Prepared by
Lukas Muttenthaler

Prepared at
Institute of Machine Design and Hydraulic Drives

Supervisor #1
Rudolf Scheidl

Supervisor #2
Bernhard Manhartsgruber

June 2017

Scripted CFD simulations and postprocessing in Fluent and ParaVIEW

Manual

about doing automated CFD simulations with

Fluent

and scripted postprocessing with

ParaVIEW

Version 1.0 - “Rocky Road”

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Table of Contents

Preface

This document is about performing Computational Fluid Dynamics (CFD) simulations in Fluent as a stand-alone program and using Fluent via Workbench. Fluent as well as Workbench are programs/components of ANSYS. Results of CFD simulations have to be analysed, whereby ParaVIEW is a possible choice for doing analysis and postprocessing. Both steps, the simulation and the analysis can be done with the Graphical User Interfaces of the programs, but it is also possible to do this with text-based scripts.

The main reason of using scripts for the application of the programs, is the more flexible usage. Additional, equal or similar simulations and analysis run more than one time. So text-based scripts and the Application Programming Interfaces (API) allows parameter studies and optimisations by external communication with various programming languages and programs. Therefore, setting parameters of CFD simulation cases by using MATLAB, C/C++, Python and so on is possible.

On the following pages script-based CFD is explained. However there are no explanations about physical or mathematical fundamentals of fluid dynamics or the correct procedure of the various settings of computational tools to get correct results.

The motivation of this publication concerning the text-based usage, is the missing documentation and the high number of programming languages of ANSYS APIs. The ambition of creating this manual is to ease other engineers and researchers the advanced utilisation of simulating and analysing fluid dynamic cases and to come in exchange with simulation experts and their experiences.

After an introduction, the manual contains three main issues:

- Defining and running a simulation setup and saving the results in two different ways:
  - Fluent stand-alone
  - Fluent via Workbench
- Analysing the results with ParaVIEW

This document was created during doctoral program at the Institute of Machine Design and Hydraulic Drives at the Austrian Johannes Kepler University in Linz. So I would like to express my gratitude and thanks to my supervisors and my scientific advisers o.Univ.-Prof. Dipl.-Ing Dr. Rudolf Scheidl and a.Univ.-Prof. Dipl.-Ing Dr. Bernhard Manhartsgruber for their advice, patience and support. Furthermore for their useful comments and remarks.

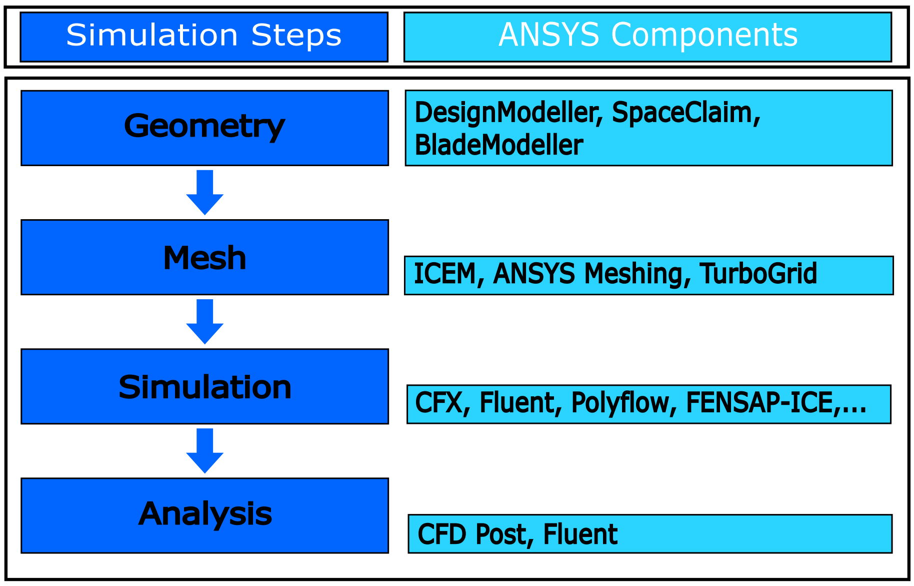
A steady improvement of this manual can only be reached by feedback, that's why it would be wonderful to get opinions, views and ideas of different ways of doing CFD simulations and analysing the results. Therefore, please contact the author via ResearchGate or E-Mail.

Chapter 1

Introduction

1.1 Methodology of CFD simulations

The procedure of simulating a fluid flow can be divided into four steps. An overview can be seen in Figure 1.1.



Simulation Steps
ANSYS Components
Geometry
DesignModeller, SpaceClaim, BladeModeller
Mesh
ICEM, ANSYS Meshing, TurboGrid
Simulation
CFX, Fluent, Polyflow, FENSAP-ICE,...
Analysis
CFD Post, Fluent

Figure 1.1: Process of CFD simulations

Foremost the geometry of the CFD simulation part is necessary. Therefore, the geometry and physical boundaries have to be created or imported from a Computer Aided Design (CAD) software. One of the most decisive points of the first step is the reduction of the geometry to a simple simulating part. Not required details may not included in the geometry model. The less details the geometry contains the easier is the second step. Thus, the geometry should be as simple as possible, but as complicated as necessary. Many details lead to a high number of cells and to a long simulation time.

In the second step the volume of the created geometry gets divided into discrete cells. The generation of a good and useful mesh may be quite complicated. The quality of the mesh is very important and significant to the convergence of the simulation and the quality of the result.

After generating a good mesh, the setup of a simulation can be specified. These setups contain the definitions of the boundary conditions and the settings of the physical model, discretisation methods, materials and others.

At the end of a typical simulation procedure, the analysis of results and the determination of the result quality is necessary.

Sometimes it is necessary to reiterate some of the steps, if the simulation is not stable, convergence can not be achieved, the result is not plausible or the result differs from measurements, experiences or other calculations. CFD simulations should never be a stand-alone technology but rather in combination with experiments and analytical or other calculations. The optimum is in the intersection of the three circles as seen in Figure 1.2. [1, 2]

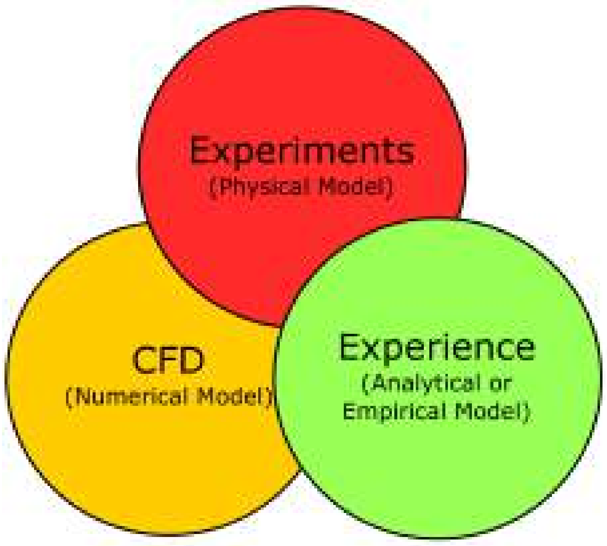


Figure 1.2: Methodology of fluid dynamics

1.2 Scripting of CFD simulations

Most commercial tools as Fluent, CFX and Star-CCM+ offers a Graphical User Interface (GUI) to interact with the software. Additional, it is possible to use Text-based User Interfaces (TUI). Otherwise open-source programs like openFOAM and SU2 are often text-based. [3–6]

TUIs open a wide range of possibilities, because you manipulate the setup, control the simulation, read and write variables and add new functionalities:

- Performing parameter studies.
- Automating several steps if wanted.
- Defining simulations without GUIs and evading limitations caused by them.
- Adding new functionalities.
- Linking CFD programs to other routines and using the calculating core.

1.3 Programs and Versions

The instructions in the next chapters were applied to the following program releases:

- ANSYS: Release 17.1
- ParaVIEW: Release 5.3.0

Chapter 2

Using Windows Shell

The Windows shell is the interface of Windows operating systems and allows to communicate with it. Files can be manipulated, created or deleted, as well as directory operations can be done. To open the Command line, one has to click the Start button and type cmd in the Search line. (Don't forget to press `Enter`.)

Normally, Windows shell starts at the user directory and the commands are not case sensitive. A ready shell is shown by the `>` symbol. To get information of available commands `help` can be used. `help command` shows information about the specific command and `command /?` displays the options.

Some useful and relevant commands for Windows shell are composed in Table 2.1.

Argument	**Process**
cd *directory*	Changes current directory to *directory.*
cd..	Move current directory one steps upwards.
cd\	Move current directory to *C:*\
mkdir *directory*	Create the specified directory.
rmdir *directory*	Delete the specified directory.
dir	Shows all files in the current directory.
rename *dir1 dir2*	Rename the specified directory.
rename *file.ext file2.ext*	Rename the specified file.
file.exe	Run the specified executable.
copy *file.ext file2.ext*	Copy the specified file.
del *file.ext*	Delete the specified file.
exit	Close the Command line.

