The background of the slide features a dark blue globe with the ANSYS logo in the center. The globe is surrounded by a complex, glowing field of blue and orange energy lines, resembling a magnetic field or a high-energy plasma. The lines are dense and swirling, creating a sense of dynamic energy and technological sophistication.

Introduction to ANSYS Workbench Scripting in ANSYS 12.1

November 2009

Agenda



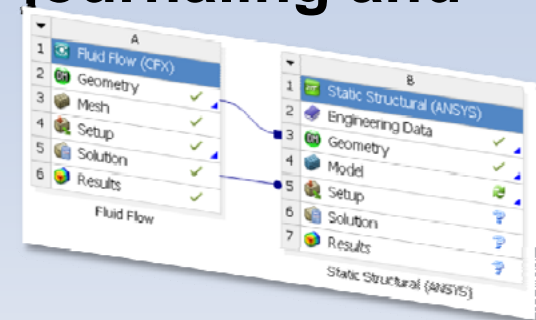
- **Overview of ANSYS Workbench Scripting**
- **Recording & Replaying Journals**
- **Command Window**
- **Scripting Basics**
- **Project & Data Model Concepts**
- **Scripting with Data-Integrated Applications**
- **Where to Get Help**
- **Examples**

Scripting Overview



- **ANSYS 12.1 fully supports Workbench journaling and scripting**

- Project concepts & operations
- Parameter management
- Native applications
 - Project Schematic, Design Exploration, Engineering Data
- File management and data models



- **Python-based scripting language**

- Object-oriented
- Platform-independent

- **Fully documented & supported**

- **Works “hand-in-hand” with application-level scripting**

- DesignModeler, Meshing, Mechanical, Mechanical APDL, FLUENT, CFX, etc.

ID	A	B	C	D
	Input Parameters	Parameter Name	Value	Unit
1	P1	InletLength	15	
2	P2	PlateThickness	0.5	
3	P3	PlateLength	10	
4	P4	InletDiameter	7	
5	P5	Velocity	1	m/s
6	P6	New Input Parameter		
7	P7	New Output Parameter		
8	P8	New Expression		
9	P9	New Expression		
10	P10	New Expression		
11	P11	New Expression		
12	P12	New Expression		
13	P13	New Expression		
14	P14	New Expression		
15	P15	New Expression		
16	P16	New Expression		
17	P17	New Expression		
18	P18	New Expression		
19	P19	New Expression		
20	P20	New Expression		
21	P21	New Expression		
22	P22	New Expression		
23	P23	New Expression		
24	P24	New Expression		
25	P25	New Expression		
26	P26	New Expression		
27	P27	New Expression		
28	P28	New Expression		
29	P29	New Expression		
30	P30	New Expression		
31	P31	New Expression		
32	P32	New Expression		
33	P33	New Expression		
34	P34	New Expression		
35	P35	New Expression		
36	P36	New Expression		
37	P37	New Expression		
38	P38	New Expression		
39	P39	New Expression		
40	P40	New Expression		
41	P41	New Expression		
42	P42	New Expression		
43	P43	New Expression		
44	P44	New Expression		
45	P45	New Expression		
46	P46	New Expression		
47	P47	New Expression		
48	P48	New Expression		
49	P49	New Expression		
50	P50	New Expression		
51	P51	New Expression		
52	P52	New Expression		
53	P53	New Expression		
54	P54	New Expression		
55	P55	New Expression		
56	P56	New Expression		
57	P57	New Expression		
58	P58	New Expression		
59	P59	New Expression		
60	P60	New Expression		
61	P61	New Expression		
62	P62	New Expression		
63	P63	New Expression		
64	P64	New Expression		
65	P65	New Expression		
66	P66	New Expression		
67	P67	New Expression		
68	P68	New Expression		
69	P69	New Expression		
70	P70	New Expression		
71	P71	New Expression		
72	P72	New Expression		
73	P73	New Expression		
74	P74	New Expression		
75	P75	New Expression		
76	P76	New Expression		
77	P77	New Expression		
78	P78	New Expression		
79	P79	New Expression		
80	P80	New Expression		
81	P81	New Expression		
82	P82	New Expression		
83	P83	New Expression		
84	P84	New Expression		
85	P85	New Expression		
86	P86	New Expression		
87	P87	New Expression		
88	P88	New Expression		
89	P89	New Expression		
90	P90	New Expression		
91	P91	New Expression		
92	P92	New Expression		
93	P93	New Expression		
94	P94	New Expression		
95	P95	New Expression		
96	P96	New Expression		
97	P97	New Expression		
98	P98	New Expression		
99	P99	New Expression		
100	P100	New Expression		



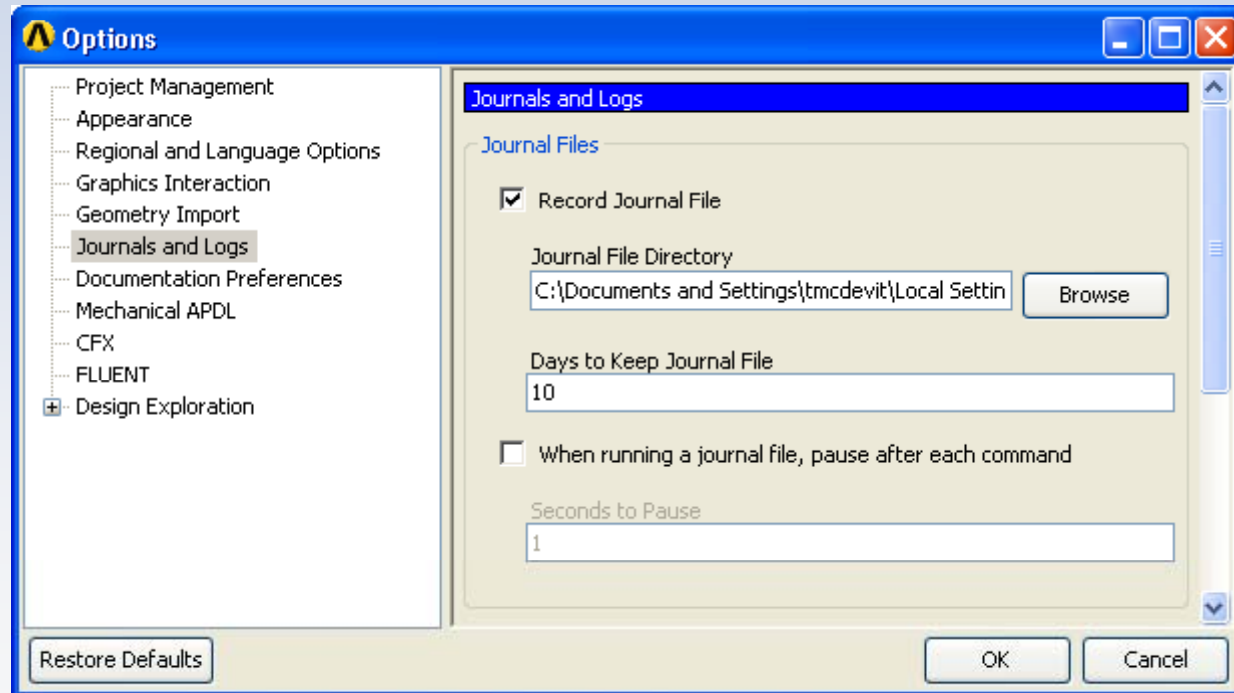
- **Workbench operations are recorded in a journal file**
 - Only operations which modify data model
 - Files have *.wbjn extension
- **Each session creates a new journal file**
- **Playing back the journal recreates the session**
- **Journals should not be confused with log files**
 - Log files record specific Workbench events
 - Log files cannot be replayed

- **Two types of Workbench journals**
 - Automatically recorded session journals
 - Restore work from a complete session
 - NOT designed to archive simulation projects
 - Manually recorded journals
 - Starting points for creating custom Workbench scripts
 - Communicate sequential Workbench steps to colleagues or ANSYS Customer Support

Journaling Options (Preferences)



