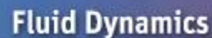
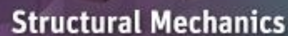
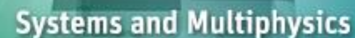
ANSYS®

Appendix A: Scripting and Automation

16.0 Release







Introduction to ANSYS CFX

Realize Your Product Promise®

ANSYS

Overview

- Introduction
- CFX User Environment (CUE) architecture
- State and Session Files
- Introduction to Perl
- CCL and Perl
- “Power Syntax”
- Perl subroutines
- Macros

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Introduction

- **Need for scripting and automation**
  - Increase productivity by simplifying repetitive tasks
  - Standardize practices
  - Save and re-use data
  - …

 © 2015 ANSYS, Inc. ANSYS Confidential

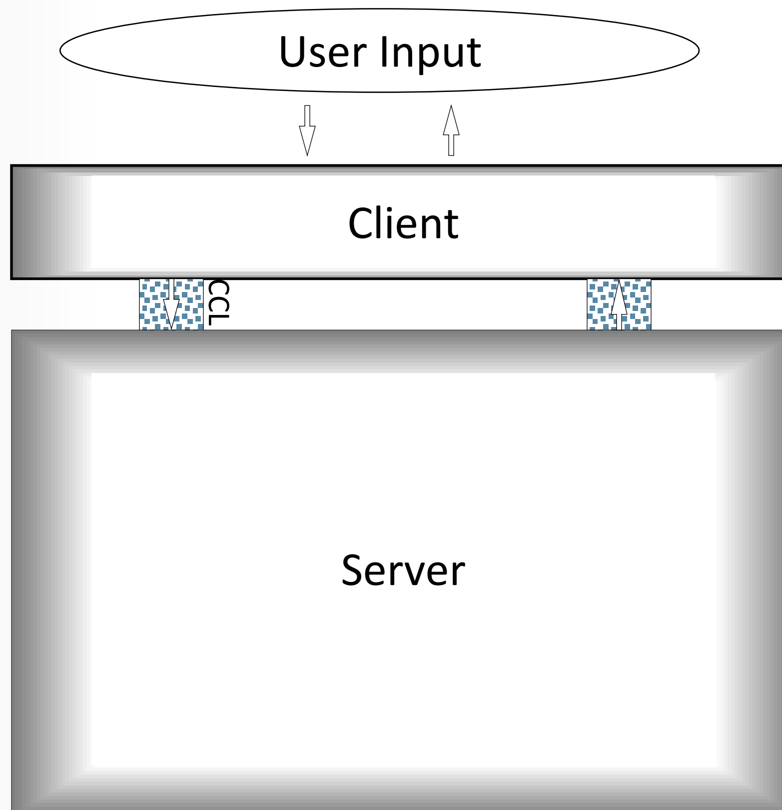
CFX User Environment

CUE is the common development environment for all CFX products.

CUE applications employ a client-server architecture.

The user interfaces with the "client", while the "server" processes the data.

The most common client is a graphical user interface, but line and batch interfaces also exist.



© 2015 ANSYS, Inc.