Table 2.1: Some relevant commands for Windows Command line

For executing shell commands more than once, commands can be written to files. Command scripts for the shell can be created and edited with any editor and stored as .bat or .sh file. Following this, the .bat or .sh file can be executed (also by double-clicking on them). These files offers a wide range of possibilities. Examples are running CFD simulations and executing different tasks, files and directories can be manipulated, process management can be done, the operating system can be monitored and configured.

It may be necessary to run the shell as administrator, otherwise it is possible, that the operating system asks for administrator rights. Otherwise Windows asks, if the user wants to allow the program to make changes to the computer. So, if one call them with the shell, the shell has to be opened with administrator rights or the .bat file can be also executed with administrator rights.

A whole list of shell instructions can be found in the Command-Line Reference of Microsoft Technet. For other operating systems exists several other websites, manuals and registers with a command overview like the A-Z Index of the Bash Command Line for Linux.

Chapter 3

Using ANSYS Workbench

3.1 Basics of ANSYS Workbench and IronPython

ANSYS Workbench is a framework of different ANSYS programs (they are called components in Workbench). Each component can be used as stand-alone program as program linked in the Workbench Environment. Linking the components with Workbench by using the GUI, allows processes like updating the whole project, using of parameters (in a limited extent, because not all settings, properties or values are parameterisable in ANSYS), bidirectionally communicating with internal (ANSYS components) and external (third-party software) interfaces, including multiphysics simulations. For this reason, the Workbench offers possibilities of an extended and advanced utilisation, but there are still limitations:

- Not all settings, properties or values are parameterisable.
- External Interfaces are only available for specific programs/programming languages.
- Restriction in variability.

IronPython, a Python implementation in C#, is the language of the API. This implementation can use .NET-Assemblies. It is an interpreted, object-oriented language with an easy and clear syntax and is ideal for rapid prototyping. [7]

The Workbench API of ANSYS Release 17.1 is based on IronPython 2.7.0.40. For this reason scripting in Workbench uses an object-based approach. Properties of objects have values, modified by modules.
There are numerous books and guides about (Iron-)Python programming. The Python Documentation and IronPython Documentation gives an explanation of many versions of (Iron-)Python and online Python interpreters help to evaluate small little scripts. But there are little differences, by using different interpreters.

3.2 IronPython in the ANSYS Workbench Environment

Workbench projects consist of systems and components and their linkages, whereby systems are predefined collections of components, which are often used together. [8]

The `>>>` symbol shows a ready command prompt. An overview of using Text User Interface commands for Workbench is given on the next pages. [9–11]

3.2.1 Syntax

In the syntax you may use lowercase (a-z) and uppercase (A-Z) letters, numbers (0-9) and the special characters for namings: _
To achieve a well readable style and a general valid syntax for naming variables, modules, classes and so on, hints and tricks can be found in the internet. [12]
Comments starts with # and end at end of line.

3.2.2 Variables

A function in python uses a local scope of variables. Variables of this scope are defined in the function or in the parameter list. If no local variable can be found, a global variable is wanted. If a global variable also can not be found, the program exits with an error.

```
>>> x = 18 #Define x as 18
>>> print(x) #Print x at the prompt
18
>>> x = None #Delete the "value" of x and sets to null
>>> del x #Delete the variable
>>> print(y) #Print y (undefined) at the prompt
UnboundNameException: name 'y' is not defined
>>> def func(): #Define a function func()
    y = 19 #(lokal) Assignment of 19 to y in the function
>>> print(y)
UnboundNameException: name 'y' is not defined
```

Listing 3.1: Definition of variables

```
>>> a = 14 #Integer variable (exact the specified value)
>>> print(a)
14
>>> b = 14.0 #Floating point variable
>>> print(b)
14.0
>>> c = 2.7e3 #Floating point variable in exponential notation (also: 2.7E3)
>>> print(c)
2700.0
>>> s1 = "Koalas like eucalyptus." #Define a string
>>> print(s1)
Koalas like eucalyptus.
>>> s2 = 'Other marsupials like eucalyptus too.' #Define also a string
>>> s3 = """Eucalyptus
smells
well.""" #Defines a multiline string (also: ''')
>>> print(s3)
Eucalyptus
smells
well.
>>> x1 = [1.1, 2.2, 3.3] #Define a list of numbers(changeable)
>>> x2 = ["Vienna","London","Paris"] #Define a list of strings
>>> x3 = [["Shanghai","China",24000000],["Rio de Janeiro","Brazil",6500000]]
    #Define a multidimensional list with different objects
>>> print(x3[0])
["Shanghai","China",24000000]
>>> print(x3[0])
>>> print(x3[0][1])
```

```
China
>>> t = True #Define t as (boolean) true
>>> f = False #Define f as (boolean) false
```

Listing 3.2: More datatypes in Python

There exist even more datatypes in Python such as tuples, sets, dictionaries and others. Have a look at the different documentations.

3.2.3 Mathematical Basic Operations

Some basic operations can be seen in the following lines.

```
>>> 78 + 2 #Addition
80
>>> 78 - 2 #Subtraction
76
>>> 3 * 5 #Multiplication
15
>>> 9 / 2 #Integer division (also: 9 // 2)
4
>>> 9.0 / 2 #Normal division (also: 9 / 2.0 or 9.0 / 2.0)
4
>>> 5.2 ** 3 #Exponential Calculation
357.911
>>> a = 3 #Define the value of a
>>> a += 2 #Add 2 to the right operand and saves it to the left: a = a + 2
>>> print(a)
5
```

Listing 3.3: Mathematical Basic Operations

3.2.4 Conditionals

Statements often have to perform different actions depending on conditional statements or depending on the value of a variable. So the program flow changes and depends on this states.

```
>>> a == b #Check, if a is equal to b
>>> a != b #Check, if a is not equal to b (also: <>)
>>> a < b #Check, if a is less to b (also: >)
>>> a <= b #Check, if a is equal to a (also: =>)
```

Listing 3.4: Relational and equivalence expressions

```
>>> not(True) #Invert boolean value
>>> a == a and b == b #True, if a is equal to a and b is equal to be
>>> a == a or b == b #True, if a is equal to a or b is equal to be
```

Listing 3.5: Boolean expressions

```
>>> #Determine. if x is bigger or not than 0 and return specified string
>>> x = 1
if x > 0:
    print("x is bigger than 0.")
else:
    print("x is not bigger than 0.")
x is bigger than 0.
```

```
>>> #Determine, if y is bigger, less or equal to 0 and return specified
    string
>>> y = 0
if y > 0:
    print("y is bigger than 0.")
elif y < 0:
    print("y is less than 0.")
else:
    print("y is 0.")
y is 0.
```

Listing 3.6: if-else-Statements

The `else` statement is not mandatory.

3.2.5 Iterations

In (Iron-)Python two kinds of loops are available.

for-Loops

For loops are used to iterate over a sequence of various elements. This sequence could be a list, string or tuple.

```
>>> #Run through specified list of numbers
>>> listOfPrimes = [2, 3, 5, 7] #Define list of prime numbers
for prime in listOfPrimes: #Run through listOfPrimes
    print(prime)
2
3
5
7
>>> #Run through specified list of strings
>>> listOfCities = ["Melbourne","New York","Rome"] #Define list of cities
for city in listOfCities: #Run through listOfCities
    print(city)
Melbourne
New York
Rome
>>> #Create list with range() and run through specified list
for numbers in range(3,6,1): #Run through list, range(start, stop, inc)
    print(numbers)
3
4
5
```

Listing 3.7: for-Loop

while-Loops

The while loop repeats as long as a specified expression is true.

```
>>> count1 = 0
>>> count2 = 100
>>> while count1 <= 3: #Repeat loop as long as count1 is less than 3
    print(count1)
    print(count2)
```

Symbol	Meaning
0	Conversion will be filled with zeros.
-	Conversion is left adjusted.
+	Conversion is always preceded by a sign character.
	No flag (default) means padding with leading blanks.

Table 3.1: Flags for string modulo operator placeholders

```
    count1 += 1 #Increase count1
    count2 += 1 #Increase count2
    if count2 >= 102: #If count2 is bigger or equal to 102, exit loop
        break
0
100
1
101
```

Listing 3.8: while-Loop

To exit a for or while loop a `break` statement can be used. To skip the current and start the next loop with the next value a `continue` statement can be used.

3.2.6 Formatted Output

In Python there are two ways of generating formatted outputs. The older way is the method with a string modulo operator `%`. The better way is, to use `format`.

String Modulo Operator

There are three indispensable syntax parts . The first part is a string with a placeholder, that specifies the format of the inserted argument. The string modulo operator is necessary and last the arguments (with their values) are needful.

