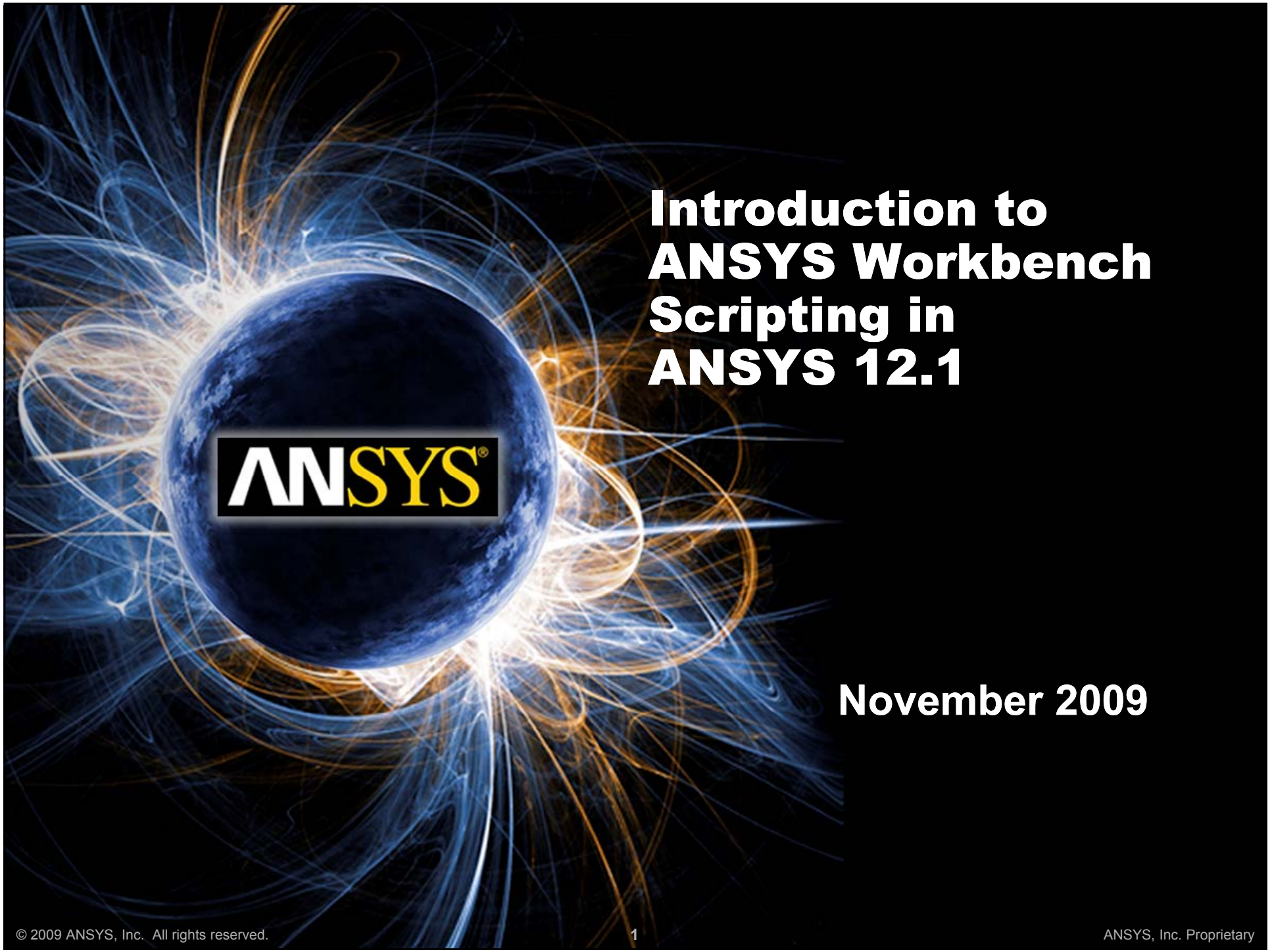


Introduction to ANSYS Workbench Scripting in ANSYS 12.1

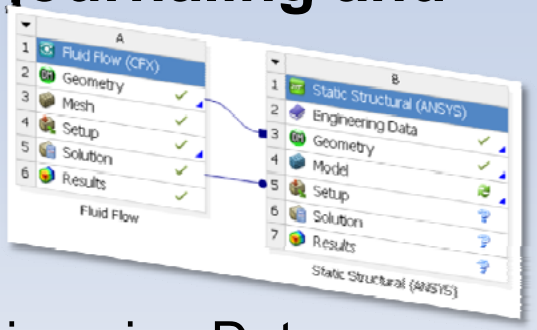
November 2009

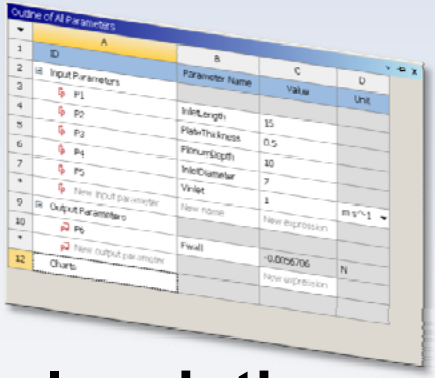
Agenda

- 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.

