be displayed:

Name of the key file

OPC CLSID

Result of checking Vendor info and version

Listening endpoints (for example, <protocol>://<IP>:<port>)

4.5.5. OLGA-Fluent Coupling Examples

4.5.5.1. OLGA OPC Example Code

These examples are intended to demonstrate usage of the OLGA OPC Connector libraries to communicate and control an OLGA session with a running OLGA OPC Server. In order to work, the OLGA OPC Connector application must also be running on the OLGA machine (OLGA OPC Connector (p. 75)).

Establishing an OLGA Connection and Exception Handling

This example represents the simplest case of connecting to an OLGA session through the OLGA OPC Connector and requesting a status report from OLGA. It also illustrates a recommended approach for using try/except blocks for catching exceptions:

```
######################################################################
#                   OLGA OPC Server Version Report                   #
######################################################################

import clr
import sys

# Load the DLLs that define the OLGA OPC connection classes and structures
sys.path.append('C:/Program Files/ANSYS Inc/v202/fluent/fluent20.2.0/addons/corba/DotNetFramework40')
clr.AddReferenceToFile('DotNetCoOpcUnit.dll')
clr.AddReferenceToFile('AasCoOpcUnit.dll')

# Import classes and structures
import AAS_CORBA
from AAS_CORBA.OPC import DotNetCoOpcServer
from AAS_CORBA.OPC import ICoOpcUnitError

try:
 # Instantiate a DotNetCoOpcServer object and use the ConnectToServerFromIorFile
 # method to connect using a provided keyfile
 iOPCServer=DotNetCoOpcServer()
 iOPCServer.ConnectToServerFromIorFile('C:/Work/OLGA-Coupling-Demo/aaS_OpcId.txt')

 # Request a status report and print the output
 print iOPCServer.getStatusReport()
except ICoOpcUnitError as error:
   # An exception was thrown by the OLGA OPC connector
 print "ICoOpcUnitError Exception caught:"
 print "\tCode       = ", error.m_iCode
 print "\tDescription = ",error.m_stringDescription
 print "\tInterface   = ",error.m_stringInterfaceName
 print "\tMore Info   = ",error.m_stringMoreInfo
 print "\tOperation   = ",error.m_stringOperation
 print "\tScope       = ",error.m_stringScope
except Exception as ex:
 print "Exception caught..."
 print ex.message
except:
 print "Unknown Exception..."
```

Creating and Manipulating Groups and Items in OLGA

This example creates an OPC item to act as a simple external clock for the OLGA simulation and increments the clock time. It demonstrates creation and manipulation of groups and items in OLGA. Such an external clock is a crucial component of coupled transient simulations.

```
######################################################################
#                   OLGA OPC Sample Clock Operations                 #
######################################################################

import clr
import sys

from System.IO import File
sys.path.append('C:/Program Files/ANSYS Inc/v202/fluent/fluent20.2.0/addons/corba/DotNetFramework40')
clr.AddReferenceToFile('DotNetCoOpcUnit.dll')
clr.AddReferenceToFile('AasCoOpcUnit.dll')

import AAS_CORBA
from AAS_CORBA.OPC import DotNetCoOpcServer

try:
 iOPCServer=DotNetCoOpcServer()
 iOPCServer.ConnectToServerFromIorFile('C:/Work/OLGA-Coupling-Demo/aaS_OpcId.txt')

 # Add a Group and use the GetIOPCSyncIO method to instantiate a syncrhonous I/O object
 iOPCItemMgt=iOPCServer.AddGroup("WB")
 iOPCSyncIO=iOPCItemMgt.GetIOPCSyncIO()

 # Add the OPC items ExternalClock and INITTIME to the newly created WB group
 ioSimServerDemoExternalClock            = iOPCItemMgt.AddItem('Sim.ServerDemo.ExternalClock')
 ioSimServerDemoINITTIME                 = iOPCItemMgt.AddItem('Sim.ServerDemo.INITTIME')

 # Use the synchronous I/O object to get a numerical and a string representation of the INITTIME value.
 stringInitTime=iOPCSyncIO.ReadStringifiedItem(ioSimServerDemoINITTIME.m_iOPCHandle)
 doubleInitTime=iOPCSyncIO.ReadItemAsDouble(ioSimServerDemoINITTIME.m_iOPCHandle)
 print 'Sim.ServerDemo.INITTIME:'
 print '\tStringified value:',stringInitTime
 print '\tNumerical value:',doubleInitTime, 'days'

 # Use the synchronous I/O object to get a numerical and a string representation of the ExternalClock value.
 stringExternalClock=iOPCSyncIO.ReadStringifiedItem(ioSimServerDemoExternalClock.m_iOPCHandle)
 doubleExternalClock=iOPCSyncIO.ReadItemAsDouble(ioSimServerDemoExternalClock.m_iOPCHandle)
 print 'Sim.ServerDemo.ExternalClock:'
 print '\tStringified value:',stringExternalClock
 print '\tNumerical value:',doubleExternalClock, ' days'

 #advance clock 1 day
 doubleExternalClock=doubleExternalClock+oneDay
 iOPCSyncIO.WriteItemAsDouble(ioSimServerDemoExternalClock.m_iOPCHandle,
  ioSimServerDemoExternalClock.m_iCOMType,doubleExternalClock)
 stringExternalClock=iOPCSyncIO.ReadStringifiedItem(ioSimServerDemoExternalClock.m_iOPCHandle)
 doubleExternalClock=iOPCSyncIO.ReadItemAsDouble(ioSimServerDemoExternalClock.m_iOPCHandle)
 print 'Sim.ServerDemo.ExternalClock advanced 1 day:'
 print '\tStringified value:',stringExternalClock
 print '\tNumerical value:',doubleExternalClock, ' days'

 # Remove the group
 iOPCServer.RemoveGroup(iOPCItemMgt)
except ICoOpcUnitError as error:
 print "ICoOpcUnitError Exception caught:"
 print "\tCode       = ", error.m_iCode
 print "\tDescription = ",error.m_stringDescription
 print "\tInterface   = ",error.m_stringInterfaceName
 print "\tMore Info   = ",error.m_stringMoreInfo
 print "\tOperation   = ",error.m_stringOperation
 print "\tScope       = ",error.m_stringScope
except Exception as ex:
 print "Exception caught..."
 print ex.message
except:
 print "Unknown Exception..."
```

Index