The general syntax of the placeholder: `%[flags][width][.precision]type`

The width specifies the total width of the number and means the total number of digits including the decimal point. As a number consists of less characters than specified, the output is filled with leading blanks by default.
Precision specifies the number after the decimal point filled with zeros on the right side.
Flags specify alignment, padding and preceding characters and are described in Table 3.1.
The syntax to configure a special type is in Table 3.2.

```
>>> "%6d, %06d, %+6d, %6i" % (789, 789, 789, 789)
'   789, 000789,   +789,    789'
>>> "%4d, %4u" % (-456, -456)
>>> "%8.3f, %08.3f, %+8.3f, %8.3F" % (123.45, 123.45, 123.45, 123.45)
' 123.450, 0123.450, +123.450,  123.450'
>>> "%8.3e, %08.3e, %+8.3e, %8.3E" % (123.45, 123.45, 123.45, 123.45)
'1.235e+02, 1.234500e+002, +1.234500e+002, 1.235E+02'
>>> "abc %s def" % (-987.65)
'abc -987.65 def'
>>> "abc %s def" % ("xyz")
'abc xyz def'
```

Symbol	Meaning
d, i	Signed decimal integer
u	Unsigned decimal integer
f, F	Floating-point number.
e, E	Floating point number in exponential format. ("e","E")
s	String (any object converted by (str())
%	"%" character

Table 3.2: Types for string modulo operator placeholders

Symbol	Meaning
0	Conversion will be filled with zeros.
-	Conversion is only preceded by a negative character.
+	Conversion is always preceded by a sign character.
	No specification (default) means padding with leading blanks.
<	Field is left aligned (default) in the available space.
^	Field is centered.
>	Field is right aligned.
,	Thousands seperators are used.

Table 3.3: Options for format placeholders

```
>>> "Bananas: %5d, Price:%8.2f" % (243, 957.4)
'Bananas:   243, Price:  957.40'
>>> "Percentage Sign: %% " % ()
'Percentage Sign: % '
```

Listing 3.9: Examples for formatted output with string modulo operator

format

The newer method for formatted outputs is the `format` way. The structure is similar to the string modulo operator way. A string s with placeholder is needful. The `format` method is called with arguments that should replace the placeholders:
`s.format(*args, **kwargs)`

The placeholders are surrounded by curly braces. Inside the round braces the first number specifies the position of the argument in the argument list. The second part of the placeholder is the format code, which is similar to the format code of the string modulo operator showed in Table 3.2. The characters which specifies the argument type are the same than above, but `u` and `i` are not used. Options, which are described in Table 3.3, modifies the format of the output.

```
>>> "{0:6d}, {0:06d}, {0:+6d}, {0:-6d}, {0:<6d}, {0:^6d}".format(-66)
'   -66, -00066,    -66,    -66, -66   ,  -66  '
>>> "{0:6.3f}, {0:06.3f}, {0:6.3F}, {0:6.3e}, {0:6.3E}".format(-11.22)
'-11.220, -11.220, -11.220, -1.122e+01, -1.122E+01'
>>>"abc {0:s} abc {0:15s} abc {0:>15s} abc".format("xyz")
'abc xyz abc xyz             abc             xyz abc'
```

Listing 3.10: Examples for formatted output with format method

3.2.7 eval- and exec-Statements

`exec` executes a (dynamically) created statement or program and accepts statements like `import`, `class`, `def`, `for`, assignments and programs but it ignores return values.

`eval` evalutes only single expressions and give back the return value. The above examples are accepted from `exec` but not from `eval`.

```
>>> a = 2
>>> b = eval("a + 3") #Evaluate the expression and assign to b
>>> print(b)
5
>>> exec("b = a + 2") #Execute the assign statement
>>> print(b)
5
```

Listing 3.11: Comparison of eval and exec

These both functions are useful in combination with `format` to generate a string with file path and name and load or store data or to do parameter studies or set values of variables.

3.2.8 Directory and File Operations

To do file operations the `sys` module is necessary. The module can be imported with `import sys` command. In order to read data from file or write data to file, the file has to be opened with the `open` command and is closed after the usage with `close`.

```
>>> import sys #Imports the module
>>> #Read from file
>>> fileId1 = open("data1.txt","r") #Open file in read mode and gives back
   file object
>>> data = fileId1.read() #Read all lines
>>> fileId1.close() #Close file
>>> #Write to file
>>> fileId2 = open("data2.txt","w") #Open file in write mode and gives back
   file object
>>> fileId2.write("First line!\n") #Write string to file
>>> fileId2.close() #Close file
```

Listing 3.12: Simple example for editing file data

There are even more possibilities to edit the data of files. The `os`, `subprocess` and `shutil` modules enables to interact with the operating system and other processes and to use high-level file and directory manipulation. Have a look at the online documentations of (Iron-)Python.

3.2.9 Useful Built-In Functions and Modules

Help information concerning an object `help(object)` is available. `dir(object)` will give a list of current scope names, `globals()` returns a dictionary of global symbol table and `locals(object)` returns a dictionary of local symbol table.

There are also some mathematical basic operations as `min(iterable)`, `max(iterable)`, `abs(number)`, `pow(x, y)` and so on.

3.3 Scripting in ANSYS Workbench

3.3.1 Executing Commands and Scripts with the GUI

Python scripts can be executed by selecting **File > Scripting > Run Script File** and choosing a .py file. Single (or even multiple) commands can be executed by using **File > Scripting > Open Command Window** and executing the intended commands. A representative example of the interaction between operating systems, scripts and ANSYS Workbench is shown in Figure 3.1. [13]

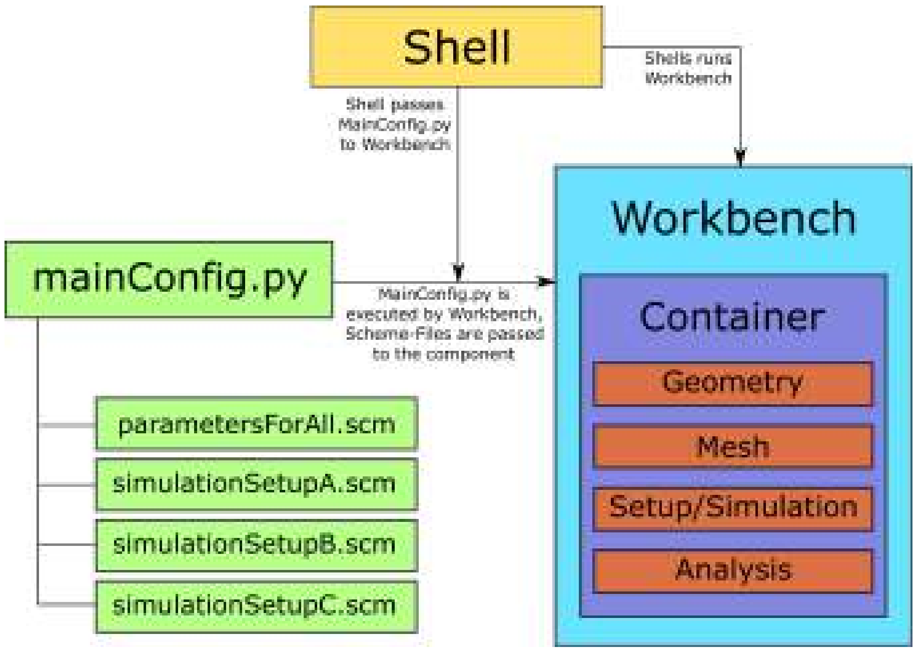


Figure 3.1: Using scripts in ANSYS Workbench environment

3.3.2 Shell Commands and Script Execution

The last section was was related to using Python commands in the Workbench GUI. It is also possible to run Workbench in batch or interactive mode and execute a Python script via Windows/Linux shell. The Windows shell commands are described in Chapter 2.

```
> :: It is possible to call the executable with an absolute path
> "<Installation Path>/<Version>/Framework/bin/<Platform>/RunWB2.exe" -B -R
   "<Any Path>/<Any Script>.py"
> :: Another way is to change directory first and then call the executable
> cd "<Working Directory>"
> "RunWB2.exe" -B -R "<Any Path>/<Any Script>.py"
```

Listing 3.13: Executing a Python script in ANSYS Workbench via Windows shell

The Working Directory specifies the current directory, where the relative paths of the process refers to. The Installation Path is needful to specify the absolute path to the Workbench executable and the Version folder is similar to ”v171” to specify the folder of Version 17.1 and Platform similar to ”win64”. There are also some arguments to specify the process, whereby the arguments can be combined.

Argument	**Process**
-B	Batch Mode: Console Window is shown and GUI not.
-I	Interactive Mode: GUI is shown and Console Window not.
-B -I	Batch and Interactive Mode: Console Window and GUI are both displayed.
-R <Specified File >	Executes the specified script.

Table 3.4: Specifying shell execution

Chapter 4

Using ANSYS Fluent

4.1 Basics of ANSYS Fluent and Scheme

ANSYS Fluent is a software to run numerical simulations in the field of fluid mechanics. The first commercial release of Fluent was in 1983 by the US company Creare Inc. The growth of Fluent led to a separate company with the same name. In 2006 Fluent was bought by the Computer-Aided Engineering (CAE) software company ANSYS. [14]

Similar to other CAE software, Fluent allows communication via interface. This API uses Scheme, which is a dialect of Lisp. Scheme was introduced in 1970 by Guy L. Steele and Gerald Jay Sussmann and it is a functional programming language. [15]

4.2 Scheme in the ANSYS Fluent Environment

In the command prompt the `>` symbol shows a ready prompt. Pressing `Enter` shows the current menu and the actual possible commands. So press `Enter` on the empty prompt shows the commands on the top level: `adapt/`, `define/`, `display/`, `exit`, `close-fluent`, `file/`, `mesh/`, `parallel/`, `plot/`, `report/`, `solve/`, `surface/`, `switch-to-meshing-mode`, `turbo/`, `views/`
The backslash shows that a submenu is available. [16]

Pressing `Enter` in a submenu shows again the available commands. Going back to the superior command is possible by typing `q` or `quit` at the prompt.