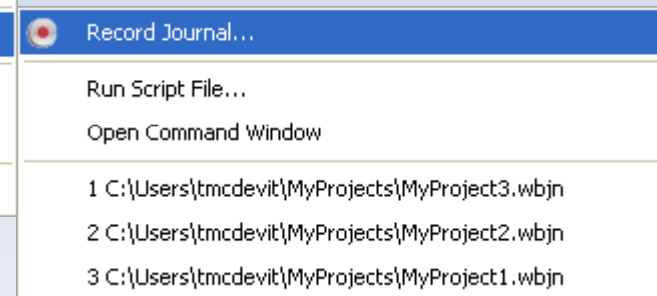
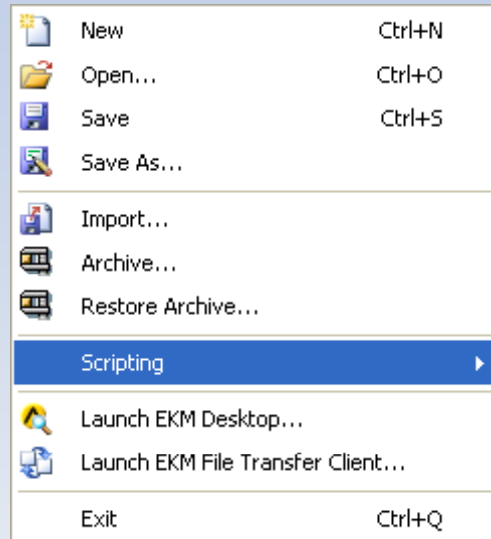
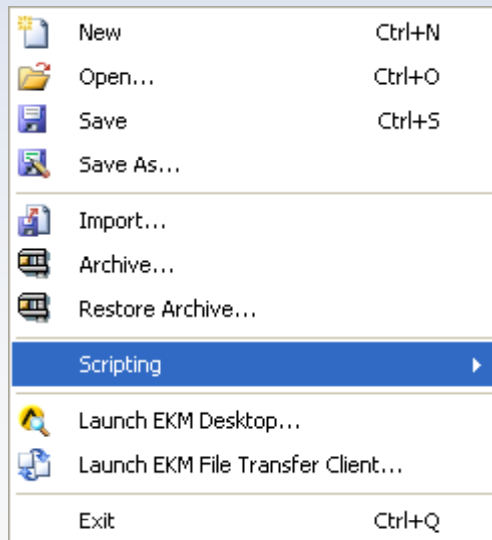
- Tools -> Options... -> Journals and Logs



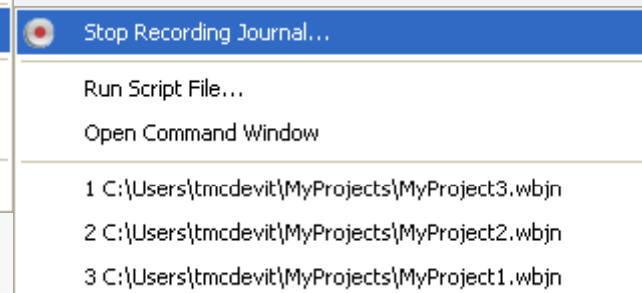
Manually Recorded Journals



Start a journal recording



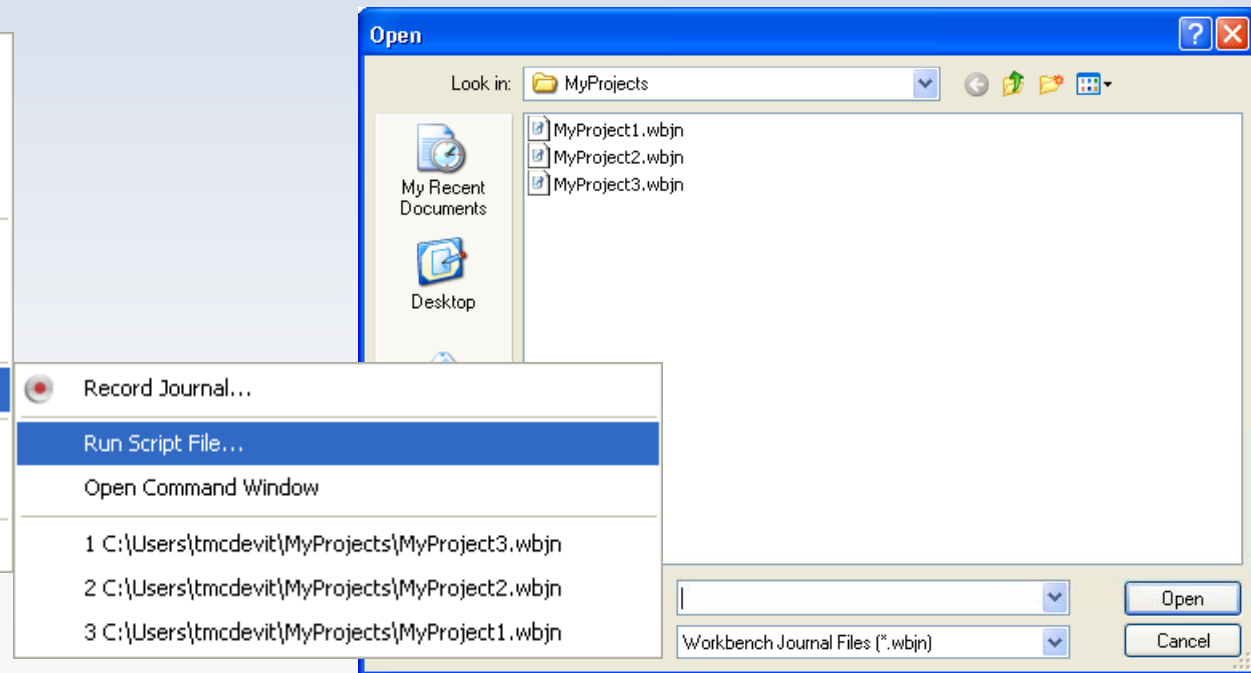
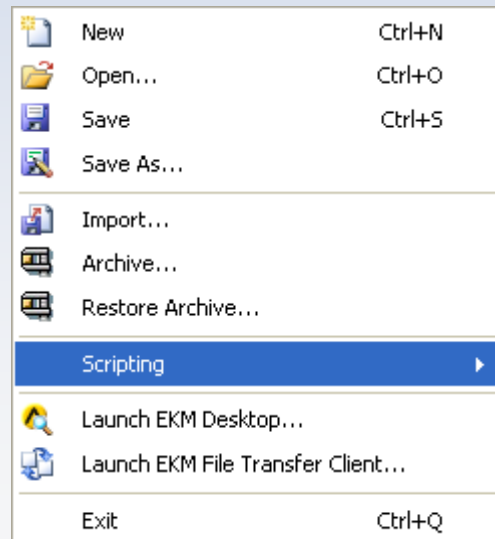
Stop a journal recording



Replaying Journals



- **Replay a journal at any time during an interactive session**
 - File → Scripting → Run Script File...



Replaying Journals



- A journal can also be replayed from the command line
 - Add `-R <filename.wbjn>`
 - Add `-I` for interactive mode or `-B` for batch

A screenshot of a Windows Command Prompt window. The title bar is blue and reads "C:\ Command Prompt". The command prompt shows the directory `C:\Users\tmcdevit\MyProjects` and the command `runwb2.exe -R MyProject.wbjn -I` entered at the prompt.

```
C:\Users\tmcdevit\MyProjects>runwb2.exe -R MyProject.wbjn -I
```