Journaling

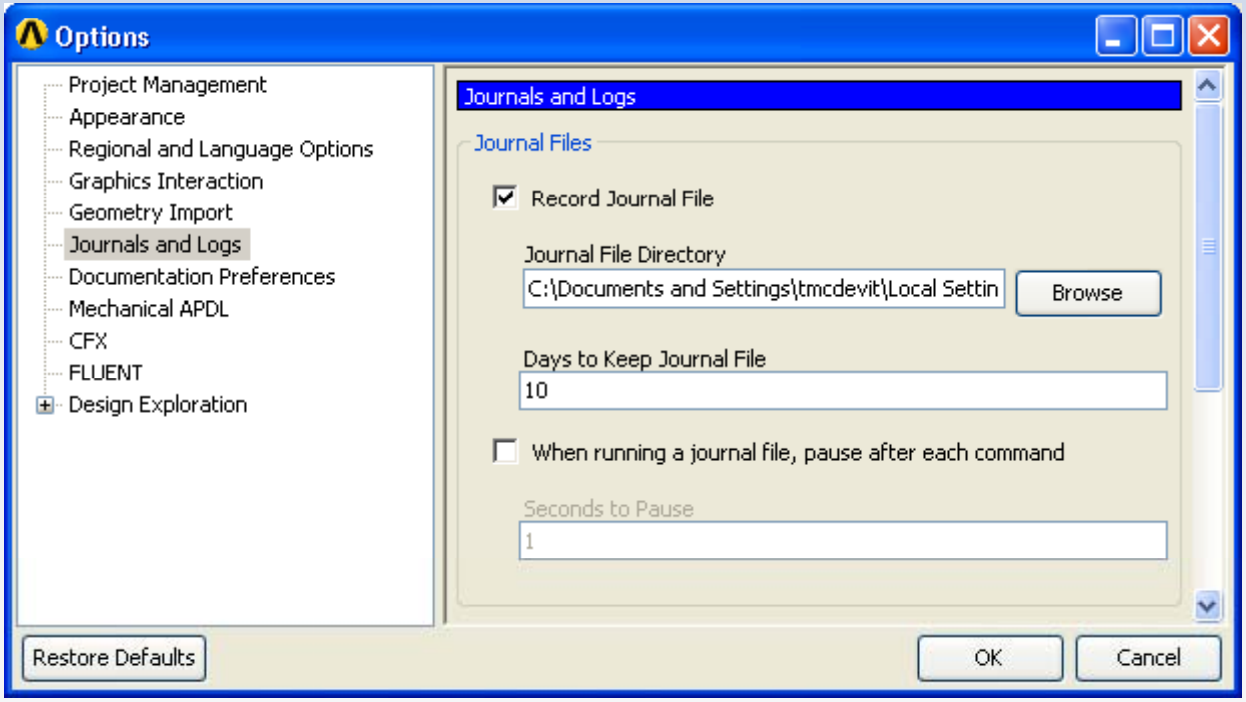
- **Workbench operations are recorded in a journal file**
  - Only operations which modify data model
  - Files have *.wbjn extension
- **Each session creates a new journal file**
- **Playing back the journal recreates the session**
- **Journals should not be confused with log files**
  - Log files record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

Journaling

- **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

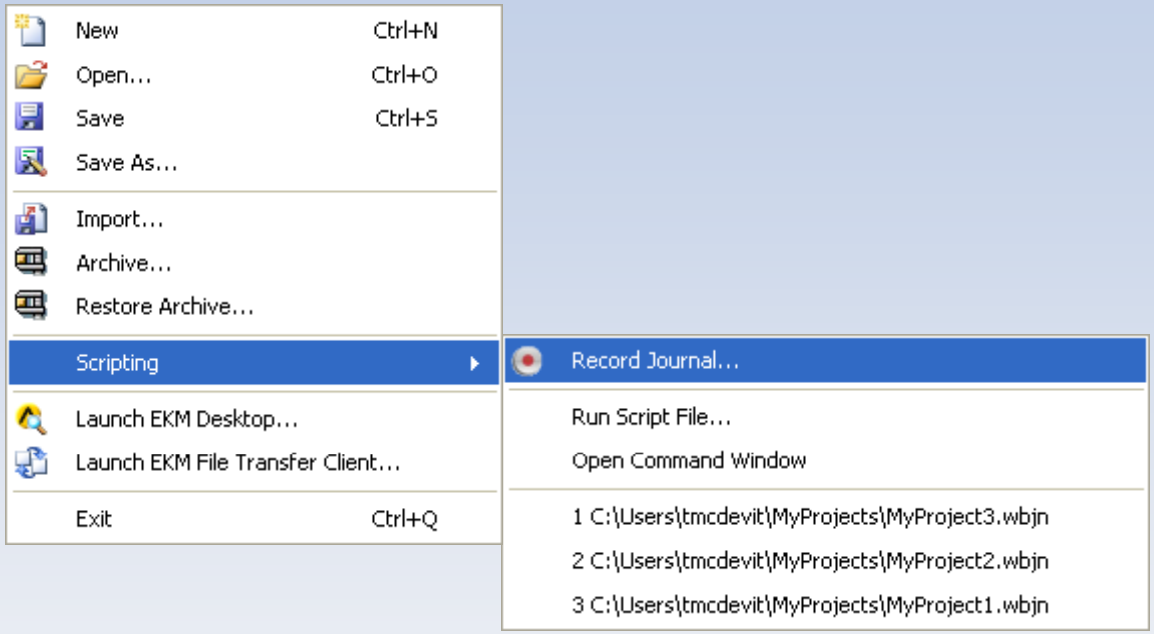
Journaling Options (Preferences)