An overview of using Text User Interface commands for Fluent is given on the next pages. [15–19]

4.2.1 Syntax

Scheme, as a Lisp dialect, uses the following basic syntax:

```
(command1 argument1 argument2 ...)
```

Listing 4.1: Scheme syntax

Command and variable names are not case sensitive (`a-z`), have to start with a letter and could contain numbers (`0-9`) and the special characters: `+ - * / < > = ? . : % $ ! ~ ^ _`
Comments starts with `;` and end at end of line.

4.2.2 Variables

Basics

Common to other programming languages, Scheme knows local and global variables. A definition of a variable types is not necessary. Values of variables can be of each type. For global definitions of variables use `define`, whereby for redefinitions `set!` should be used. Otherwise the new definiton leads to a new local variable. `let` defines variables as local.

```
> (define x 10) ;Defines a global variable
> (set! x 20) ;Set! for redefinitions of global variables
> (display x) ;Prints the value of the variable on the prompt
20
```

Listing 4.2: Definition and redefinition of global variables

```
> (let ((x 2) (y 3)) (* x y))
6
```

Listing 4.3: Definition of local variables

The value of a variable can be seen by `display`.

The standard Scheme environment in Fluent contains all variables and functions that are defined from Fluent or from the user. To check, if a variable is bounded and assigned in this environment.

```
> (define a1 #t) ;Define the variable as: true
> (define a2 #f) ;Define the variable as: false
> (define c1 #\A) ;Define the variable as the uppercase letter: A
> (define d1 "teststring") ;Define the variable as the string: teststring
> (define e1 'testsymbol) ;Define the variable as the symbol: testsymbol
> (symbol-bound? 'e1 (the-environment)) ;Check, if variable is bound
#t
> (symbol-assigned? 'e1 (the-environment)) ;Check, if variable is assigned (
   contains a value)
#t
```

Listing 4.4: More datatypes in Scheme

RP-Variables in Fluent

All RP-Variables are model-related variables and are defined in the case-Files and the variables can be read and written with:

```
> (%rpgetvar 'nut-kappa) ;Get actual value of nut-kappa (Karman constant)
0.41
> (rpsetvar 'nut-kappa 0) ;Set nut-kappa to the specified value
```

Listing 4.5: RP-Variables

CX-Variables in Fluent

All CX-Variables are related to the Fluent environment and variables can be read and written with:

```
> (cxgetvar 'CMAP-list) ;Get actual value of CMAP-list (colour map)
0.41
> (cxsetvar 'def CMAP "rgb") ;Set CMAP (colour map) to the specified value
```

Listing 4.6: CX-Variables

4.2.3 Mathematical Basic Operations

Some mathematical basic operations can be seen in the following lines. Some operations can be used with more than two arguments.

```
> (define x 10) ;Define variable x as 10
> (define y 20) ;Define variable y as 20
> (define z1 ((+ x y)) ;Calculates x+y= 30
> (- x y) ;Calculate x-y= -10
> (* x y) ;Calculate x*y= 200
> (/ x y) ;Calculate x/y= 0.5
> (/ x y 4) ;Calculate x/y/4= 0.125
> (exp 2) ;Calculate e^2=7.389
> (expt 2 5) ;Calculate 2^5=32
> (log 7.389) ; Calculate log(7.389)=1.9999
> (sin 0) ;Calculate sin(0 rad) = 0, same for cos and tan
> (asin 1) ;Calculate arcsin(1) = pi/2, same for acos and atan
```

Listing 4.7: Mathematical operations

4.2.4 Lists

A List is a fundamental type in Scheme and is a sequence of objects separated by whitespace and enclosed in parentheses.

```
> (define l1 '(a b c)) ;Define l1 as list with elements a, b and c
> (car l1) ;Return the first element of a list
> (cdr l1) ;Return all elements of a list, except the first
> (length l1) ;Return the length (3) of a list
> (list-ref l1 2) ;Return the second element of a list
```

Listing 4.8: List Operations

4.2.5 Conditionals

To allow the execution of specific expressions or not, conditional expressions are necessary and boolean values are used. In Scheme these values are represented as `#t` and `#f`. To determine how a value relates to another, relational and equivalence expressions are needful:

```
> (= a b) ;Check, if a and b are equal, use for numbers
> (< a b) ;Check, if a is less than b, use for numbers
> (<= a b) ;Check, if a is less than or equal to b, use for numbers
> (positive? a) ;Check, if a is a positive number, (also: zero? and negative
   ?)
> (eqv? a b) ;Check, if values of a and b are equal, use for non-numeric
   values
> (eqal? a b) ;Check, if values of a and b are equal, use for lists, vectors
    etc.
```

Listing 4.9: Relational and equivalence expressions

```
> (not a) ;Invert logical value of a
> (and((= 3 3)(eqv? a b))) ;#t, if 3 is equal to 3 AND values of a and b
> (or((= 3 3)(eqv? a b))) ;#t, if 3 is equal to 3 OR values of a and b
```

Listing 4.10: Boolean expressions

```
> (if (< 2 3) 'yes 'no) ;If the test expression is #t, yes is returned
   otherwise no
> (if (< 2 3) ;Begin enables more than one expression per body
>     (begin (+ 10 20) (+ 30 40)) ;True body
>     (begin(+ 1 2) (+ 3 4))) ;False body
```

Listing 4.11: if-else-Statements

Sometimes more than one conditional have to be checked. A good example is a piecewise defined function with multiple intervals. The sytnax of a cond-Statement looks like:
`(cond (test-expr1 body1) (test-expr2 body2) ... (else bodyn))`

```
> (cond
    ((positive? x1) 'executesbody1) ;If x1 is positive, these expressions
       are executed
    ((zero? x1) 'executesbody2) ;If x1 is 0, these expressions are executed
    (else 'executeselsebody)) ;If no previous check is true, these
       expressions are executed, else body is not necessary
```

Listing 4.12: cond-Statements

To enable a control flow in a multiway branch, the value of a variable defines the flow:
`(case controlvar (val1 body1) ((val2 val3) body 2) ... (else bodyn))`

```
> (case x1
    ((0 2 4 6 8) 'evennumber) ;Xheck, if x1 is equal to one of the numbers
       in list
    ((1 3 5 7 9) 'oddnumber) ;Same like before
    (else 'outofrange)) ;Else body is not necessary
```

Listing 4.13: case-Statements

4.2.6 Iterations

Here is a short summary of looping in Fluent Scheme is given:

do-Loops

The sytnax of a do-Loop is:
`(do ((x1 x1start (+ x1 x1delta))) ((> x1 x1end)) ... loop-body ... )`

```
> (do ((x1 1 (+ x1 1))) ;define count-variable, start value and increment
    ((> x1 5)) ;define stop condition
    (display (* x1 10))) ;specify loop body
1020304050
```

Listing 4.14: do-Loops

Argument	Meaning
~a	Next argument (any object) is printed as if by display.
~s	Next argument (any object) is printed as if by write.
~d	Next argument is an integer number.
~f	Next argument is a floating-point number.
\n	Newline character
~%	Newline character
\"	" character

Table 4.1: Format directives and special characters for format-Statements

for-each-Loops vs. map-Function

In Scheme it is possible to do a function for each element of a list. There are two different ways to do the same job, with slight differences. Using the map command stores every return value of all calls. For-each-Loops do not store the return values of each call. The structure of the commands looks as:

```
(map function list1 list 2 ...)
```

```
(for-each function list1 list2 ...)
```

```
> ;Multiply each element of the list by 3
> (map (lambda (x) (* x 3)) '(1 2 3 4 5 6))
(3 6 9 12 15 18)
> ;Do the same as above, but with different output
> (for-each (lambda (x) (display (* x 3))) '(1 2 3 4 5 6))
369121518
```

Listing 4.15: for-each-Loop vs. map-Loop

4.2.7 Formatted Output

To create strings and insert variables, the format procedure is used. The `format` command is like `printf` in C. Instead the % sign the ~ introduces a format directive and the next character specifies the kind of format. Table 4.1 shows the meanings of the specifiers.

```
> (define sVariable1 "bananas")
> (define iVariable1 4)
> (define iVariable2 13)
> (define fpVariable1 0.001)
> (format "Monkeys like ~a." sVariable1) ;Insert sVariable1 in the string
Monkeys like bananas.
> (format "Monkeys like ~a." "bananas") ;Insert "bananas" in the string
Monkeys like bananas.
> (format "Monkeys like ~s." sVariable1) ;String specifier inserts the
   string with " into the string.
Monkeys like "bananas".
> (format "Elephants can reach a height of ~d meters / ~d feet." iVariable1
   iVariable2) ;Usage of more than one format specifier
Elephants can reach a height of 4 meters / 13 feet.
> (format "Elephants can reach a height of ~02d meters / ~04d feet."
   iVariable1 iVariable2) ;Fill zeros on the left side
Elephants can reach a height of 04 meters / 0013 feet.
```

```
> (format "A millisecond is specified as ~f seconds." fpVariable1) ;Use a
   floating point variable with no extra specifications
A millisecond is specified as 0 seconds.
> (format "A millisecond is specified as ~4.3f seconds." fpVariable1) ;
   Specify explicitly 4 digits, 3 after the decimal
A millisecond is specified as 0.001 seconds.
```