ANSYS Confidential

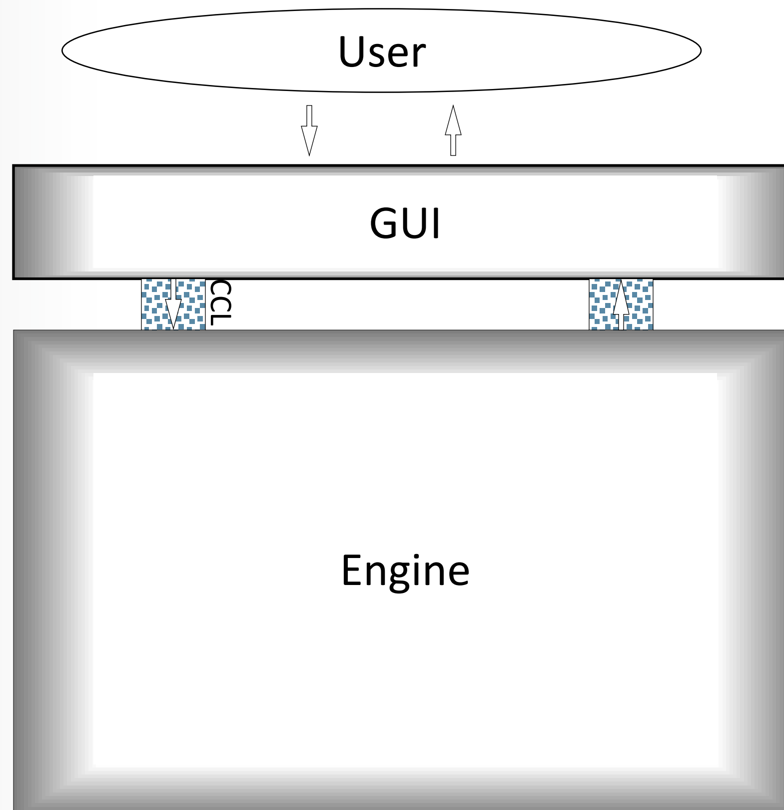
ANSYS

GUI Mode

Default mode of operation

Graphical client driven by user input

User loads results, states, runs sessions and macros



 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Line Input Mode

Launch session from command line or a script by specifying '-line' flag

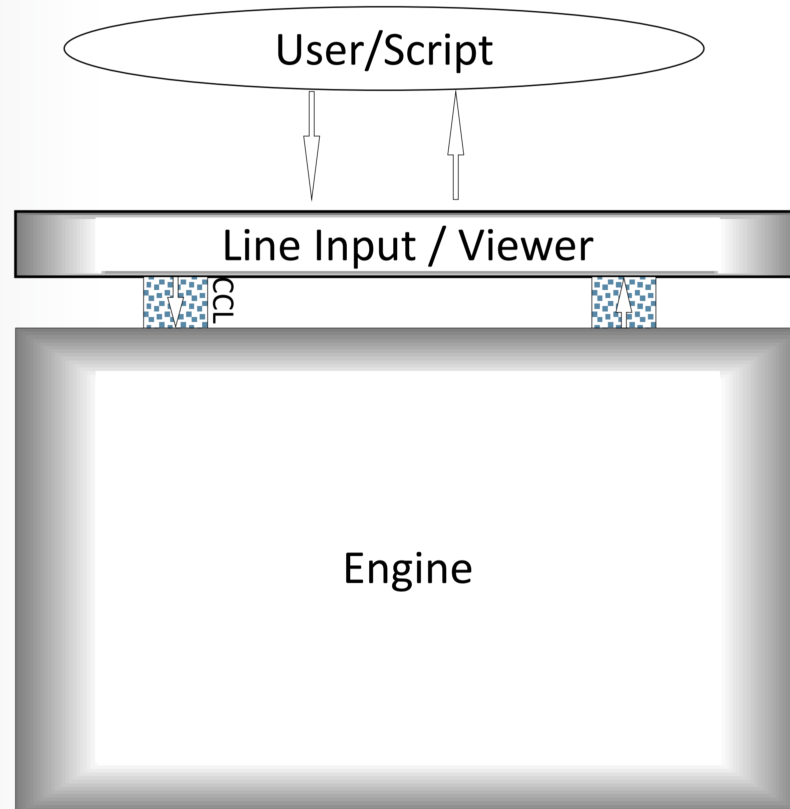
- e.g. `>cfx5post -line`

Client includes viewer and a command line input

CCL objects and commands are input one line at a time

Allows interactive scripts with control outside of script

Line input modes exist for TurboGrid, Pre, Post, Solver (solver uses –ccl flag)



© 2015 ANSYS, Inc.