Workbench Command Line Options

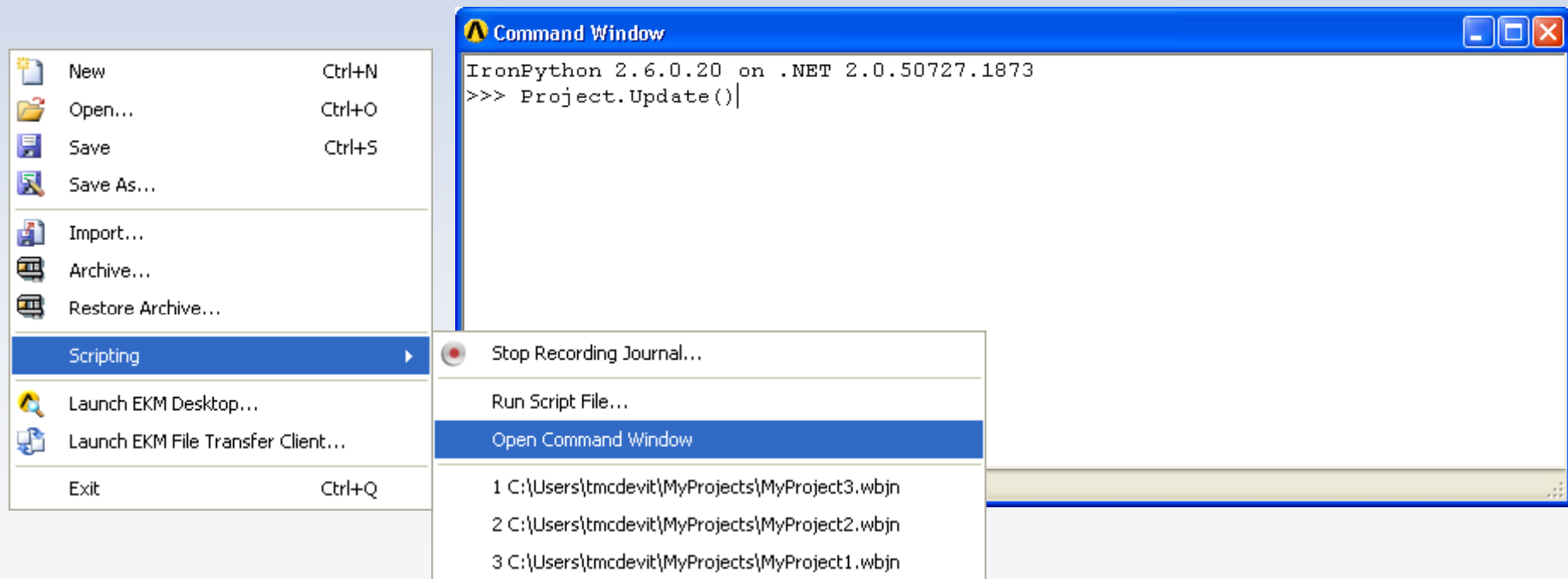


Argument	Operation
-B	Run Workbench in batch mode. The user interface is not displayed and a console window is opened. The functionality of the console window is the same as the Workbench Command Window.
-R <ANSYS Workbench script file>	Replay the specified Workbench script file on start-up. If specified in conjunction with -B , Workbench will start in batch mode, execute the specified script, and shut down at the completion of script execution.
-I	Run Workbench in interactive mode. This is typically the default, but if specified in conjunction with -B , both the user interface and console window are opened.
-X	Run Workbench interactively and then exit upon completion of script execution. Typically used in conjunction with -R .
-F <ANSYS Workbench project file>	Load the specified Workbench project file on start-up.
-E <command>	Execute the specified Workbench scripting command on start-up. You can issue multiple commands, separated with semicolons (;), or specify this argument multiple times and the commands will be executed in order.

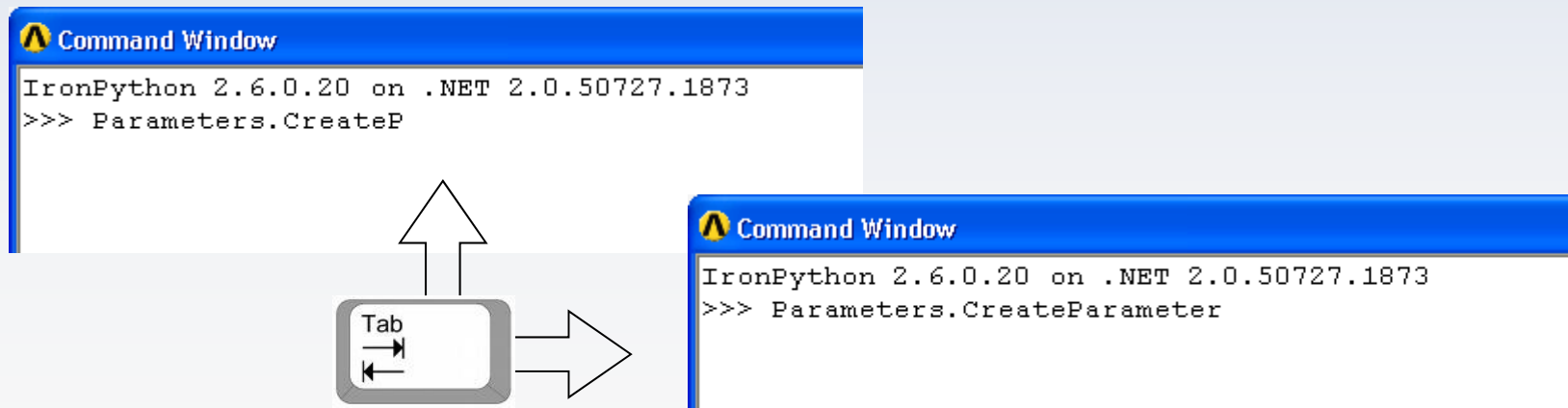
Command Window



- Scripting commands can be entered manually in Command Window



- **Command Window supports:**
 - Command completion
 - Command history
 - Keyboard shortcuts for cursor navigation and editing



- **Script:** a set of instructions to be issued to **Workbench**
 - Can be a modified journal or a completely new set of instructions written directly
 - Scripts and journals use the same programming language—Python
- **Use scripts for:**
 - Automating repetitive tasks
 - Performing Workbench operations in batch mode

Objects and Properties



- **Object:** combination of data and methods which act on the data
- **Property:**
 - Data belonging to an object
 - Defined by its name, type, and value
 - Property types: Boolean, String, Integer, Real, etc.
 - Dictionaries and Lists can also be property types
- **Properties are accessed via dot operator**
 - `parameter1.Expression = 10`
 - `parameter1` is an object of type `Parameter`
 - `Expression` is a property of that object
 - Value of `Expression` is set to 10

- **Method: command that is bound to an object**
 - Can change property values, create or delete properties, or invoke complex calculations and operations
- **Like properties, methods are accessed via dot operator**
 - `parameter1.SetQuantityUnits("m")`
 - parameter1 is an object of type Parameter
 - SetQuantityUnits is a method called to set the object's units to meters

Object-Based Scripting Approach



1. Query for an object

- Query methods return object references that are assigned to variables
 - `Parameter = DOEModel.GetParameter(Name="P1")`

2. Interrogate/modify object properties

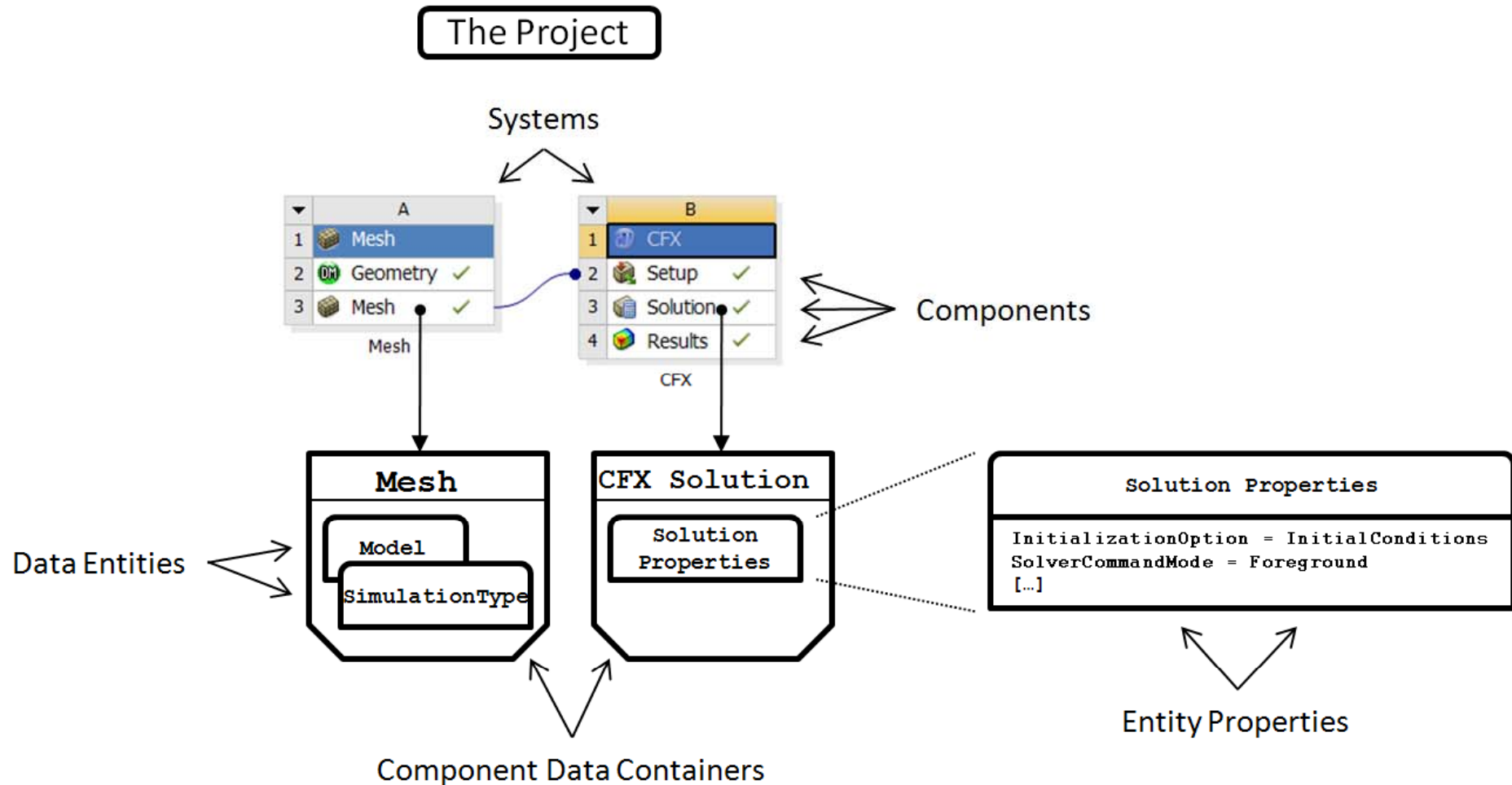
- Apply dot operator and property name to variable
 - `Parameter.Nature = "NatureUsability"`