Listing 4.16: format-Statements

4.2.8 eval-Statements

The eval-Statement takes the Scheme objects and evaluates it. For using this eval command it is necessary to specify the environment.

```
> (define a 'b) ;Value of a is the symbol b
> (define b 3.14) ;Value of b is 3.14
> (eval a (the-environment)) ;Return the symbol b
b
> (eval (eval a (the-environment)) (the-environment)) ;Return the value of b
    = 3.14
3.14
> (define c (list '+ b 5)) ;Define a list with +, b and 5
> (eval c (the-environment)) ;Evaluate (+ 3.14 5)
8.14
```

Listing 4.17: eval-Statements

In Fluent it is also possible to evaluate a text string. The Fluent Scheme command `ti-menu-load-string` - in combination with the `format` command - enables the creation and execution of commands via text string, which includes the values of variables.

```
> (define a1 27) ;Value of a is 27
> (ti-menu-load-string "(define b1 28)") ;Execute the specified string as
   command
> (display a1) ;Show value of a1
27
> (display b1) ;Show value of b1
28
```

Listing 4.18: ti-menu-load-string-Statements

4.2.9 Aliases

In Fluent command aliases can be defined, to execute commands by typing a (shorter) abbreviation/command.

```
> (alias 'time (lambda () (display (rpgetvar 'flow-time)))) ;Define the
   abbreviation
> time ;Display the flow time by typing time
0
```

Listing 4.19: alias-Statements

Argument	Process
2d	Use two dimensional, single precision solver.
2ddp	Use two dimensional, double precision solver.
3d	Use three dimensional, single precision solver.
3ddp	Usethree dimensional, double precision solver.
-help	Show list of available options.
-i file.jou	Execute specified file (.jou, .scm) in Fluent.
-tx	Specify number of local parallel processes x.
-gpgpu=x	Specify number GPGPUs x per machine.
-hidden	Run Fluent hidden and noninteractively. (Windows only)
-g	Run Fluent without GUI and without graphics.
-gr	Run Fluent without graphics.
-gu	Run Fluent without GUI.
-cc	Open Fluent in Classic Color Scheme.

Table 4.2: Specifying shell execution options of Fluent

4.3 Scripting in ANSYS Fluent

4.3.1 Executing Commands and Scripts with the GUI

Scheme scripts can be executed by selecting **File > Read > Scheme** and choosing a .scm file. Commands can also be executed by using the command prompt.

4.3.2 Shell Commands and Script Execution

CFD simulations in Fluent can be done by shell commands in different operating systems. General commands of Windows and Linux shells are documented on the internet and a short explanation of Windows shell commands are available in Chapter 2.

```
> :: It is possible to call the executable and show help functionality
> cd "<Installation Path>/<Version>/fluent/ntbin/<Platform>"
> fluent.exe ::Using no arguments
> fluent.exe 3ddp -hidden -i predefinedCommands.scm ::Using some arguments
```

Listing 4.20: Executing a Scheme script in ANSYS Fluent via Windows shell

The most important possible arguments are shown in Table 4.2. It is possible to specify properties for solver settings, visualization, parallel processing and input files.

Chapter 5

Using ParaView

5.1 Basics of ParaView and Python

ParaView is an open source application for visualisation and data analysis. Developers are Sandia National Laboratory, Kitware Inc. and Los Alamos National Laboratory and the developing started in 2000. It is able to use ParaView on supercomputers and supports distributed computing. It uses a Python API and the functionality can be expanded by C++ classes. [20–22]

Python was introduced in 1991 and the API is based on the version 2.7.3. It is a interpreted language and it is very easy to read.

5.2 Python in the ParaView Environment

The Python commands and the usage of them is the same as in Chapter 3.2. The slight differences of the two different versions can be read in the documentations. [10, 23]

5.3 Scripting in ParaView

5.3.1 Recording Scripts with the GUI

A good opportunity is to trace scripts. This means that the interaction with the GUI can be recorded (**Tools > Start Trace**). In ParaView this recording leads to the same coding as a human would program the same functionality. So an analysis can be done with GUI which is really simple and then the traced code can be reused.

5.3.2 Executing Commands and Scripts with the GUI

The Python interface can be found by selecting **Tools > Python Shell** in the GUI. The Python interface allows to type in Python commands and execute them and it is also possible to run Python scripts in this interface. Another possibility is to start the PvPython application. Only the interface without the GUI and a window with the visualisation are shown.

5.3.3 Shell Commands and Script Execution

PvPython can also be started from shell of the operating system. Links for commands of Windows and Linux shell can be found in Chapter 2.

```
> :: It is possible to call the executable and run a Python script
> "<Installation Path>/<Version>/Framework/bin/pvpython.exe" "<Any Path>/<
   Any Script>.py"
```

Listing 5.1: Executing a Python script in PvPython via Windows shell

Bibliography

[1] Stefan Pirker. *Numerical Methods in Fluid Dynamics, Lecture Notes.* Johannes Kepler University, March 2017.

[2] Andre Bakker. *Applied Computational Fluid Dynamics.* FLUENT, 2002-2006.

[3] Homepage ANSYS. `http://www.ansys.com`, June 2017.

[4] Homepage STAR-CCM+. `http://mdx.plm.automation.siemens.com`, June 2017.

[5] Homepage OpenFOAM. `http://www.openfoam.com`, June 2017.

[6] Homepage SU2. `http://su2.stanford.edu`, June 2017.

[7] Thomas Theis. *Einstieg in Python.* Rheinwerk, Bonn, 4 edition, 2016.

[8] ANSYS, Canonsburg. *ANSYS AIM and Workbench Scripting Guide*, 17.1 edition, April 2016.

[9] IronPython Community. IronPython Documentation. `http://ironpython.net/documentation/dotnet`, June 2017.

[10] Python Software Foundation. Python 2.7.4 Documentation. `https://docs.python.org/release/2.7.4`, June 2017.

[11] Bodenseo. Python course. http://www.python-course.eu/index.php, June 2017.

[12] Anthony Reid. Python Naming Conventions. `https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html`, June 2017.

[13] ANSYS, Canonsburg. *ANSYS Fluent in ANSYS Workbench Users Guide*, 17.1 edition, April 2016.

[14] University of Kentucky Center for Computational Sciences. A Brief History of Fluent. `https://www.ccs.uky.edu/UserSupport/SoftwareResources/Fluen/`, May 2017.

[15] R. Kent Dybvig. *The Scheme Programming Language.* The MIT Press, 4 edition, 2009.

[16] ANSYS, Canonsburg. *ANSYS Fluent Users Guide*, 17.1 edition, April 2016.

[17] Mirko Javurek. *Fluent Scheme Documentation.* Johannes Kepler University, Linz, August 2015.

[18] University of Illinois Computational Science and Engineering. ANSYS Fluent CFD - Text User Interface & Scheme for Automation. `https://uiuc-cse.github.io/me498cm-fa15/lessons/fluent/handout-tui-scheme.pdf`, October 2015.

[19] ANSYS, Canonsburg. *ANSYS Fluent Text Command List*, 17.1 edition, April 2016.

[20] Homepage ParaView. `https://www.paraview.org`, June 2017.

[21] Utkarsh Ayachit. *The ParaView Guide*. Kitware Inc., Clifton Park, 5.0 edition, November 2016.

[22] Kenneth Moreland. *The ParaView Tutorial*. Kitware Inc., Albuquerque, 5.2 edition, November 2016.

[23] Python Software Foundation. Python 2.7.3 Documentation. `https://docs.python.org/release/2.7.3`, June 2017.

Appendix A

Example to do a scripted Simulation and Analysis

The following scripts are based on the structure of Figure 3.1. The IronPython file which is executed, does the looping and sends the setup and commands, which are specified in both Scheme files. The parameters scheme file is executed first in Fluent. Then the remaining commands are sent to Fluent. The case, the results and the residual histories are saved. The results are opened and analysed with ParaView.

The following scripts assume that a Workbench Project (.wbpj) was created, the Analysis System “Fluid Flow (Fluent)” was used, the Geometry was imported or created and the mesh was generated. Naturally the way without Workbench can be used, but to demonstrate and show the interaction between Workbench and Fluent the way with the Framework is displayed.