- Tools -> Options… -> Journals and Logs

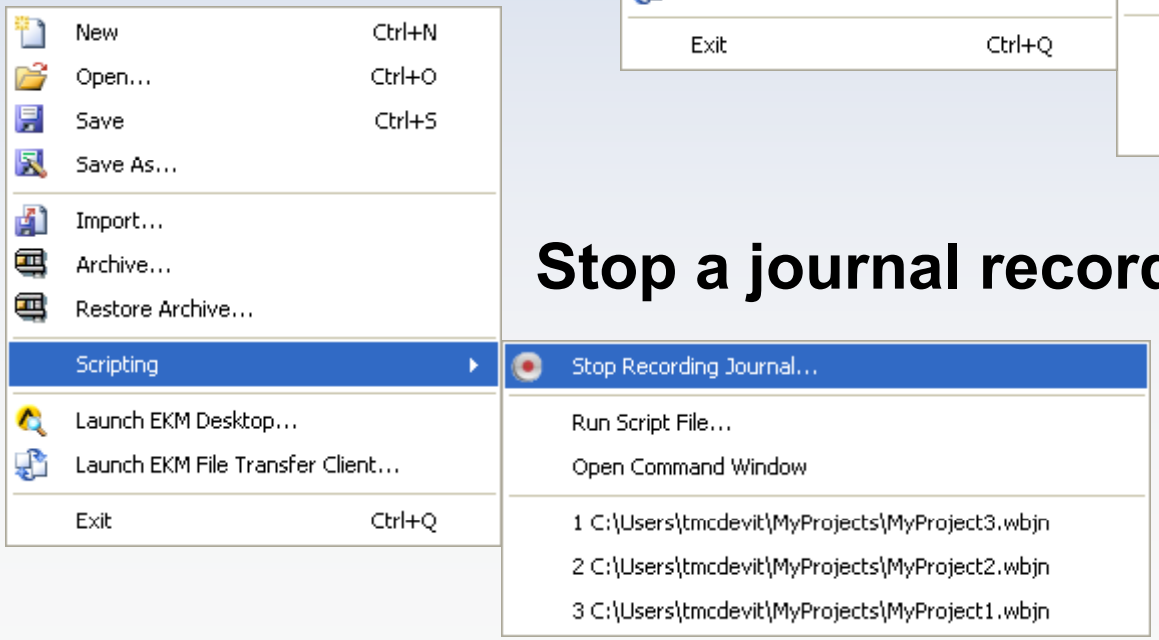


Manually Recorded Journals

Start a journal recording

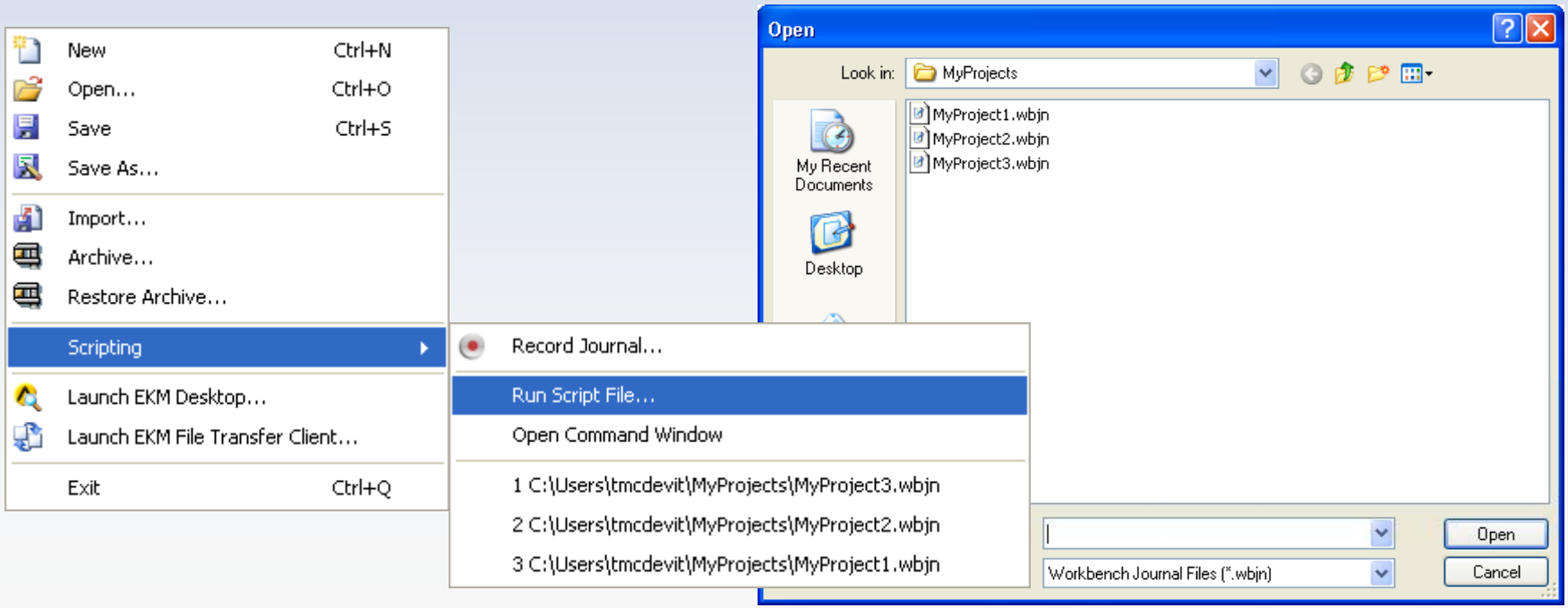


Stop a journal recording