3. Call methods to operate on object's internal data

- Methods require comma-separated argument list within parentheses
 - `Parameter.AddLevels(Levels=["65","70","75","80"])`

- **Note: Python is a loosely-typed language; you do not need to declare variables.**

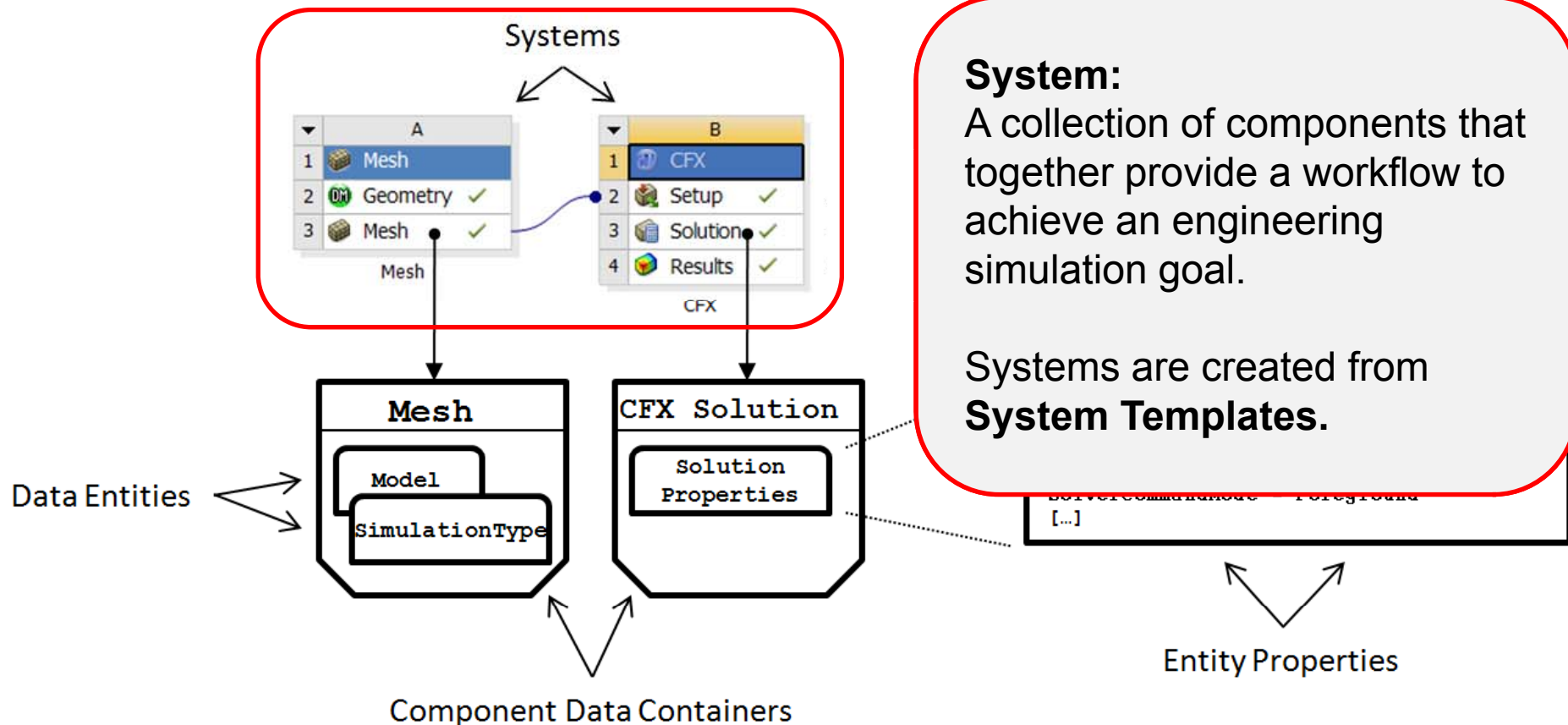
Project & Data Model Concepts



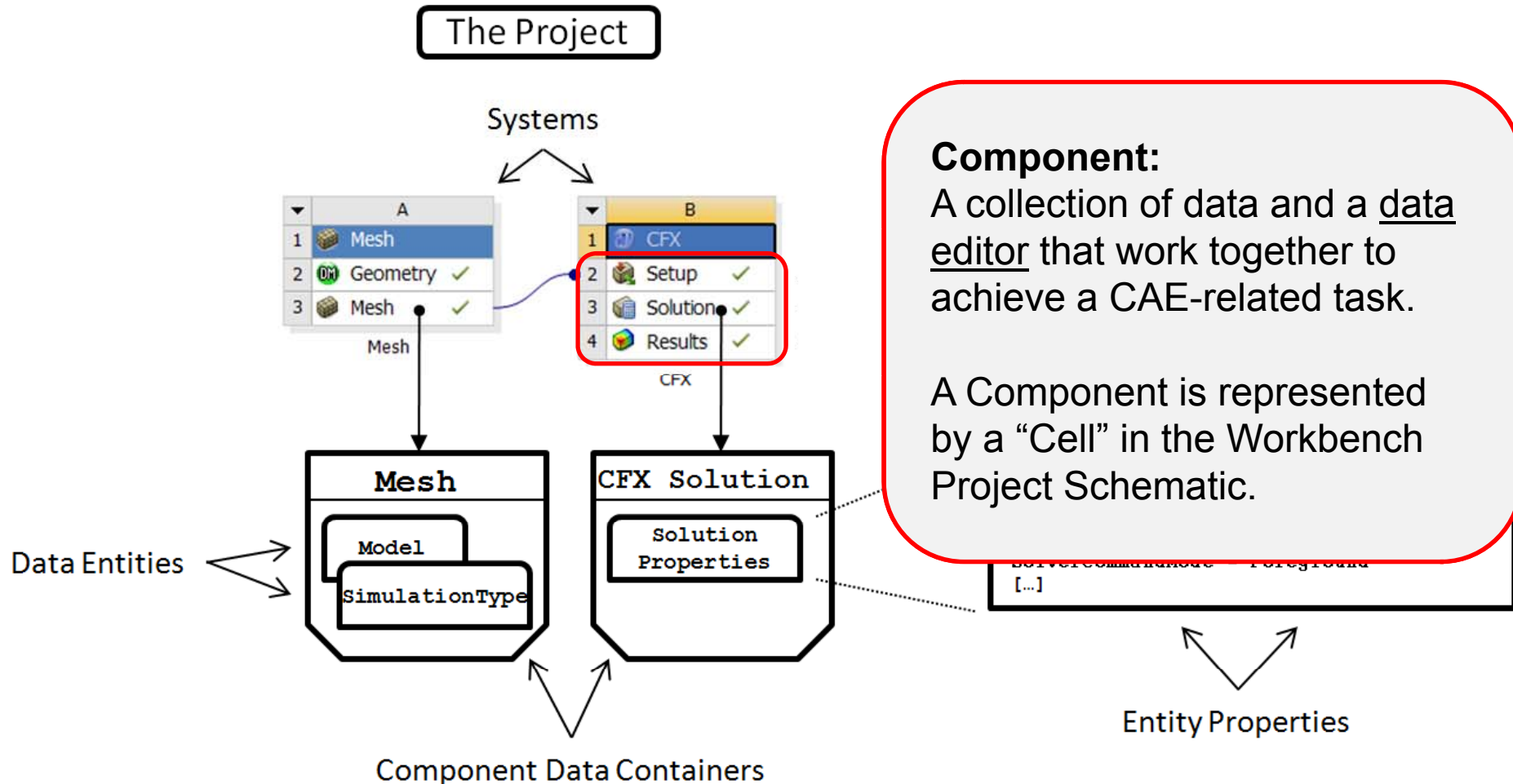
Project & Data Model Concepts



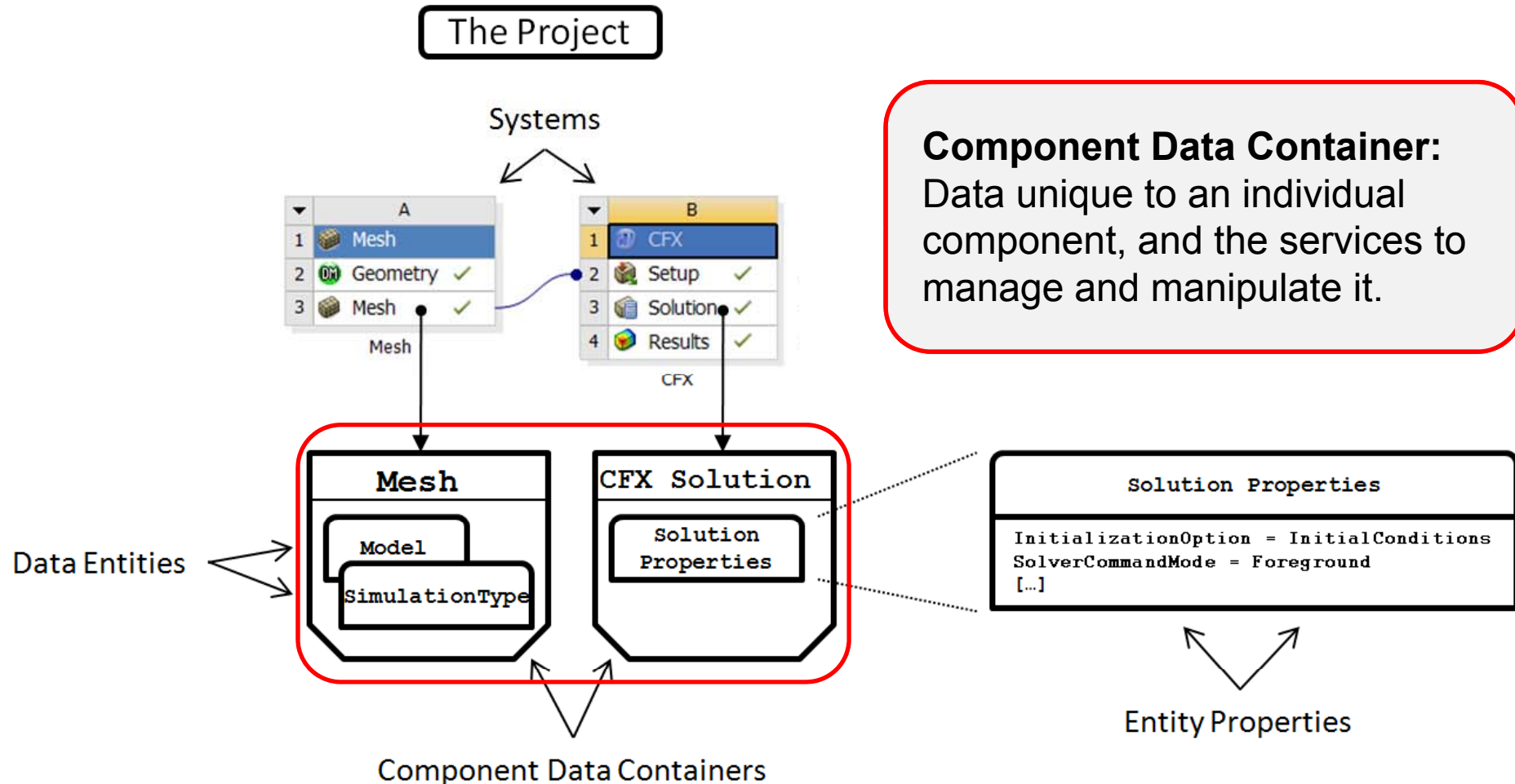
The Project



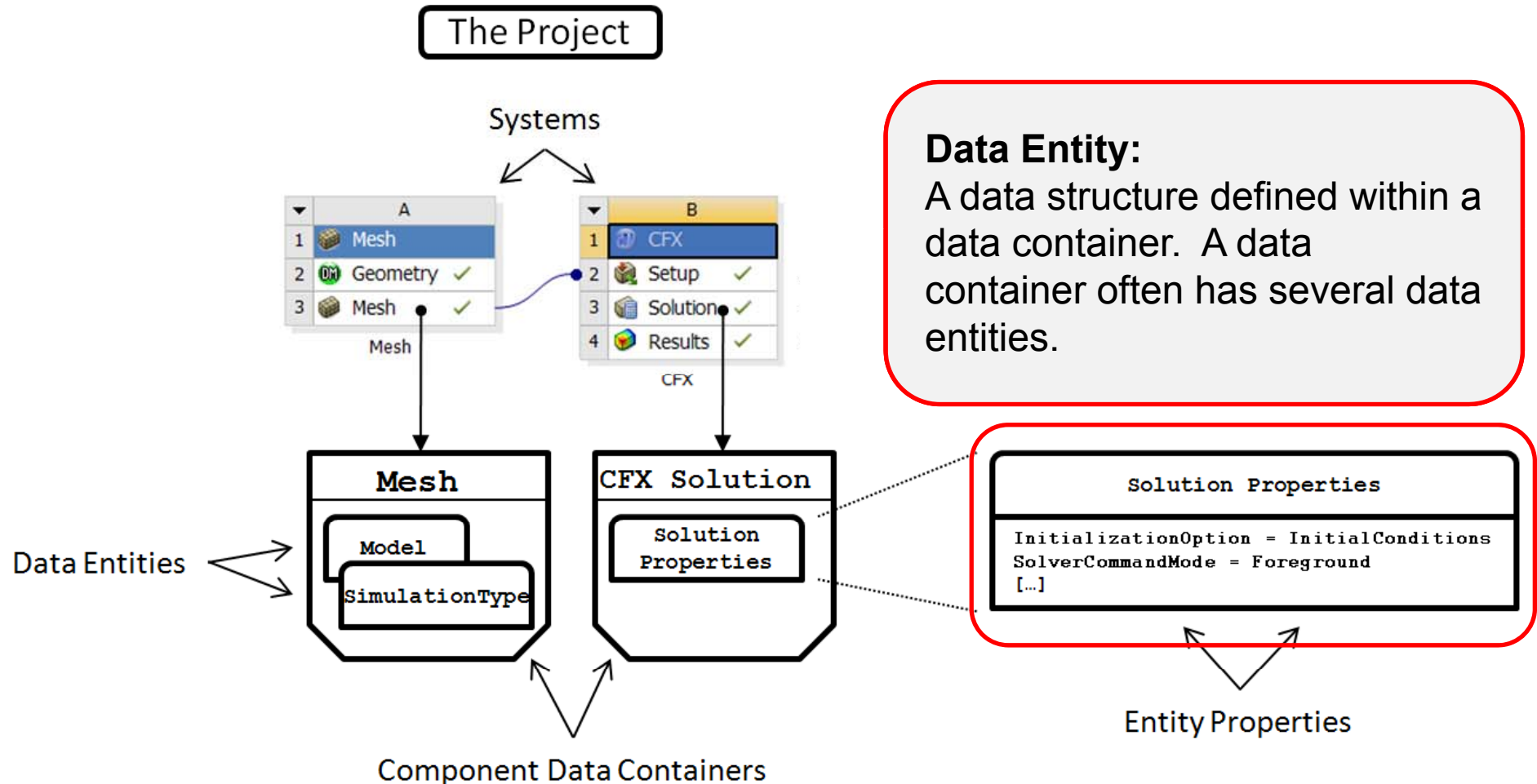
Project & Data Model Concepts



Project & Data Model Concepts



Project & Data Model Concepts



Example: Change a Material Property



```
# Query for the static structural analysis template
ss_template = GetTemplate(TemplateName="Static Structural",
    Solver="ANSYS")

# Create an analysis system from the template
ss_system    = ss_template.CreateSystem()

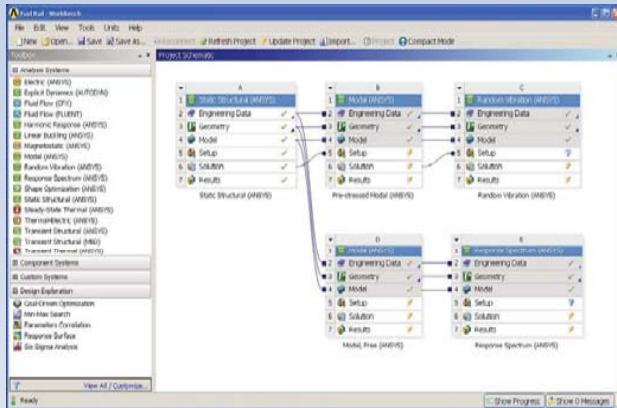
# Query for the Engineering Data container
ed_container = ss_system.GetContainer(ComponentName="Engineering Data")

# Query for the material data entity in the data container
steel = ed_container.GetMaterial(Name="Structural Steel")

# Query for the property data entity associated with structural steel
elasticity = steel.GetProperty(Name="Elasticity")

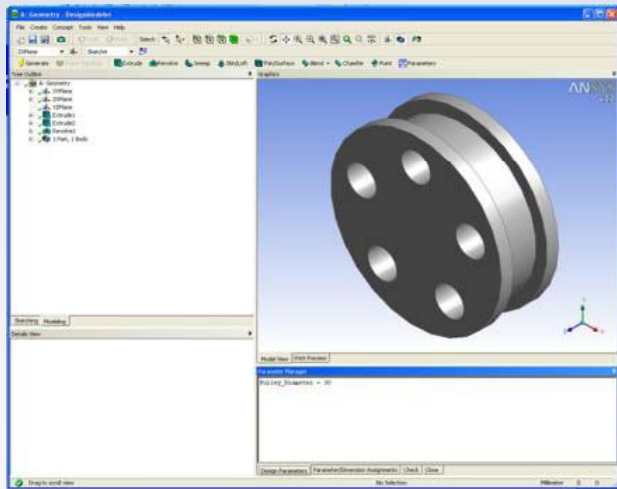
# Set Young's modulus
elasticity.SetData(Variables="Young's Modulus", Values=2E+11)
```


Data-Integrated Applications



Native applications

- Built entirely on new Workbench Framework