A.1 Script for ANSYS Workbench

```
##########################################################################
#---------------------------------HEADER----------------------------------
# Author: Lukas Muttenthaler
# Date: May 2017
# Description: A simulation case should be done with three different
# velocities at the boundary. The results and residuals should be stored.
# The files with parameters and commands are here:
# C:/ProjectPath/StudyA_SimParameters.scm
# C:/ProjectPath/StudyA_SimCommands.scm
# The Workbench Project is here:
# C:/ProjectPath/MyProject.wbpj
##########################################################################
#------------------------------PRE OPERATIONS-----------------------------

#Import necessary Modules
import os

#Set Path and Project
myPath ="C:/ProjectPath/"
myProj = "MyProject.wbpj"

#Project operations
Open(FilePath=os.path.join(myPath, myProj)) #Opens the Project
```

```
system1 = GetSystem(Name="FFF") #Get the System with Name "FFF"
setup1 = system1.GetContainer(ComponentName="Setup") #Get "Setup"-Container

#File Name Parts
parsFilename = 'SimParameters' #Specify Filename of parameters file
prefixFilename = 'StudyA' #Prefix of File with Commands
mainFilename = 'SimCommands' #File with Commands for Fluent
saveFilename = ['SimA','SimB','SimC'] #Naming Parts for Saving File
suffixFilename = '.scm' #Ending of the Filename with Scheme Commands

velocityMaximum = [2,4,6] #Simulate with different Velocities; [m/s]

count1StartIndex = 0 #Start Index of the Loop
count1StopIndex = 2  #Stop Index of the Loop

##############Load file with important parameters############
actualPars = prefixFilename+parsFilename+suffixFilename #Join together
   string parts
parFile = open(actualPars,'r') #Opens the specified file in "read"-Mode and
   get the file-Object
parCommands = parFile.read() #Reads the whole file
parFile.close() #Close File

##############Load file with commands for Fluent#############
actualCalc = prefixFilename+mainFilename+suffixFilename
commandsFile = open(actualCalc,'r')
commands = commandsFile.read()
commandsFile.close()

##########################################################################
#-----------------------LOOP THROUGH THE SIMULATIONS----------------------
for count1 in range(count1StartIndex, (count1StopIndex+1)):
    setup1.Reset() #Resets the Fluent Container
    setup1.Refresh() #Refreshes Input

    ##############Run Fluent and set properties#############
    setup1.GetFluentLauncherSettings().ShowLauncher = False #Showing Fluent
       Launcher at Startup of Fluent?
    setup1.GetFluentLauncherSettings().Precision = 'Double' #Precision "
       Single" or "Double"
    setup1.GetFluentLauncherSettings().RunParallel = True #Using Parallel
       Cores
    setup1.GetFluentLauncherSettings().NumberOfProcessors = 20 #Number of
       used Cores
    setup1.Edit() #Open Fluent

    ##############Send changing Parameters#############
    setup1.SendCommand(Command='(define calcCase %s)' %str(saveFilename[
       count1]))
    setup1.SendCommand(Command='(define velMax %d)' %velocityMaximum[count1
       ])

    ##############Set important variables in Fluent############
    #Send the commands specified in the file as string to Fluent:
    setup1.SendCommand(Command=parCommands)

    ##############Send commands to Fluent#############
```

```
     #Send the commands specified in the file as string to Fluent:
    setup1.SendCommand(Command=commands)

    setup1.Exit() #Close Fluent
#-----------------------------END OF SCRIPT------------------------------#
```
Listing A.1: Workbench Commands (IronPython)

A.2 Scripts for ANSYS Fluent

A.2.1 Script of Important Parameters

```
;###########################################################################
;#--------------------------------HEADER--------------------------------#
;# Author: Lukas Muttenthaler
;# Date: May 2017
;# Description: Set different important Parameters and combine them in
;# this file
;# The results has to be stored here:
;# C:/ProjectPath/Results/DoneIn2017_SimA also for SimB and Sim C
;###########################################################################
;#------------------------------Parameters-------------------------------#

;-----Limits of the simulated state values
(define pAbsMin 70000)  ;Min. Absolute Pressure
(define pAbsMax 120000) ;Max. Absolute Pressure
(define viscRatioTurbMax 100)   ;Maximum Turbulent Viscosity Ratio
(define energyKinTurbMin 1e-14) ;Minimum Turbulent Kinetic Energy
(define dissRateTurbMin 1e-20)  ;Minimum Turbulent Dissipation Rate

;-----Set Abortion Conditions
(define numbOfIterations 1500)     ;Number of max. Iterations
(define convCriterion 1e-5) ;Convergence Criterion

;-----Turbulent Parameters at Boundaries
(define dHBig 0.02) ;Hydraulic Diameter of the Big Port
(define IBig 5.5)   ;Turbulence Intensity of the Big Port

(define dHSmall 0.015) ;Hydraulic Diameter of the Small Port
(define ISmall 5)      ;Turbulence Intensity of the Small Port

(define dHConn 0.25) ;Hydraulic Diameter of the Connection Port
(define IConn 7.5)   ;Turbulence Intensity of the Connection Port

;-----Initial Values for Turbulence
(define ke-turbulence-k 0.02)        ;Turbulent Kinetic Energy [m^2/s^2]
(define ke-turbulence-epsilon 0.025) ;Turbulent Dissipation Rate [m^2/s^3]

;-----Specify the Saving
(define saveDirectory "C:/ProjectPath/Results")  ;Path, where the results
   should be stored
(define calcCasePrefix "/DoneIn2017_") ;Prefix of the saved Files
(define savecasdat 1)  ;Save .cas and .dat-File
(define saveensight 1) ; Save EnSight-Gold-Files
;#-----------------------------END OF SCRIPT-----------------------------#
```

Listing A.2: Setting Parameters in Fluent (Scheme)