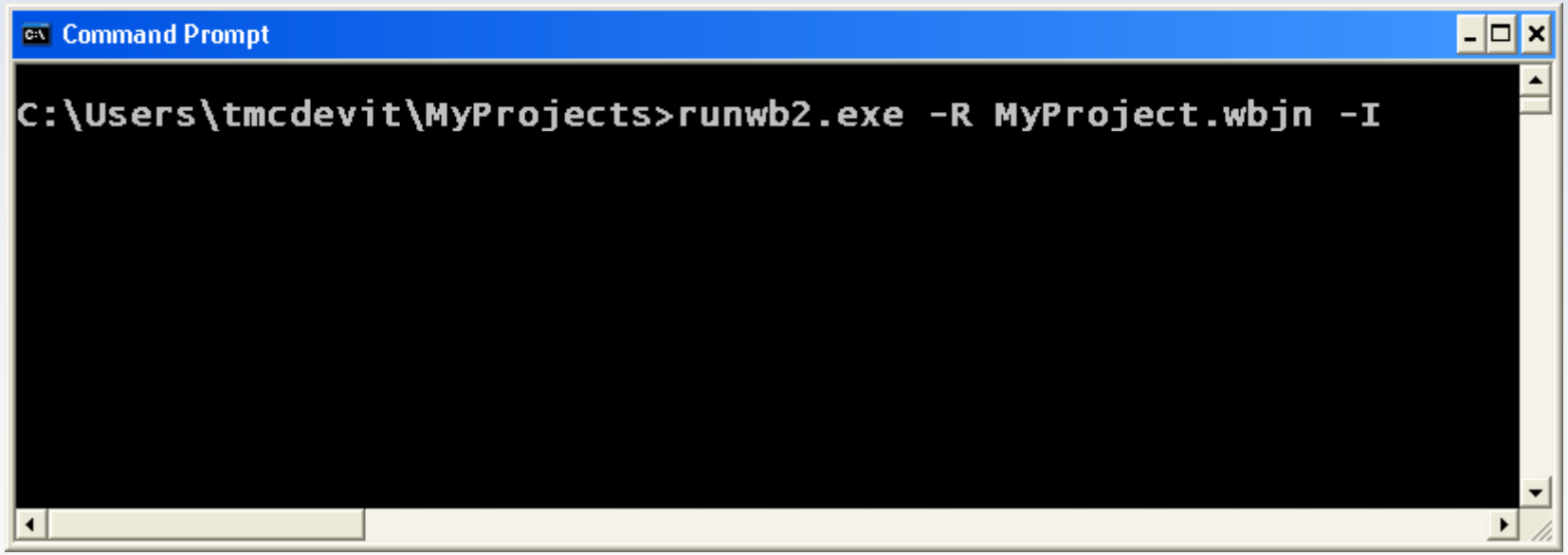
Replaying Journals

- **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

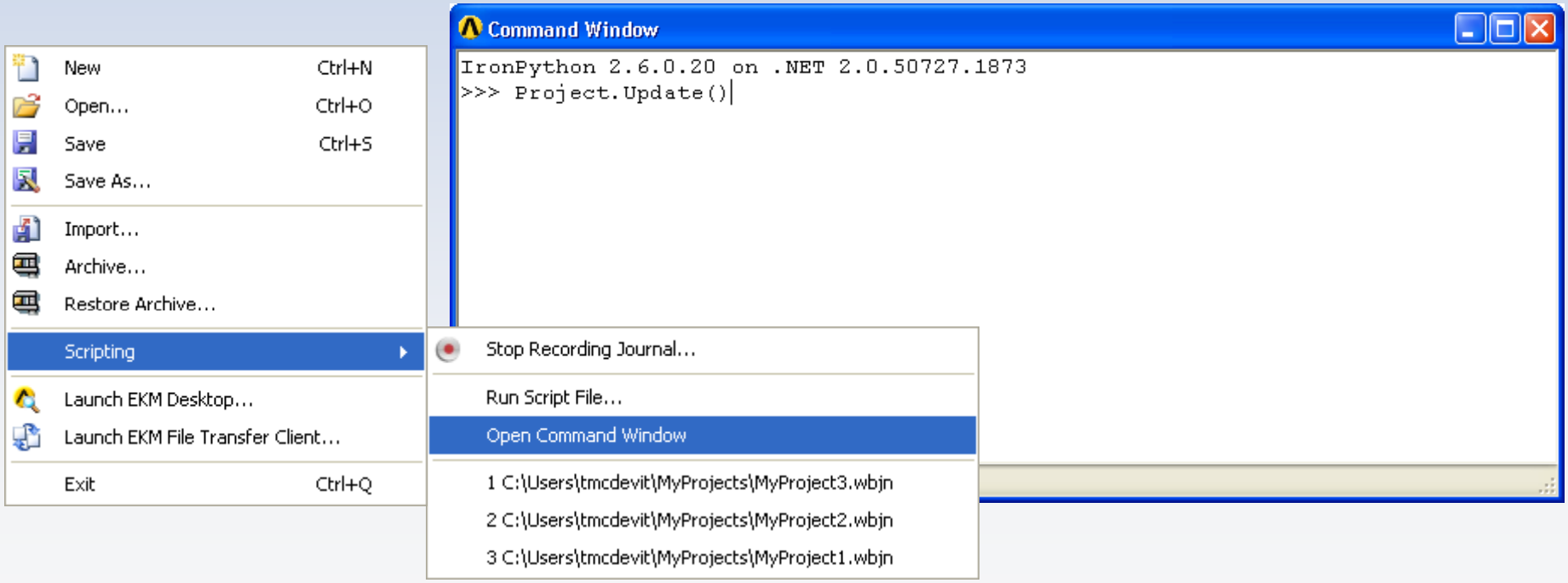


Workbench Command Line Options

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.