ANSYS Confidential

Batch Mode

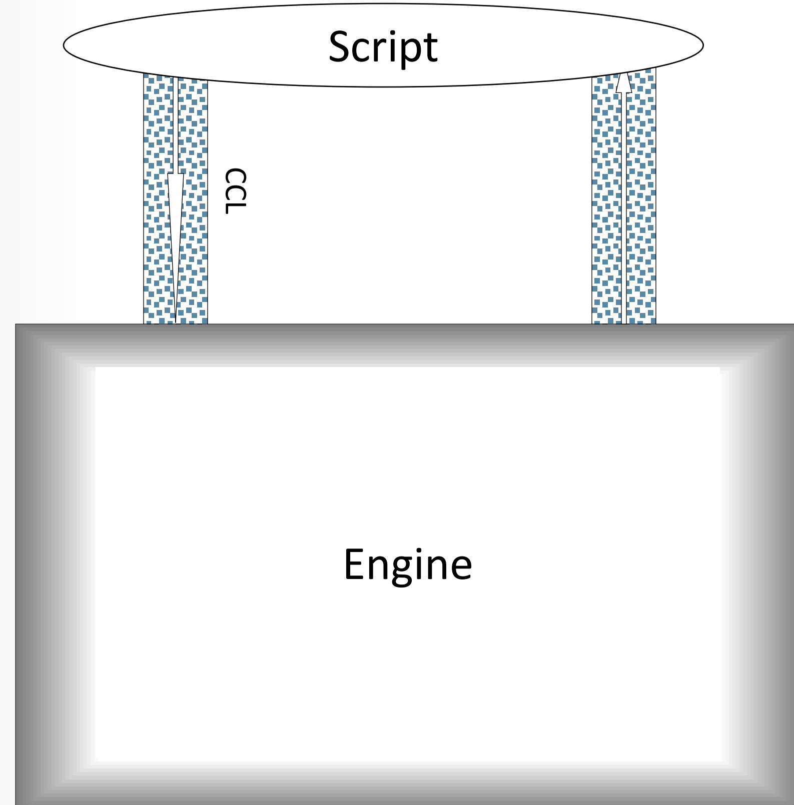
Closed session (not interactive) launched from command line or script

- Direct to engine

No viewer

Session file specified at run time

- Session file may include interactive commands, load states, results, etc.
- **Must** end with a >quit statement



© 2015 ANSYS, Inc.

ANSYS

Session Files

- **Session files contain a list of commands and CCL objects**
  - Can record the commands executed during a session to a file and then play back the file at a later date or in batch mode.
  - Can write/modify session files in a text editor
  - Produced in Pre, Post, TurboGrid
  - Session files can perform actions, for example Input/Output

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

State Files

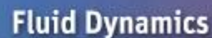
- **State files are a snap-shot of the current state of all objects**
  - Can be created to save or load a number of objects
  - Contain CCL objects-parameter definitions
  - Can write/modify state files using a text editor
  - Produced in Pre, Post, TurboGrid
  - State files cannot perform actions

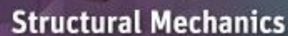
© 2015 ANSYS, Inc.

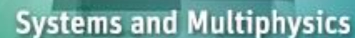
ANSYS Confidential

Introduction to Perl

16.0 Release







Introduction to ANSYS CFX

ANSYS®
Realize Your Product Promise®

ANSYS

What is Perl?

- **Perl is a public domain scripting language that combines the features and purposes of many command languages and tools.**
  - It is a fully featured programming language (even supports Object Oriented programming)
  - Has replaced shell scripting, awk, sed, regexp, grep, etc. inside of CFX
  - Good text handling and parsing capabilities

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Why use Perl?

- **Advantages**
  - Powerful, consistent, fully-featured programming language
  - System interoperability (Windows/Unix)
  - Strong user base & public support
    - Many useful Perl modules (subroutine/object libraries) freely available

- **Disadvantages**
  - It is an interpreted language
    - Can't 'hide' code
    - Slow for computationally intensive processes
  - **Many ways to do the same thing**
    - Easy to write 'obfuscated' Perl

 © 2015 ANSYS, Inc.