A.2.2 Script of Fluent Commands

```
;##########################################################################
;#--------------------------------HEADER---------------------------------#
;# Author: Lukas Muttenthaler
;# Date: May 2017
;# Description: This file contains the commands for the setup of an easy
;# kepsilon-Standard-Model simulation, with two velocity inlets, one
;# pressure outlet. The fluid should be an hydraulic oil.

;##########################################################################
;#-------------------------------GENERAL---------------------------------#

/define/models/solver/pressure-based yes ;Pressure-Based Solver
/define/model/steady ;Time: Steady or Transient Analysis
/define/operating-conditions/gravity yes 0 0 9.81 ;Gravity Xomponents (x,y
   and z)
/define/model/energy no ;Energy Model

;##########################################################################
;#-------------------------------MODELS----------------------------------#

/define/models/viscous/ke-standard yes ;Set kepsilon-Standard-Model
/define/models/viscous/near-wall-treatment/enhanced-wall-treatment yes ;Near
    Wall Treatment

;-----Options of the k-epsilon-Standard-Model
/define/models/viscous/buoyancy-effects yes ;Consider Buoyancy Effects?
/define/models/viscous/curvature-correction yes ;Consider Curvature
   Correction?
(rpsetvar (quote curvature-correction-coefficient) 1.) ;Curvature Correction
    Coefficient
/define/models/viscous/turbulence-expert/kato-launder-model yes ;Production
   Kato-Launder
/define/models/viscous/turbulence-expert/production-limiter yes 10 ;
   Production Limiter with Clip Factor

;-----Model Constants of the k-epsilon-Standard-Model
(rpsetvar (quote kecmu) 0.0845) ;GUI: Cmue-Factor
(rpsetvar (quote kec1) 1.42) ;GUI: C1-Factor
(rpsetvar (quote kec2) 1.68) ;GUI: C2-Factor
(rpsetvar (quote kesigk) 1) ;GUI: TKE Prandtl Number
(rpsetvar (quote kesige) 1.2) ;GUI:TDR Prandtl Number

;##########################################################################
;#------------------------------MATERIALS--------------------------------#

;-----Copy Material from Data Base
/define/material/copy fluid engine-oil ;Copy fluid "engine-oil"
/define/material/copy solid steel ;Copy solid "steel"

;Arguments for copy material engine-oil to oil are:
; Density Change? /  Method = constant / Value 879 kg/m^3 /
; Specific Heat [J/(kgK)] / Thermal Conductivity [W/(m*K)] /
; Dynamic Viscosity [kg/(m*s)] / Molecular Weight [kg/(kg*mol)] /
; Thermal Expansion coefficient [1/K] / Speed of Sound [m/s]
/define/material/change-create engine-oil oil yes constant 879 yes constant
```

```
   1885 no yes constant 0.0404 no yes 0.00065 yes constant 1306 no

;######################################################################
;#------------------CELL ZONES and BOUNDARY CONDITIONS-------------------#

;Set the types of boundary conditions to the right zones
/define/boundary-conditions/modify-zones/zone-type connection pressure-
   outlet
/define/boundary-conditions/modify-zones/zone-type volume1 fluid
/define/boundary-conditions/modify-zones/zone-type wall1 wall
/define/boundary-conditions/modify-zones/zone-type wall2 wall
/define/boundary-conditions/modify-zones/zone-type int1 interior
/define/boundary-conditions/modify-zones/zone-type portbig velocity-inlet
/define/boundary-conditions/modify-zones/zone-type portsmall velocity-inlet

/define/boundary-conditions/set/fluid volume1 material yes oil () ;Set the
   right fluid type

;Velocity Inlet Arguments: Name / Velocity Specification Method (Magnitude&
   Direction?/Components?/Magnitude, Normal to Boundary?) / Ref. Frame:
   Absolute? / Profile for Vel. Magn.? / Vel. Magn. Value / Profile and
   Value for Supersonic-Initial Gauge Pressure ? Value / Turbulent
   Specification Method (Mod. Turbulent Viscosity?/Intensity and Length
   Scale?/Turbulent Viscosity Ratio?/Intensity and Hydraulic Diameter?) /
   Turbulent Intensity Value / Hydraulic Diameter Value
/define/boundary-conditions/velocity-inlet suctionportsmall no no yes yes no
    velMax no 0 no no no yes ISmall dHSmall ;BC of Small Suction Port
/define/boundary-conditions/velocity-inlet suctionportbig no no yes yes no
   velMax no 0 no no no yes IBig dHBig ;BC of Big Suction Port

;Pressure Outlet Arguments: Name / Backflow Reference Frame Absolute? / Use
   Profile for Gauge Pressure ? / Gauge Pressure Value / Backflow
   Specification Method (Dir. Vector?/Normal to Boundary?) / Turbulent
   Specification Method (Mod. Turbulent Viscosity?/Intensity and Length
   Scale?/Turbulent Viscosity Ratio?/Intensity and Hydraulic Diameter?) /
   Turbulent Backflow Intensity Value / Backflow Hydraulic Diameter Value /
   Radial Equili. Pressure Distribution? / Average Pressure Specification? /
    Specify Targeted Mass Flow?
/define/boundary-conditions/pressure-outlet conn2calmingchamb yes no 101325
   no yes no no no yes IConn dHConn no no no

;Wall Arguments: Name / Change Wall Motion (?, "motion-bc-stationary" or "
   motion-bc-moving") / Shear Boundary Condition (?, shear-bc-noslip, shear-
   bc-specular or shear-bc-spec-shear) / Profile and Height for Wall
   Roughness ? Value / Profile and Constant for Wall Roughness ? Value
/define/boundary-conditions/wall wall1 yes motion-bc-stationary yes shear-bc
   -noslip no 0 no 0.5
/define/boundary-conditions/copy-bc wall wall2 () ;BC of inflationfaceswall;
    Copy BC from wall1 to wall2


;######################################################################
;#--------------------------SOLUTION METHODS-----------------------------#

;Pressure-Velocity-Coupling: 20 = SIMPLE (def.), 21 = SIMPLEC, 22 = PISO, 24
    = Coupled
/solve/set/p-v-coupling 20
```

```
;Spatial Gradient Discretization: Green-Gauss Node Based?/Least-Squares Cell
    Based? (def.) otherwise Green-Gauss Cell Based
/solve/set/gradient-scheme no yes
;Spatial Pressure Discretization: 10 = Standard, 11 = Linear, 12 = Second
   Order (def.), 13 = Body Force Weighted, 14 = PRESTO!
/solve/set/discretization-scheme/pressure 12
;Spatial Momentum Discretization: 0 = First Order Upwind, 1 = Second Order
   Upwind (def.), 2 = Power Law, 4 = QUICK, 5 = Third-Order MUSCL
/solve/set/discretization-scheme/mom 1
;Spatial Turbulent Kinetic Energy Discretization: 0 = First Order Upwind (
   def.), 1 = Second Order Upwind, 2 = Power Law, 4 = QUICK, 5 = Third-Order
    MUSCL
/solve/set/discretization-scheme/k 0
;Spatial Turbulent Kinetic Energy Discretization:
/solve/set/discretization-scheme/epsilon 0

;Warped-Face Gradient Correction
/solve/set/warped-face-gradient-correction/enable yes yes
 ;High Order Term Relaxation
/solve/set/high-order-term-relaxation/enable no
;###########################################################################
;#--------------------------SOLUTION CONTROLS-----------------------------#

;-----Under-Relaxation Factors
/solve/set/under-relaxation/pressure 0.3 ;Pressure (0.3 = def.)
/solve/set/under-relaxation/density 1 ;Density (1 = def.)
/solve/set/under-relaxation/body-force 1 ;Body-Force (1 = def.)
/solve/set/under-relaxation/mom 0.7 ;Momentum (0.7 = def.)
/solve/set/under-relaxation/k 0.8 ;Turbulent Kinetic Energy (0.8 = def.)
/solve/set/under-relaxation/epsilon 0.8 ;Turbulent Dissipation Rate (0.8 =
   def.)
/solve/set/under-relaxation/turb-viscosity 1 ;Turbulent Viscosity (1 = def.)

;Limits Arguments: Min. & Max.  Abs. Pressure / Min. Turbulent Kinetic
   Energy / Minimum Turbulent Dissipation Rate/Max. Turbulent Viscosity
   Ratio / Default Values: 1 5e10 1e-14 1e-20 1e5
/solve/set/limits pAbsMin pAbsMax viscRatioTurbMax energyKinTurbMin
   dissRateTurbMin viscRatioTurbMax

/solve/monitors/residual/criterion 0 ;Convergence Criterion: 0 = absolute, 3
    = none
/solve/monitors/residual/convergence-criteria convCriterion convCriterion
   conveCriterion convCriterion convCriterion convCriterion ;Limit of
   Convergence Criteria; continuity, x-velocity, y-velocity, z-velocity, k,
   epsilon
/solve/monitors/residual/n-save 100000 ;Number of iterations to store

;###########################################################################
;#-----------------------SOLUTION INITIALIZATION---------------------------#

;###################Standard Initialization##################
/solve/initialize/reference-frame absolute ;absolute or relative to Cell
   Zone
/solve/initialize/set-defaults pressure 0 ;Gauge Pressure [Pa]
/solve/initialize/set-defaults x-velocity 0 ;Velocity x-Component [m/s]
/solve/initialize/set-defaults y-velocity 0 ;Velocity y-Component [m/s]
/solve/initialize/set-defaults z-velocity 0 ;Velocity z-Component [m/s]
```

```
/solve/initialize/set-defaults k ke-turbulence-k ;Turb. Kin. Ener. [m^2/s^2]
/solve/initialize/set-defaults epsilon ke-turbulence-epsilon ;Turbulent
   Dissipation Rate [m^2/s^3]

/solve/initialize/initialize-flow ;do the Initialization

;###########################################################################
;#---------------------------RUN CALCULATION--------------------------------#
/solve/set/reporting-interval 1 ;Interval between reporting of convergence
   monitors
/define/profiles/update-interval 1 ;Interval between updates of Dynamic
   Profiles of Boundary Conditions

/solve/iterate numbOfIterations ;Run iterations (with max. number of it)

;###########################################################################
;#-----------------------SAVE SOLUTION AND RESIDUALS-----------------------#

/file/binary-files no ;Write Files as Binaries

;Define variables that should be saved
(define saveVariables "pressure x-velocity y-velocity z-velocity density
   viscosity-lam viscosity-turb turb-kinetic-energy turb-diss-rate dynamic-
   pressure total-pressure velocity-magnitude production-of-k viscosity-eff
   ()")

;Save Case and Data in .cas and .dat-Files with specified file name
(case savecasdat
        ((1)    (ti-menu-load-string (format #f "/file/data-file-options ~a"
            saveVariables))
                        (ti-menu-load-string (format #f "/file/write-case-
                           data ~a~a~a" saveDirectory calcCasePrefix
                           calcCase)))
)

;Save Data in EnSight-Files with specified file name
;Arguments: Filename / Variables to safe / Binary File? / Cell Zone ID/Name
   / Interior Zone Surfaces (empty) / Cell-centered?
(case saveensight
        ((1)    (ti-menu-load-string (format #f "/file/export/ensight-gold ~
           a~a~a ~a () no 2 () () no" saveDirectory calcCasePrefix calcCase
           saveVariables)))
)

;-----Save Residuals
(define port)
(set! port (open-output-file (format #f "~a~a~a_Residuals.dat" saveDirectory
    calcCasePrefix calcCase)))
(format port "Iteration ~a" (residual-history "iteration")) (newline port)
(format port "Continuity ~a" (residual-history "continuity")) (newline port)
(format port "x-Velocity ~a" (residual-history "x-velocity")) (newline port)
(format port "y-Velocity ~a" (residual-history "y-velocity")) (newline port)
(format port "z-Velocity ~a" (residual-history "z-velocity")) (newline port)
(format port "Turbulent Kinetic Energy ~a" (residual-history "k")) (newline
   port)
(format port "Turbulent Dissipation Rate ~a" (residual-history "epsilon")) (
   newline port)
```

```scheme
(close-output-port port)
;#-----------------------------END OF SCRIPT-------------------------------#
```

Listing A.3: Fluent specific and remaining Commands (Scheme)