- Fully supported by Workbench scripting
- E.g., Project Schematic, Design Exploration, Engineering Data



Data-integrated applications

- Share data and parameters with Workbench, native applications, and other data-integrated applications
- Created independently from new Workbench Framework
- Often have their own scripting languages
- E.g., Mechanical, Mechanical APDL, CFX, FLUENT, DesignModeler

Data-Integrated Applications



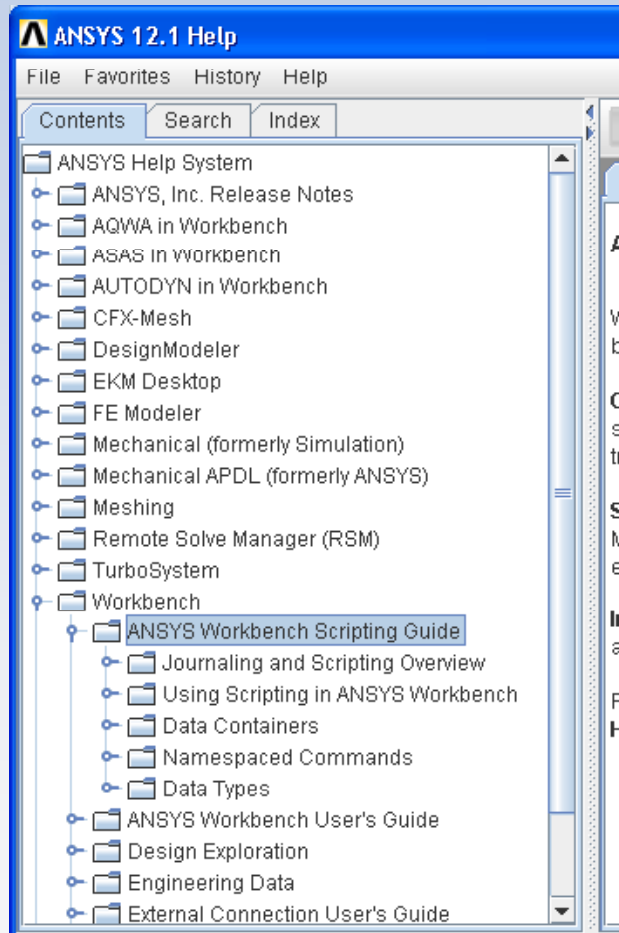
- **Application-level scripting can be embedded in a Workbench script**
 - APDL, CCL, Scheme, JScript, etc.
- **Some data-integrated applications record their operations in the Workbench journal**
 - CFX, FLUENT

Example: Create Geometry



- **The following Workbench script sketches an elliptical curve in DesignModeler:**

```
# create a geometry system
system = GetSystem(Name="Geom")
# query the data container of the geometry system
geometry = system.GetContainer(ComponentName="Geometry")
# send a JScript command to DM to create an ellipse
geometry.SendCommand(Command = """var ps1 = new Object();
    ps1.Plane = agb.GetActivePlane();
    ps1.Origin = ps1.Plane.GetOrigin();
    ps1.XAxis = ps1.Plane.GetXAxis();
    ps1.YAxis = ps1.Plane.GetYAxis();
    ps1.Sk1 = ps1.Plane.NewSketch();
    ps1.Sk1.Name = "Sketch1";
    with (ps1.Sk1) { ps1.E17 = Ellipse( 8.0, 10.0, 9.0,
    6.0, 5.0, 12.0); }
    agb.Regen();""")
```



Online Help includes:

- Overview
- How-to
- Complete command reference
- Many examples

select help based on the product in which you are working. The sections tree can be expanded to display all related books and sections.

Search: Select the **Search** tab to open the Search panel. You can search Mechanical APDL (formerly ANSYS) command, section title, or regular expression.

Index: Select the **Index** tab to see a list of the products for which Help is available from this help viewer. Double-click a product name to review its

For detailed instructions on how to use ANSYS Help, select **Using Help** from the **Help** menu.

Parameters

Parameters

This container holds project-level Parameters and Design Points.

Methods

GetParameters

Returns the set of all parameters associated with all entity properties in the given container. This method can be run on any container.

Return The set of parameters used by the given container.
Type DataReferenceSet

Example

In this example 'paramSet' becomes the set of parameters associated with the properties of any entity that resides within the Results container of system1.

```
paramSet = system1.GetContainer(ComponentName="Results")
paramSet = paramSet.GetParameters()
```

Data Entities

DesignPoint

The data entity which describes a project-level design point.

Properties

DisplayText

The general property that defines the user-visible name of an entity. This property is defined for all Data Entities, but is used only in those entities that present a label in the User Interface.