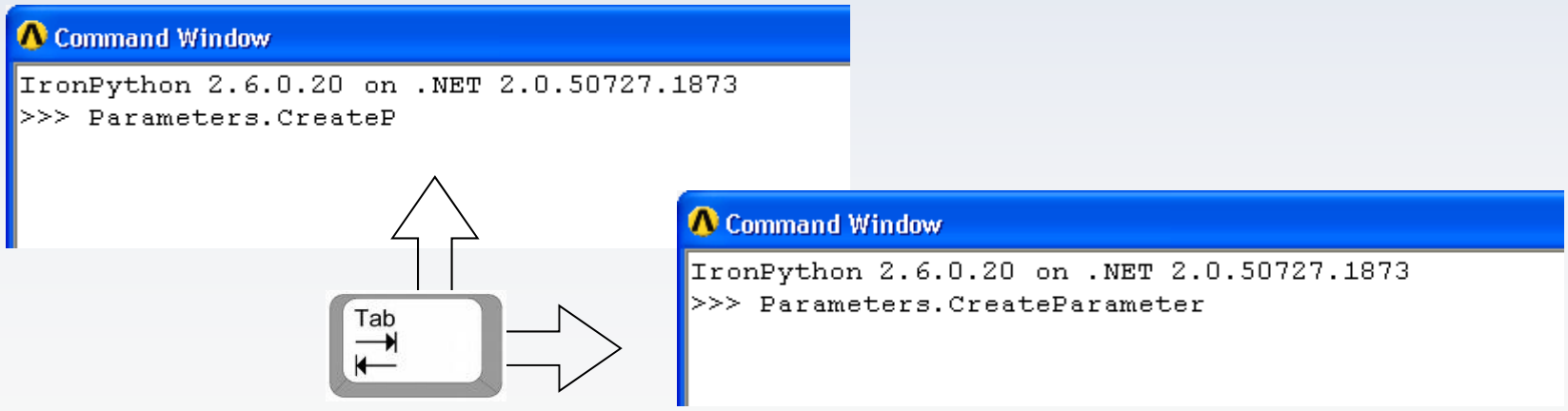
Command Window

- **Scripting commands can be entered manually in Command Window**



Command Window

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



Scripting Basics

- **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

Methods

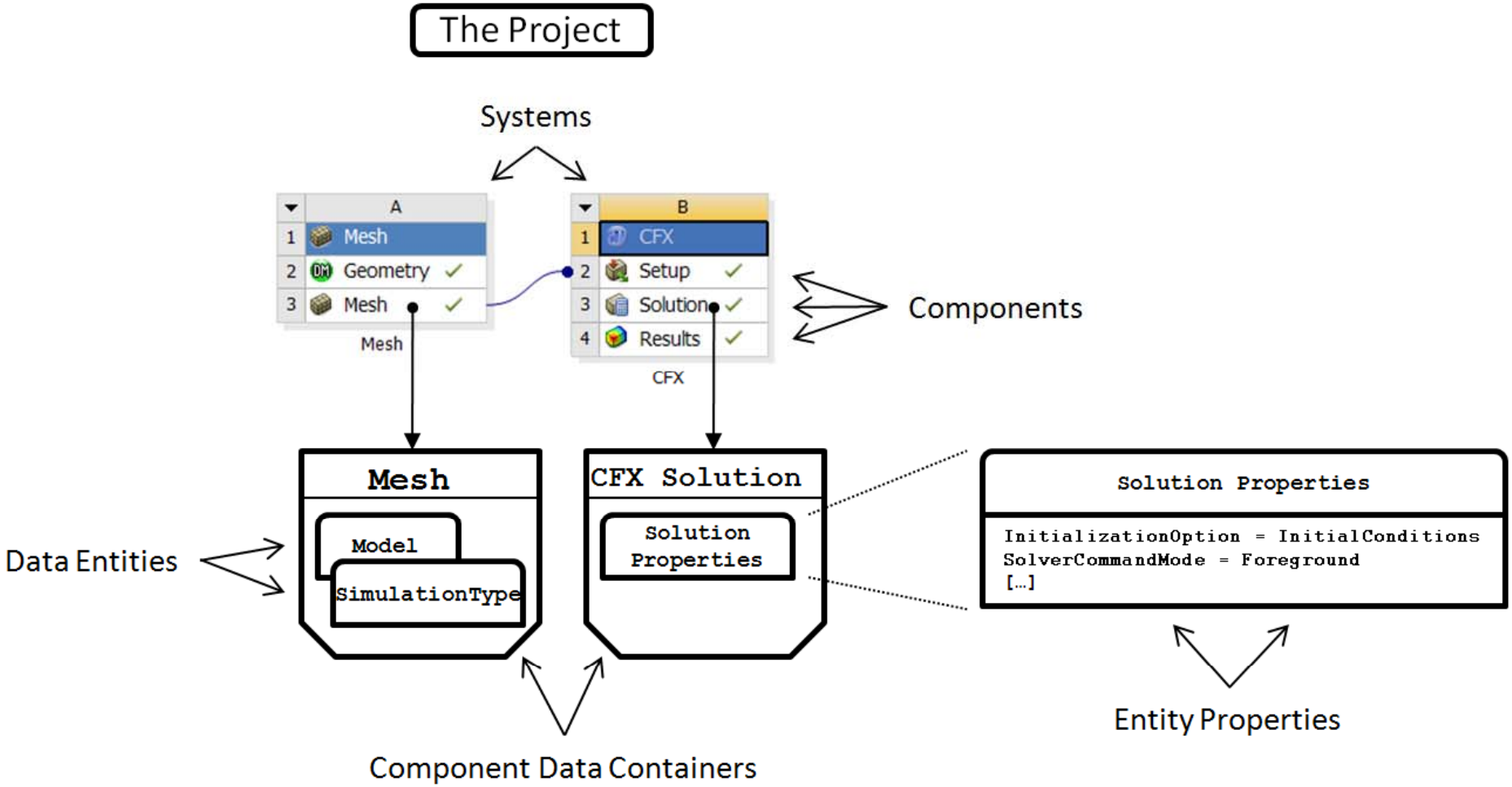
- **Method: command that is bound to an object**
  - Can change property values, create or delete properties, or invoke complex calculations and operations
- **Like properties, methods are accessed via dot operator**
  - `parameter1.SetQuantityUnits("m")`
    - parameter1 is an object of type Parameter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

Object-Based Scripting Approach

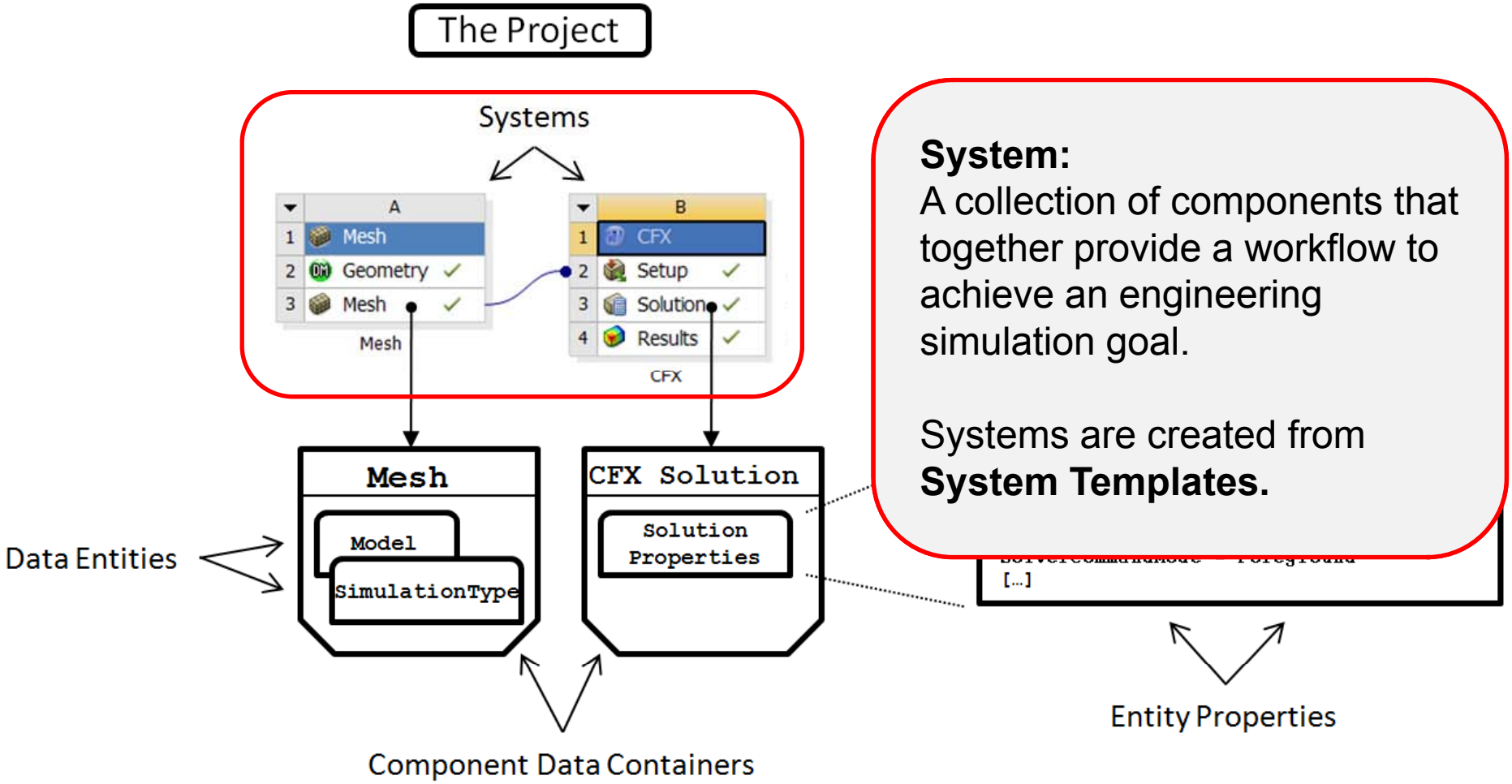
1. **Query for an object**
   - Query methods return object references that are assigned to variables
     - `Parameter = DOEModel.GetParameter(Name="P1")`
2. **Interrogate/modify object properties**
   - Apply dot operator and property name to variable
     - `Parameter.Nature = "NatureUsability"`
3. **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