A.3 Script for ParaView

```
########################################################################
#--------------------------------Header----------------------------------
# Author: Lukas Muttenthaler
# Date: May 2017
# Description: This file opens the stored results of the simulations,
# imports the specified variables, creates several cutting planes,
# looks from above on this plane, sets the scalar bar and saves
# a .png picture of this visualisation.
# The file does this for all simulation cases.

########################################################################
#-----------------------------Import Packages----------------------------
from paraview.simple import *
import os
import math
import matplotlib.cm

########################################################################
#------------------------Creating a Slice Function-----------------------
def sliceWithPlane(data, sliceOrigin, sliceNormal, sliceOffset, sliceCrinkle
   ):
 slice = Slice(Input=data)       #Specify the Data which should be sliced
 slice.SliceType = 'Plane'       #Specified by Plane
 slice.SliceOffsetValues = sliceOffset #Offset
 slice.SliceType.Origin = sliceOrigin  #Point included by the Plane
 slice.SliceType.Normal = sliceNormal  #Vector Normal to the Plane
 slice.Crinkleslice = sliceCrinkle     #Shows an flat or crinkled Plane
 return slice

########################################################################
#--------------------Define Cases, Planes and FieldVars------------------

mainPath = "C:/MyProject" #Define the Main Path
caseLoadArray = ['/Results/DoneIn2017_SimA','/Results/DoneIn2017_SimB','/
   Results/DoneIn2017_SimC'] #Define which Cases should be loaded
caseFileExtension = '.encas' #Specifiy which Format should be loaded

caseSavingArray = ['/Analysis/DoneIn2017_SimA','/Analysis/DoneIn2017_SimB','
   /Analysis/DoneIn2017_SimC'] #Define the naming of the Pictures, that are
   created
picSavingFileExtension = ['.png'] #Specifiy which Picture Formats should be
   stored

planeArray = ['A100_XZPlaneBigPort','A110_XYPlaneSmallPort','
   A100_XYPlaneConnection'] #Slicing Planes
fieldVarArray = ['viscosity_lam','viscosity_turb','turb_diss_rate'] #
   Variables of that Pictures should be stored
fieldVarArrayOtherNamings = ['visc_lam','visc_turb', 'turb_diss_rate'] #
   Naming of the Variables in the Pictures

########################################################################
#------------------------Loop Simulation Results-------------------------
for iCase in range(0,len(caseLoadArray)):
 #------------------Load Data and set Field Variables--------------------
 data1 = EnSightReader(CaseFileName = mainPath + "/" + caseLoadArray[iCase]
    + caseFileExtension) #Load Data
data1.CellArrays = [] #Specify which Cell-Based Field Variables are loaded
data1.PointArrays = fieldVarArray  #Specify which Point-Based Field
    Variables are loaded

#--------------------Define Origin Points and Vectors--------------------
#---Definition of Origin Points of Cutting Planes
middleBigPort = [-0.08, 0.0, 0.06]
middleSmallPort = [-0.08, 0.0, 0.02]
middleConnection = [-0.08, 0.2, 0.0]

#---Definition of Vectors of Cutting Planes
vectorX = [1.0, 0.0, 0.0] #yz-Plane
vectorY = [0.0, 1.0, 0.0] #xz-Plane
vectorZ = [0.0, 0.0, 1.0] #xy-Plane

sliceOffset = 0.0    #Offset of the plane
sliceCrinkle = False #Showing Crinkles or not?

#########################################################################
#--------------------------Loop Cutting Planes--------------------------
for iPlane in range(0,len(planeArray)):
 #--------------------------Define Cutting Planes-------------------------
 if iPlane == 0:
  slice1 = sliceWithPlane(data1,  middleBigPort, vectorY, sliceOffset,
      sliceCrinkle)
  camPosition = [0,-1,0]
  camViewUp = [0,0,-1]
 elif iPlane == 1:
  slice1 = sliceWithPlane(data1,  middleSmallPort, vectorZ, sliceOffset,
      sliceCrinkle)
  camPosition = [0,0,-1]
  camViewUp = [0,1,0]
 elif iPlane == 2:
  slice1 = sliceWithPlane(data1, mmiddleConnection , vectorZ, sliceOffset,
      sliceCrinkle)
  camPosition = [0,0,-1]
  camViewUp = [0,1,0]

 Hide(data1)
 Show(slice1)

 #----------Specify the Viewing Parameters and Display Properties----------
 view1 = GetActiveView()    #Get current View
 view1.Background = [1,1,1] #White
 view1.OrientationAxesVisibility = True    #Visibility of Orientation Axes?
 view1.OrientationAxesLabelColor = (0,0,0) #Black Colored Label
 view1.ViewSize = [1920, 1200]             #Width and Height in Pixels

 #--------------------------Get Display Properties------------------------
 dp1 = GetDisplayProperties() # Get the properties of current Display

 #----------------------Specify the Camera Parameters----------------------
 camera = GetActiveCamera()
 camera.SetFocalPoint(0,0,0)
 camera.SetPosition(camPosition)
 camera.SetViewUp(camViewUp)
camera.SetViewAngle(30)

Hide3DWidgets(proxy=slice1.SliceType) #Hides the Slicing Plane

view1.ResetCamera()

#########################################################################
#--------------------------Loop Field Variables--------------------------
for iFieldVar in range(0,len(fieldVarArray)):

 ColorBy(dp1, ('POINTS', fieldVarArray[iFieldVar]))  #Set Scalar Coloring
 dp1.RescaleTransferFunctionToDataRange(True, False) #Rescale Color and/or
     Opacity Maps used to include current Data Range
 dp1.SetScalarBarVisibility(view1, True)             #Show Color Bar
 colorMap1 = GetColorTransferFunction(fieldVarArray[iFieldVar])

 #--------------------Specify the Color Map Parameters--------------------
 [minRange1, maxRange1] = slice1.GetDataInformation().
    GetPointDataInformation().GetArrayInformation(fieldVarArray[iFieldVar
    ]).GetComponentRange(0)

 #Color Map Specification
 colorMap1 = GetColorTransferFunction(fieldVarArray[iFieldVar])
 colorMap1.Discretize = 0
 colorMap1.UseLogScale = 0
 colorMap1.LockDataRange = 0

 #"Rainbow" Colormap from matplotlib
 N = 10
 stepWidth = 1
 cmap = matplotlib.cm.get_cmap('rainbow')
 colorMap1.ColorSpace = "RGB"
 RGB1 = [None]*4*(N+1)

 #Create Array with specified Points in the Color Map
 for ii in xrange(0, N+stepWidth, stepWidth):
  RGBA1 = cmap(ii/float(N)) #Colormap is normalized to Values from 0 to 1
  RGB1[4*ii] = minRange1+ii*(maxRange1-minRange1)/float(N) #Write Scalar
     Points to the Array
  RGB1[4*ii+1:4*ii+4] = RGBA1[0:3] #Write RGB-Values to the Array

 colorMap1.RGBPoints = RGB1

 #-------------------Specify the Scalar Bar Parameters--------------------
 scalarBar1 = GetScalarBar(colorMap1,view1) #Get current Bar
 dp1.SetScalarBarVisibility(view1,True) #Show Scalar Bar

 scalarBar1.Title = fieldVarArrayOtherNamings[iFieldVar] #Title
 scalarBar1.ComponentTitle = ""    #Additional Text to the Title
 scalarBar1.TitleColor = [0.0, 0.0, 0.0] #Color of the titel: Black
 scalarBar1.TitleFontSize = 10        #Font Size of the Title of the
    Legend
 scalarBar1.DrawNanAnnotation = False #Annotation for NaN?
 scalarBar1.AspectRatio= 20           #Width of the Color Bar
 scalarBar1.AutomaticLabelFormat = False #Auto Format of Labels?
 scalarBar1.RangeLabelFormat = "%#-3.5e" #Label Format of the First and
    Last Value
```

```
   scalarBar1.LabelFormat = "%#-3.5e"   #Label Format except the First and
      Last Value
   scalarBar1.LabelFontSize = 8          #Fontsize
   scalarBar1.DrawTickMarks = True       #Show Main Tick Marks?
   scalarBar1.DrawSubTickMarks = False   #Show Sub Tick Marks?
   scalarBar1.DrawTickLabels = True      #Show Labels of the Main Tick Marks?
   scalarBar1.LabelColor = [0.0, 0.0, 0.0] #Color of the Labels: Black
   scalarBar1.NumberOfLabels = 7         #Number of Ticks
   scalarBar1.Position = [0.025, 0.225] #Rel. Position of the Bar: Left,
      Bottom Corner
   scalarBar1.Position2 = [0.2, 0.75]   #Rel. Position of the Bar: Right,
      Top Corner
   scalarBar1.Orientation = "Vertical"  #Orientation
   scalarBar1.TitleJustification = "Right" #Alignment of Title: Left,
      Centered and Right
   scalarBar1.Visibility = 1            #Show or Hide Scalar Bar

   Render() #Update View

   #----------------------Create Subfolder if necessary---------------------
   try:
    os.stat(mainPath + "/" + subDirArray[iCase])
   except:
    os.mkdir(mainPath + "/" + subDirArray[iCase])

   #------------------------------Save Pictures------------------------------
   SaveScreenshot(mainPath + "/" + caseSavingArray[iCase] + "_" + planeArray
      [iPlane] + "_"  + fieldVarArrayOtherNamings[iFieldVar] +
      picSavingFileExtension[0]) #Store Screenshot to specified Path

   dp1.SetScalarBarVisibility(view1, False) #Show Color Bar

  Hide(slice1)   #Hide Slice Plane
  Render()       #Update View
  Delete(slice1) #Delete Slice after completing a specific Viewing
  del slice1     #Delete the Variable
 Delete(data1) #Delete Data after Postprocessing of each Simulation Result
 del data1     #Delete the Variable
#-----------------------------END OF SCRIPT------------------------------#
```

Listing A.4: Postprocessing Commands for ParaView (Python)