Type string

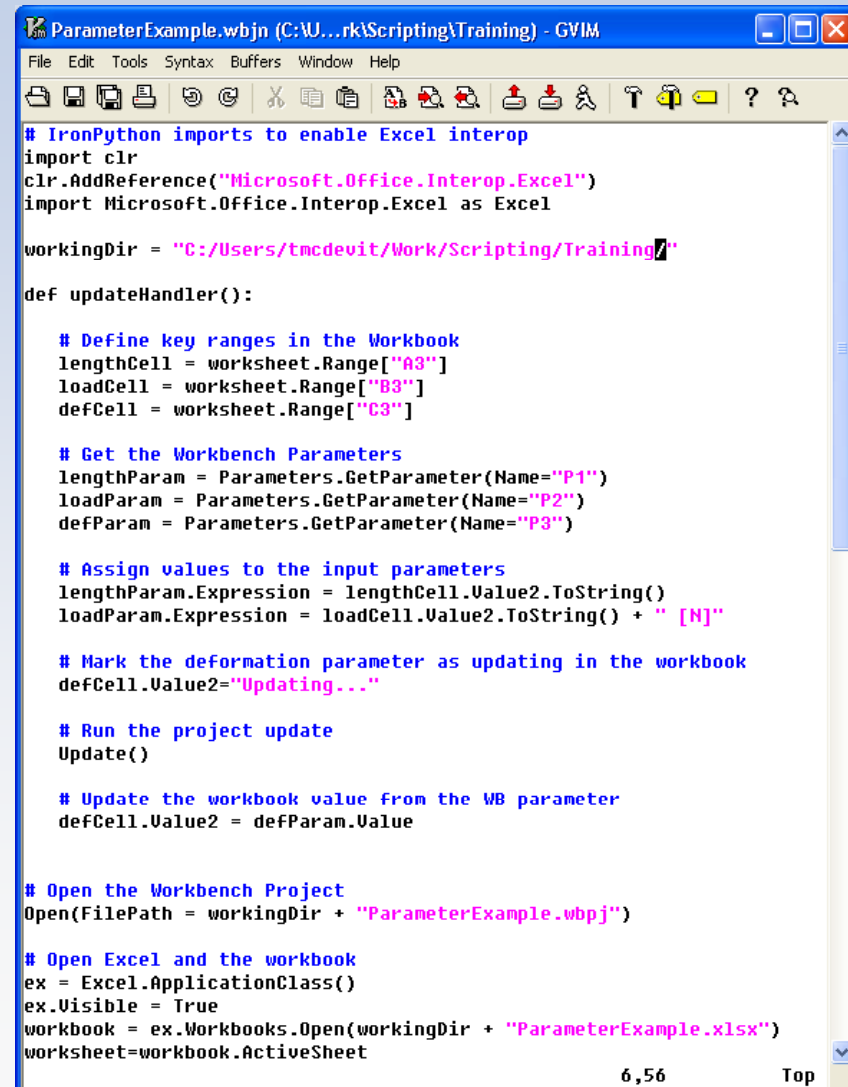
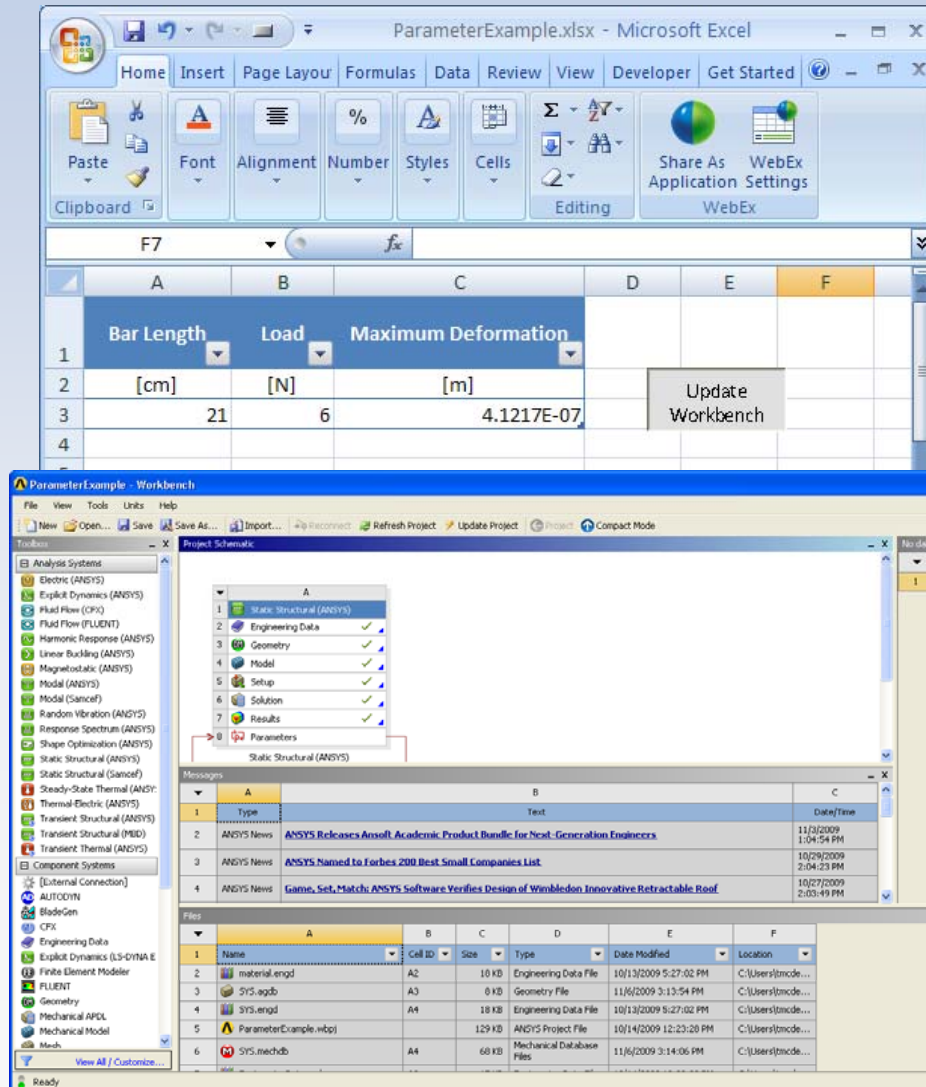
Read Only No

IsUpToDate

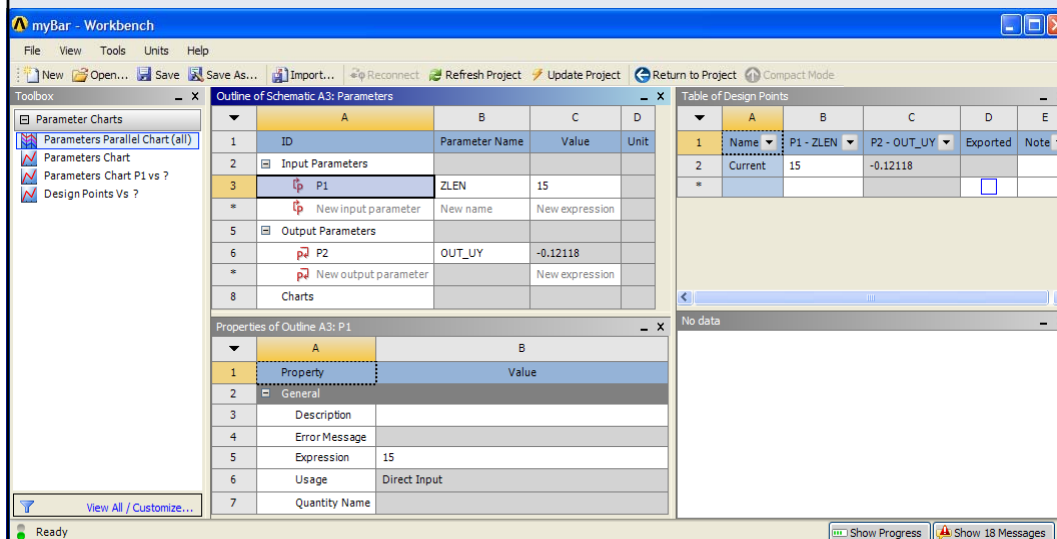
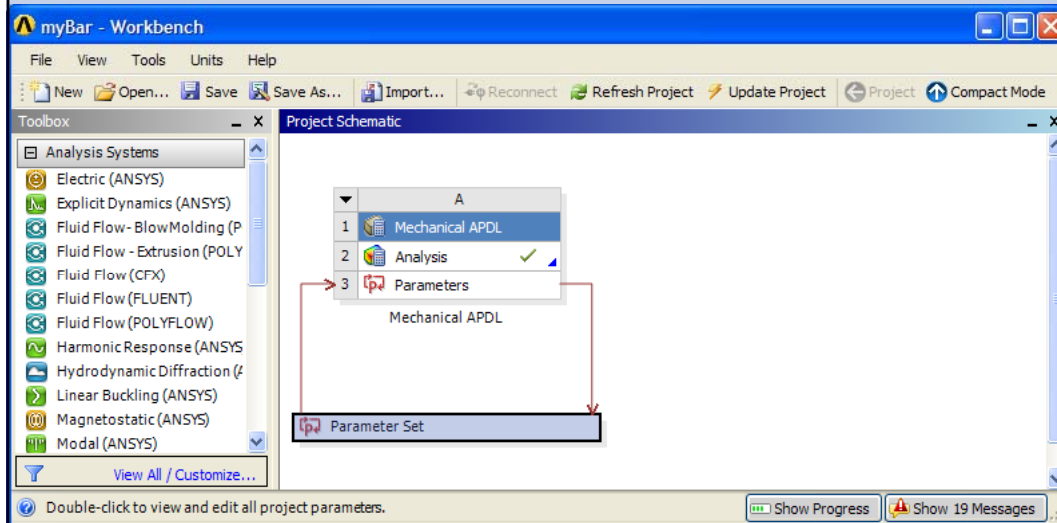
requires an update.