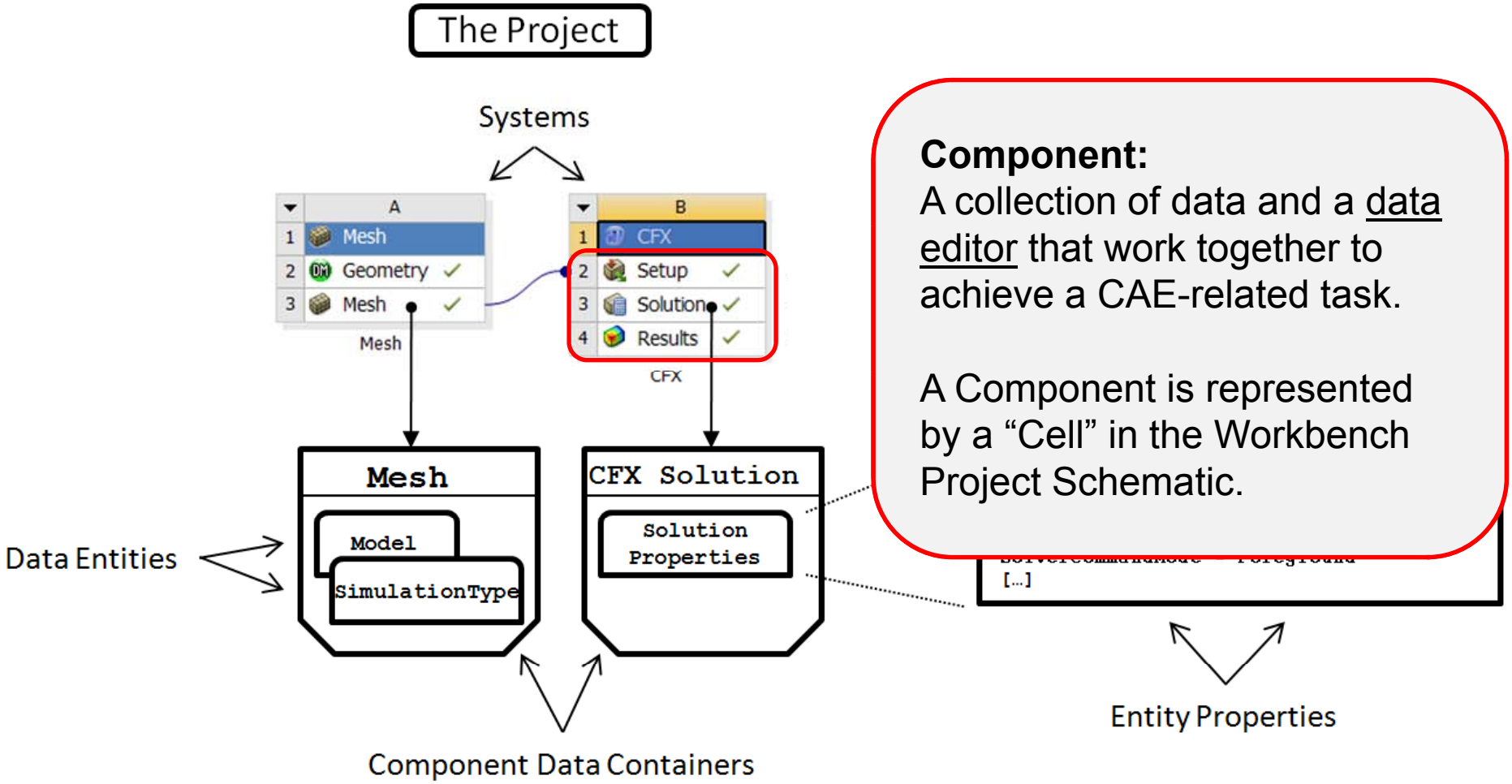
Project & Data Model Concepts



System:
A collection of components that together provide a workflow to achieve an engineering simulation goal.