Perl References

- **Books:**
  - **Introductory**
    - Randal L. Schwartz *et al*, Learning Perl, O'Reilly and Associates, Inc.
    - Hoffman, Perl 5 For Dummies, IDG Books, ISBN 0-7645-0460-6
  - **The Perl Bible**
    - Larry Wall *et al*, Programming Perl, O'Reilly and Associates, Inc.
  - **Advanced Use**
    - S. Srivivasan, Advanced Perl Programming, O'Reilly and Associates, Inc.
- **Web:**
  - www.perl.org, www.perl.com, www.perldoc.perl.org
  - newsgroups

Perl Example

```
#!/usr/bin/perl
print “What is your name? “;
$name = <STDIN>;
chomp($name);
if ($name eq “Steve”) {
    print “Hi Steve! Good to see you again!\n”;
     #friendly greeting
} else {
    print “Hello, $name.  Nice to meet you.\n”;
     #ordinary greeting
}
```

ANSYS

Syntax Basics

- **Perl statements are terminated by a semicolon (;)**
- **Whitespace and indentation do not matter**
  - Except for making the code readable...
- ***Everything* is case sensitive**
- **Comments are preceded by a pound sign (#)**
  - There are no multi-line comments (e.g. /* [..] */ in C++)

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Perl Variables

- **Variable type is implied, not declared**
- **Leading character determines return type**
  - **Scalars: $…**
    - Denotes a 'single' value
    - Can be a number, character string, pointer, array element, etc.
  - **Linear Arrays: @…**
    - Elements reference by position
    - Elements may be any type (scalar, array, hash)
  - **Hash (associative array): %**
    - Elements referenced by lookup (associative)
    - Elements may be any type
    - Very useful for nested data

ANSYS

Scalar Variables $

- **Scalars are single valued numbers or strings**
- **Scalar variable names are of the form $*varName***
  - The first character in *varName* must be a letter
- **All numbers treated internally as double-precision floats. Format is flexible:**
  - 1.25, -5.34e34, 12, -2001, 3.14E-5
- **Variable assignment uses the equal sign (=)**
  - $pi = 22/7.0 #close enough

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Strings “…”, ‘…’

- **Strings are a quoted group of characters**
  - double-quoted (“) and single-quoted (‘) strings are handled slightly differently.
- **Double quoted strings act a lot like strings in C**
  - Can include ‘backslash escapes’ to represent special characters.

Escape Character	*Meaning*
\n	Newline
\t	Tab
\\	literal \
\"	literal "
\x*nn*	hex ascii value *nn*

- **$greeting = “hello world\n”; # hello world, newline**

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Arrays (lists)

- **An array is an ordered list of scalar data**
- **Arrays can have any number of elements**
  - Perl deals with all memory management issues
  - Arrays are zero indexed
- **The @ sign denotes an array variable**
  - @evens=(2,4,6,8);
  - @numbers = (1..5); # (1,2,3,4,5)
- **Access a array elements using $*listName*[*indices*];**
  - $four = $evens[1];
  - ($four,$five) = $numbers[3,4];

 © 2015 ANSYS, Inc.

ANSYS

Arrays (lists)

- **To determine the size of an array, use 'scalar(@listName)'**
- **Useful array functions:**
  - **push, pop: To add/remove elements from the high end of a list**
    - `push(@list,$newValue)` or `push(@list,@moreElements)`
    - `$oldValue=pop(@list)`
  - **shift, unshift: To add/remove elements from the low end of a list**
  - **reverse: reverses the order of the elements in a new list**
    - `@backwardList=reverse(@forwardList)`
  - **sort: sorts the elements in ASCII order into a new list**
    - `@ordered=sort(@unordered)`

© 2015 ANSYS, Inc.

ANSYS Confidential

Scalar Variable Operators

- **Perl uses standard maths operators**
  - add(+), subtract(-), multiply(*), divide(/), exponentiat4(**), mod(%)
  - increment (++) and decrement (--)
  - binary assignment supported (+=,*=, etc.)

```
$a = $a + 1;                  #
$a += 1;                      # All are equivalent
$a++                          #
```

- **String operators**
  - concatenation (. or .=)

```
"hello" . "world"             #gives "helloworld"
"fred" . " " . "wilma"        #gives "fred wilma"
```

  - string repetition (x)

```
"la" x 3                      #gives "lalala"
"foo" x (4 + 1)               #gives "foofoofoofoofoo"
```

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Perl Functions

- **Perl has many, many built in functions**
  - Functions for:
    - Text processing, numeric functions, process control, list processing, file IO, data processing, etc.
  - No distinction between built-in functions & user-defined subroutines
- **They all:**
  - Have a name
  - Return a value or list of values
  - Can accept arguments

```
$res = log(123);
$angle = atan2(.5,-.5);
push(@myList,2,4,7);
$value = pop(@myList);
$textBit = substr($myString,3,2);
```

 © 2015 ANSYS, Inc.