PDF version available on ANSYS Customer Portal

Example & Demo: Parametric Update from MS Excel



Example & Demo: Stress Analysis via APDL



```

apdl.wbproj - Notepad
File Edit Format View Help

# Import the 'os' module, which provides a portable way of using
# operating system dependent functionality
import os

# Specify the Mechanical APDL Input file to be processed
inputFile = AbsUserPathName("Demo/ScriptExample4/bar.dat")

# Provide a list of bar length (Z) values to solve and write CDB files for.
# Note: we expect '0' to fail.
zvalues = [3,5,0,12,15]

# Open a log file to record script progress
logFile = open(AbsUserPathName("my_bar_script.log"), "w")

# Start a new project and create the Mechanical APDL system
Reset()
template1 = GetTemplate(TemplateName="Mechanical APDL")
system1 = template1.createSystem()

# Read the input file into the Mechanical APDL Setup
setup1 = system1.GetContainer(ComponentName="Setup")
mapdlInputFile1 = setup1.AddInputFile(FilePath=inputFile)

# Create workbench parameters from two of the Mechanical APDL parameters
# in the input file
mapdlInputFile1.PublishMapdlParameter(Name="ZLEN")
parameter1 = Parameters.GetParameter(Name="P1")

mapdlInputFile1.PublishMapdlParameter(
    Name="OUT_UY",
    IsDirectOutput=True)
parameter2 = Parameters.GetParameter(Name="P2")

# Save the initial project definition.
Save(
    FilePath=AbsUserPathName("Demo/ScriptExample4/myBar.wbpj"),
    Overwrite=True)

# Loop through all provided bar lengths
for zval in zvalues:

    # Set the Z (length) parameter expression
    parameter1.Expression = str(zval)
    logFile.write("Updating for z = %s\n" % zval)

    # update the project for the new parameter value, and report
    # success or failure to the log file.
    try:
        update()
    except:
        logFile.write(" update failed.\n")
    else:
        logFile.write(" update succeeded. UY = %s\n" % parameter2.value)

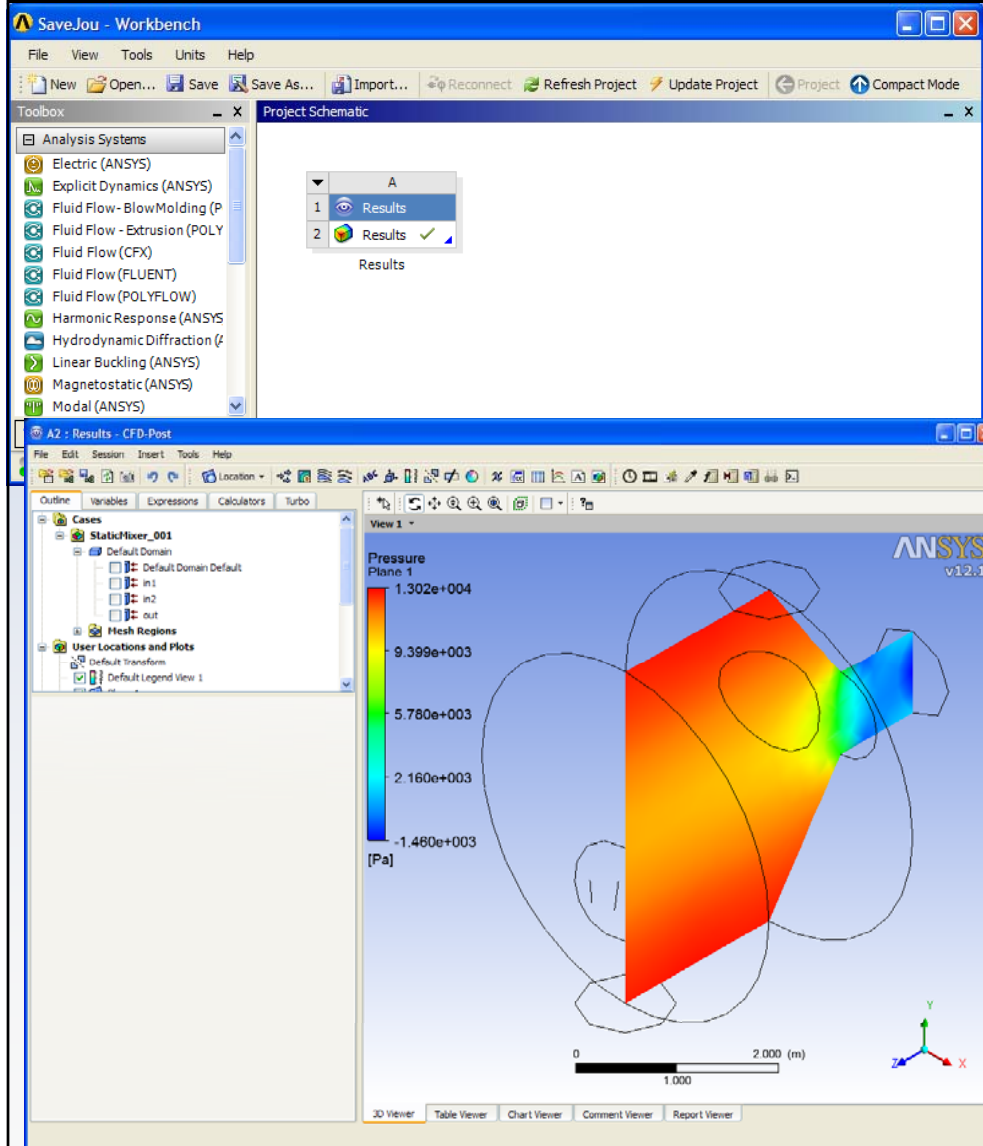
    # Generate the name of the CDB file to save
    cdbName = os.path.join(GetUserFilesDirectory(), "my_bar_" + str(zval) + ".cdb")
    cdbNameForCmd = cdbName.replace("\\", "/")

    # Delete the cdb file if it already exists, to prevent
    # Mechanical APDL from prompting us about overwrite.
    if os.path.exists(cdbName):
        os.remove(cdbName)

    # Generate the APDL command to save the CDB file and send it.
    apdlCmd = "cdw,db,%s" % cdbNameForCmd
    setup1.SendCommand(Command=apdlCmd)
    logFile.write(" CDB written to %s\n" % cdbName)

# Save the final project state.
Save()
logFile.close()
    
```


Example & Demo: Plane Creation in CFD-Post



```
ccl.wbjn - Notepad
File Edit Format View Help

# Create the Results system
template1 = GetTemplate(TemplateName="Results")
system1 = template1.CreateSystem(Position="Default")

# Edit the Results cell and load the Results file (StaticMixer_001.res)
results1 = system1.GetContainer(ComponentName="Results")
results1.Edit()
results1.SendCommand(Command=r""DATA READER:
Clear All Objects = false
Append Results = true
Edit Case Names = false
Open to Compare = false
Multi Configuration File Load Option = Separate Cases
Open in New View = true
Keep Camera Position = true
Load Particle Tracks = true
Files to Compare =
END
DATA READER:
Domains to Load=
END
> load filename=C:/users/sdg/ScriptingDemos/StaticMixer_001.res, multifile=append""")

# Set the camera and define a plane colored with a constant color
results1.SendCommand(Command=""VIEW:View 1
Camera Mode = User Specified
CAMERA:
Option = Pivot Point and Quaternion
Pivot Point = 0, 0, 0
Scale = 0.226146
Pan = 0, 0
Rotation Quaternion = 0.279848, -0.364705, -0.115917, 0.880476
Send To Viewer = False
END
END
> autolegend plot=/PLANE:Plane 1, view=VIEW:View 1""")
results1.SendCommand(Command=""PLANE:Plane 1
Apply Instancing Transform = on
Apply Texture = off
Blend Texture = on
Bound Radius = 0.5 [m]
Colour = 0.75, 0.75, 0.75
Colour Map = Default Colour Map
Colour Mode = Variable
Colour Scale = Linear
Colour Variable = Pressure
Option = Point and Normal
#
# (Lines omitted for brevity)
#
END""")
results1.SendCommand(Command=""# Sending visibility action from viewutilities
>show /PLANE:Plane 1, view=VIEW:View 1""")

# Save the project
Save(FilePath=r"SaveJou.wbpj", overwrite=True)
```