Systems are created from **System Templates.**

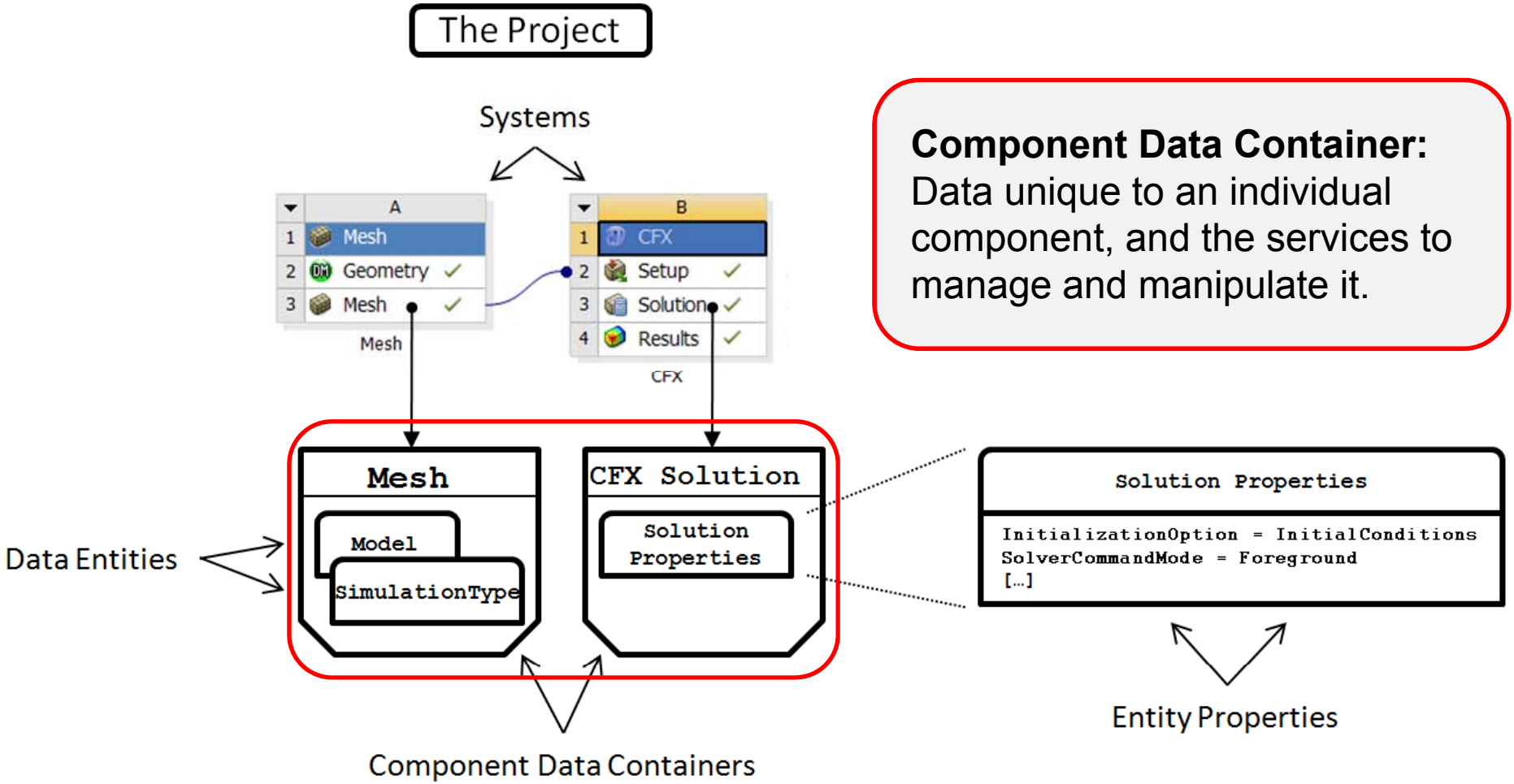
Project & Data Model Concepts



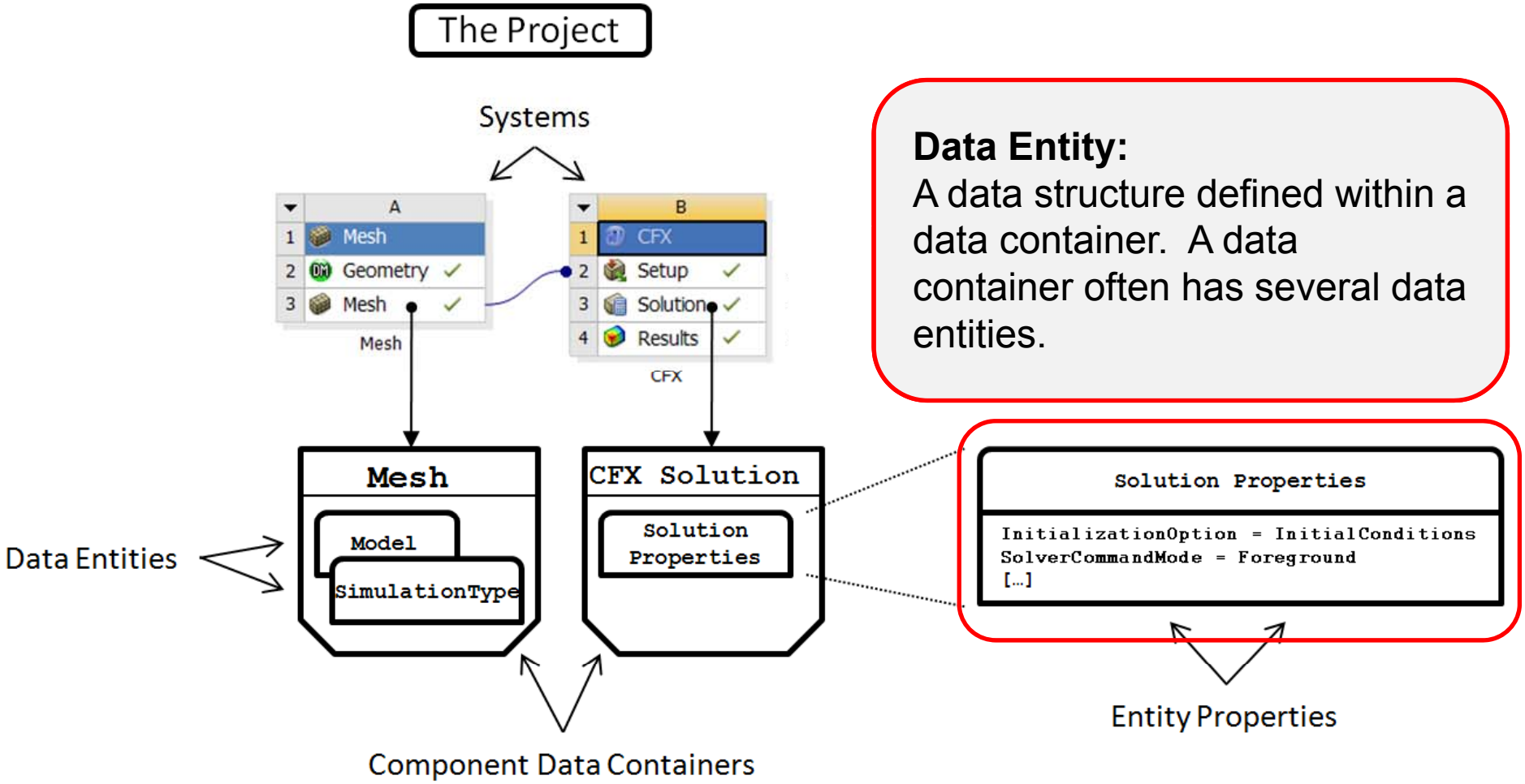
Component:
A collection of data and a data editor that work together to achieve a CAE-related task.