ANSYS

Logic in Perl

- **Perl contains a number of control structures (if, else, etc.) based on logic expressions**
- **The following is ‘true’:**
  - Any string except for “” and “0”.
  - Any number except 0.
  - Any reference is true
- **Anything else is false.**
- **Tests in control structures can contain any expression or operators and the result is evaluated using the above rules**

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Scalar Comparison Operators

- **Comparison operators come in numeric and string varieties**

Comparison	*Numeric*	*String*
Equal	==	eq
Not equal	!=	ne
Less than	<	lt
Greater than	>	gt
Less than or equal to	<=	le
Greater than or equal to	>=	ge

- **Make sure you use the right one…**
  - e.g. '(30 le 7)' is true. It's evaluated using ascii precedence.

 © 2015 ANSYS, Inc. ANSYS Confidential

if...

- **if:**

```
if (some expression) {
   statement 1;
   statement 2; ...
} elsif (another_expression) {
   statement 3; ...
} else {
   statement 4;
   statement 5; ...
}

print "How old are you?";
$a = <STDIN>;
chomp($a);
if ($a < 18) {
   print "Sorry, you're too young.\n";
   die;
} else {
   print "Welcome…\n";
}
```


© 2015 ANSYS, Inc.

for... & while...

- **for:**

```
for (initial_exp; test_exp; increment_exp) {
   statement 1;
   statement 2; ...
}

for ($I = 1; $I < 10; $I++) {
   print "$I\n";
}
```

- **while:**

```
while (some expression) {
   statement 1;
   statement 2; ...
}

print "How old are you?";
$a = <STDIN>;
chomp($a);
while ($a > 0) {
   print "At one time, you were $a years old.\n";
   $a--;
}
```


© 2015 ANSYS, Inc.

ANSYS Confidential

foreach...

- foreach is a useful for walking through the elements of a list

```
foreach $var (@some_list) {
  statement 1;
  statement 2; ...
}

@a = (3,5,7,9);
foreach $number (@a) {
  $number += 2;
}  # @a is now (5,7,9,11);
```

- Any changes to the scalar variable affect the list

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Other Logical Operators

- ‘!’ will negate any logical expression (i.e. ‘*not*’)
- && is a logical ‘and’
- || is a logical ‘or’

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Basic I/O

- **The connection to an input or output location is called a *handle*.**
- **Filehandles are created using open()**
  - **open for read:** `open(MYFILE,"infile.dat");`
  - **open for write:** `open(MYFILE,">outfile.dat");`
    - Will fail if file exists
  - **open for append:** `open(MYFILE,">>logfile.dat");`
- **open() returns true/false for success/failure**

```
open(IN,"infile.dat") || print "Error: couldn't open file\n";
```

 © 2015 ANSYS, Inc. ANSYS Confidential

Basic I/O

- **Write to a filehandle by printing to it**

```
print LOGFILE "Done current task.\n";
```

- **Read from a filehandle by putting it inside <...>**
  - Assigned to scalar – returns next line in file & empty string at end

```
open(INFILE,"myfile.dat");
while($line = <INFILE>) {
  print "Just read: $line\n";
}
```

  - Assigned to array - returns all lines

```
open(INFILE,"myfile.dat");
@allLines = <INFILE>;
foreach $line (@allLines) {
  print "The file contained: $line\n";
}
```

Basic I/O

- `close(FILEHANDLE)` **when done with a file**
- **`STDIN`, `STDOUT`, `STDERR` are automatically managed filehandles.**

```
print “How old are you?”;
$a = <STDIN>;
chomp($a);
if ($a < 18) {
  print “Welcome, my child.\n”;
} else {
  print “Wow, $a is pretty old.\n”;
}
```

- **The `chomp($a)` function removes the last character from a string if it is a newline (\n)**

© 2015 ANSYS, Inc.

ANSYS Confidential

External Processes

- **Handles can also be external apps**
- **Perl can launch and interact with external processes**
  - System() spawns an external process and waits for it to finish

```
$logfile = “logfile.dat”;
system(“grep -i error $logfile > errors.dat”);
```

  - Can open a command as a handle, and interact via STDIN. Command is executed when handle is closed.

```
open(SOLVER,|cfx5solve -def $myRun);
close(SOLVER); #waits until done to continue
```

 © 2015 ANSYS, Inc.

Regular Expressions

- Perl can use *regular expressions* for general text pattern matching and replacement.
- Complex and ugly, but powerful

```
print “Any last requests? ”;
chomp($a = <STDIN>);
if ($a =~ /^y/i) { # does the input begin with y
  print “What is it?”;
   <STDIN>;
   print “Sorry, I can’t do that.\n”;
}
print “Ready, aim, fire !\n”;

$string = “foot fool buffoon”;
$string =~ s/foo/bar/g; # string is now “bart barl bufbarn”

$line = “X,Y,Z,1,,1.234,34”;
@fields = split(/\,/, $line); # @fields is (“X”,”Y”,”Z”,”1”,””,”1.234”,”34”)
```


© 2015 ANSYS, Inc.

ANSYS Confidential

Subroutines

- **Subroutines are just user defined functions. Declared with ‘sub’.**

```
sub subName {
  statement 1;
  statement 2; [...]
}
```

- **Subroutines return one or more values from a *return* statement**
  - … or the value of the last statement if no explicit return.

```
$result = doSomething($a,$b);
```

- **Invoked by calling subName(args).**

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Subroutines

- **Arguments are passed to a subroutine using the @_ list.**

```
sub addThreeNumbers {
   ($a, $b, $c) = @_;
   $result = $a + $b + $c;
   return $result;
}
```

- **Must pass lists and arrays by *reference*.**

© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

References

- **Use '\' in front of a variable to create a scalar reference (pointer).**
  - `$listRef = \@myList;`
- **De-reference by using a variable reference type ($, @ or %)**
  - `push(@$listRef,"new Value");`
- **Directly access elements in a reference using -> notation**
  - For arrays: `$listRef->[$index]`
  - For hashes: `$hashRef->{$key}`

 © 2015 ANSYS, Inc. ANSYS Confidential

References

```
sub printHash {
   ($hashRef) = @_;
   foreach $key (keys(%$hashRef)) {
      print “Key: $key, Value: “ . $hashRef->{$key} .
  “\n”;
   }
}

%myHash = (“a” => “b”, “c” => “d”);
printHash(\%myHash);
```


© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Variable Scope

- **By default, all variables have global scope. Can create private variables using 'my' specification.**

```
sub addThreeNumbers {
  my ($a, $b, $c) = @_;
  my $result = $a + $b + $c;
  return $result;
}
```

- **Put 'use strict;' in a script to force explicit scoping of all variables**
  - All variables must be declared using 'my' or 'local'
  - Catches mis-typed variable names

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Libraries and Modules

- **Significant benefit of Perl is the ability to re-use other people's work.**
- **You can include a set of subroutines from another file with `require 'filename.pl'`**
- **A wide range of modules are publicly available**
  - www.cpan.org
  - e.g. matrix algebra, HTML parsing, database manipulation, graphing, GUIs, 3rd party interfaces

 © 2015 ANSYS, Inc.

External Perl Scripting

- **External Perl scripts can be used to drive the CFX-Solver for multiple runs, optimisation loops, etc.**
- **The CFX-Solver can be sent CCL through the command line to over-ride local settings.**
  - `cfx5solve -ccl <filename | ->`
  - '-' means read from stdin
  - `cfx5solve -def duct.def -ccl special.ccl`

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

External Perl Scripting

- **CFX-Pre, CFD-Post and TurboGrid can also be launched from within a Perl script and automatically run a session file to perform quantitative or graphical post-processing in batch mode.**

```
system(“cfx5post -batch mysession.cse results.res”);
```

Or

```
open(CFDPOST, |cfx5post -line);
  print CFDPOST …CCL COMMANDS…
  …
close(CFDPOST);
```

 © 2015 ANSYS, Inc. ANSYS Confidential

CCL & Perl

- **CCL includes `Power Syntax` as a programming language**
  – Indicated by a “!” at the start of the line
- **Power Syntax *is* the Perl programming language**
  – Full support for loops, if/then/else, subroutines and much more

```
! $numSteps = 20;
! for (my $i=0; $i<=$numSteps; $i++) {
!  $transparency = $i/$numSteps;
BOUNDARY:Default
 Transparency = $transparency
END
!}
```

```
! $speed = 2.0;
! if ($speed gt 1.5) {
!   $turbulence = on;
! }
...
BOUNDARY: inlet1
  Normal speed in = $speed [m/s]
  ! if ($turbulence == on) {
   Eddy length scale = 0.001 [m]
  ! }
END
```

 © 2015 ANSYS, Inc. ANSYS Confidential

Power Syntax in the Solver

- **The Solver accepts Power Syntax (Perl)**
  - Embed Perl in the CCL to pass it to the Solver
  - Parsed as one chunk
  - Loop over the objects/parameters modifying the data, the last value wins
  - The final state is sent to the solver for execution
- **Primary use is to define and use variables from external sources to modify existing objects**

© 2015 ANSYS, Inc.

ANSYS Confidential

Power Syntax in CFD-Post

- **CFX-Pre and CFD-Post operate more interactively than the solver**
  - CCL is treated as a ‘session’ not a ‘state’
  - Can have actions
  - Have a number of custom of Perl functions defined
- **Can create macros (Perl subroutines) that contain power syntax and standard CCL**
  - Read in subroutine definition from a session file
  - GUI definition in header

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

CFD-Post Perl Functions

- **See CFD-Post Advanced documentation**
- **Quantitative functions**
  - All CEL extended functions have power syntax equivalents (e.g.)
    - $val = massFlow(location);
    - $val = areaAve(variable,location);
  - Just return the value
- **Evaluate the value and units of any single-valued CEL expression**
  - ($val,$units) = evaluate("myExpression");
  - Preferred to use this instead of the quantitative functions
    - More general and integrated with CEL

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

CFD-Post Perl Functions

- **Determine the value of any CCL parameter in the post-processor**
  - $val = getValue(“OBJECT:NAME”,“Param Name”);
- **List currently defined subroutines to command window**
  - showSubs
- **List currently defined power syntax variables and values**
  - showVars

 © 2015 ANSYS, Inc. ANSYS Confidential

ANSYS

Post Macros

- Power Syntax macros can be loaded as “macros” in Pre and Post
- A macro is basically a session with a user interface
- User interface elements are defined in the macro header

© 2015 ANSYS, Inc.

ANSYS Confidential

Macro Header

- Header defines macro name, subroutine and parameters
- Choice of parameter type

```
#Macro GUI begin
#
# macro name = A simple macro
# macro subroutine = mySub
#
# macro parameter = Var
#  type = variable
#  default = Y
#
# macro parameter = Location
#  type = location
#  location type = plane
#
# Macro GUI end

! sub mySub {
! ( $variable, $plane) = @_;
!
! print "variable = $variable, plane =
     $plane\n";
!}
```


© 2015 ANSYS, Inc.

ANSYS Confidential

ANSYS

Macro Header

Value	Definition
# macro name = <name>	The macro identifier to appear in the macro combo
# macro subroutine = <subname>	The subroutine to call
# macro report file = <filename>	The file generated by the macro (if any). This enables the "View Report" button, which attempts to load the file in a text/html browser.
# macro related files = <file1>, <file2>	Other related files to load when loading this macro. This is useful when your macro uses subroutines from other files
# macro parameter = <name> #type = <type> #<option1> = <val> #<option2> = <val> #.....	Specifies a GUI widget for a subroutine parameter. The type of widget is determined by the type of parameter. For each type there can be several possible options. The order of the GUI widgets must match the order of the arguments for the subroutine.

© 2015 ANSYS, Inc.

ANSYS Confidential

Macro Header

Type	Option	Example
`string`	`default`	`My String`
`integer`	`default`	`10`
	`range`	`1, 100`
`float`	`default`	

© 2015 ANSYS, Inc.

ANSYS Confidential

Perl Examples in CFX #1

```
LIBRARY:
CEL:
EXPRESSIONS:
  mdot1 = Water.massFlow()@vane1
  mdot2 = Water.massFlow()@vane2
  mdot3 = Water.massFlow()@vane3
  mdot4 = Water.massFlow()@vane4
END
END
END

# Evaluate to Perl variable for output
!my $MD1 = getExprString(mdot1);
!my $MD2 = getExprString(mdot2);
!my $MD3 = getExprString(mdot3);
!my $MD4 = getExprString(mdot4);

! my $fileName = getValue("DATA READER","Current Results File");
! my $date = scalar(localtime);

# Get a sensible filename for the output
! my @subStrings = split /\//, $fileName;
! my $size = @subStrings;
! my $outputFile = $subStrings[$size - 1];
! $outputFile =~ s/\.res/_output\.txt/;

# Output to text file
! open( OUTPUT, "> $outputFile" );
! print OUTPUT "Generated from $fileName at $date\n";
! print OUTPUT "Mass Flow on Vane1 = $MD1\n";
! print OUTPUT "Mass Flow on Vane2 = $MD2\n";
! print OUTPUT "Mass Flow on Vane3 = $MD3\n";
! print OUTPUT "Mass Flow on Vane4 = $MD4\n";
! close OUTPUT;
```

- **CFD-Post**
  - **This example exports four mass flow rate values to a text file**

Perl Examples in CFX #2

```
# This routine makes ten XY
# planes coloured by Pressure
# at 0.03 [m] intervals

!$numsteps = 10;
!for ($i=0; $i < $numsteps; $i++) {
  ! $numb = ($i+1)/33.3333;
  PLANE: Plane $i
    Colour Mode = Variable
    Colour Variable = Pressure
    Option = XY Plane
    Range = Global
    Z = $numb [m]
  END
!}
```

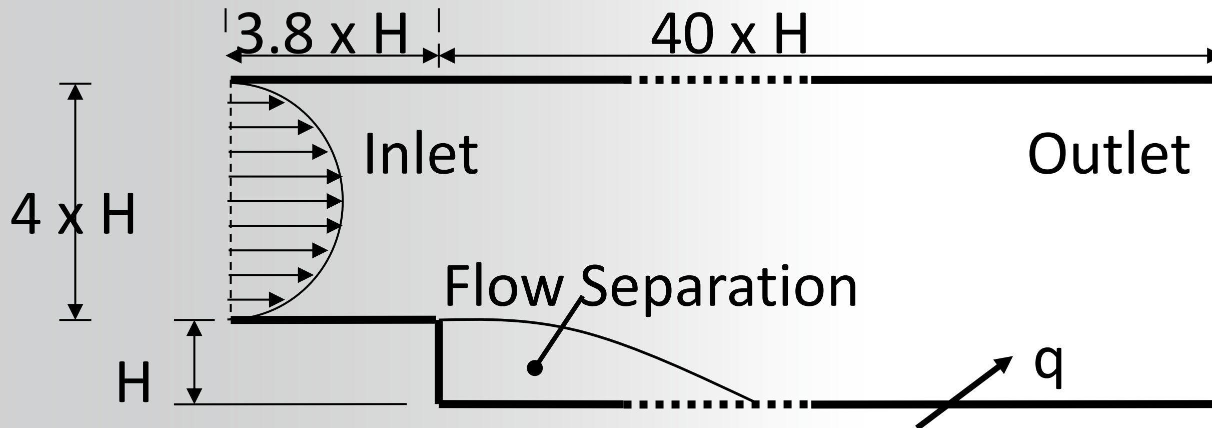
- **CFD-Post**
  - **This example creates 10 cut planes colored by pressure through a domain**

 © 2015 ANSYS, Inc.

ANSYS

Workshop Appendix A

This workshop takes you through the use of session files and scripts to run a series of simulations of flow over a backward facing step to compare the results obtained with different turbulence models.



© 2015 ANSYS, Inc.

ANSYS Confidential