A Component is represented by a “Cell” in the Workbench Project Schematic.

Project & Data Model Concepts



Project & Data Model Concepts



Data Entity:
A data structure defined within a data container. A data container often has several data entities.

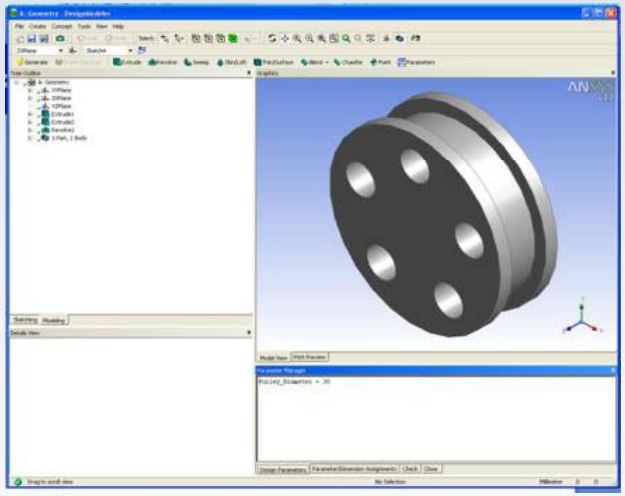
Example: Change a Material Property


```
# 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

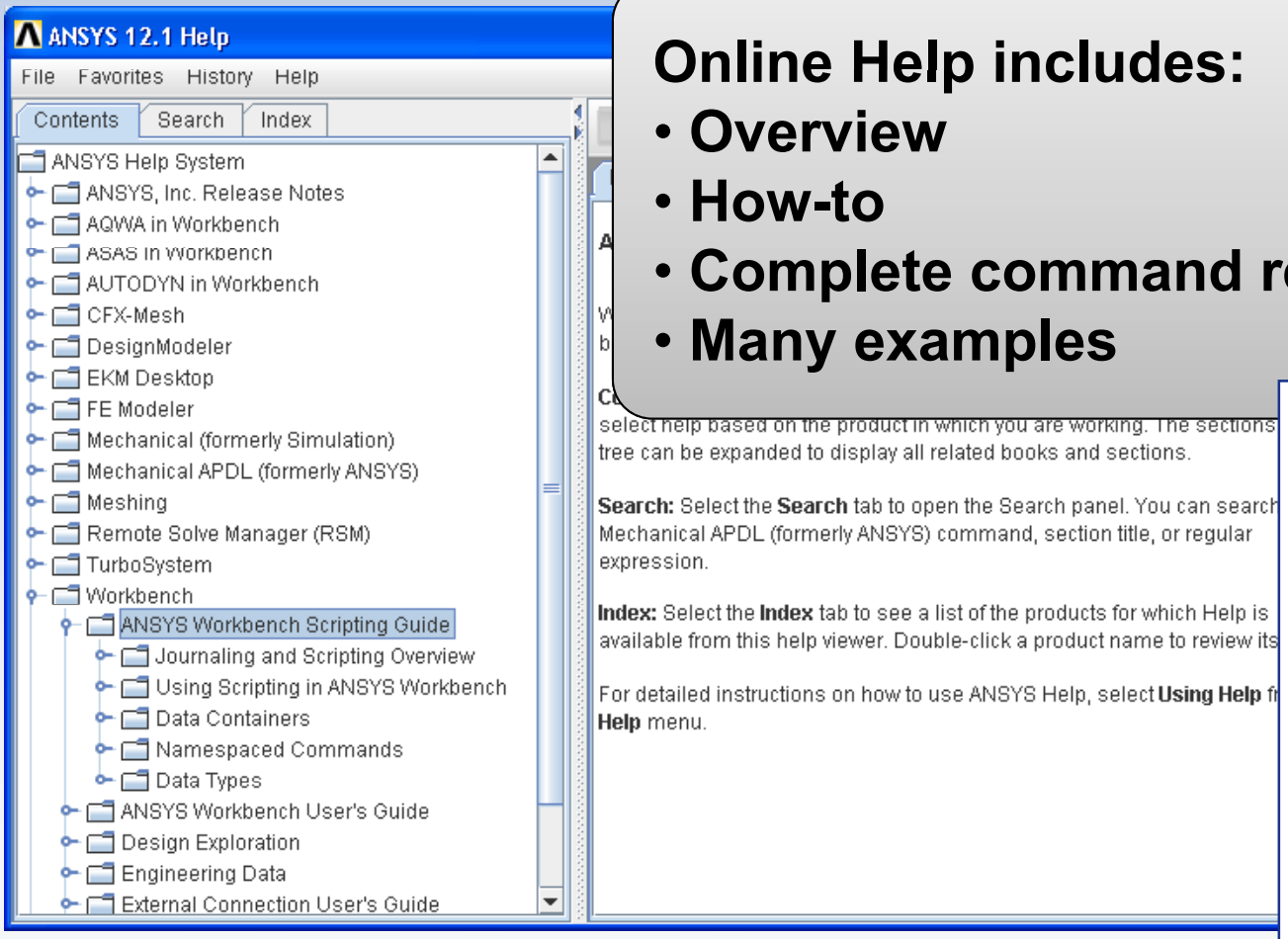
- **Application-level scripting can be embedded in a Workbench script**
  - APDL, CCL, Scheme, JScript, etc.
- **Some data-integrated applications record their operations in the Workbench journal**
  - CFX, FLUENT

Example: Create Geometry

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

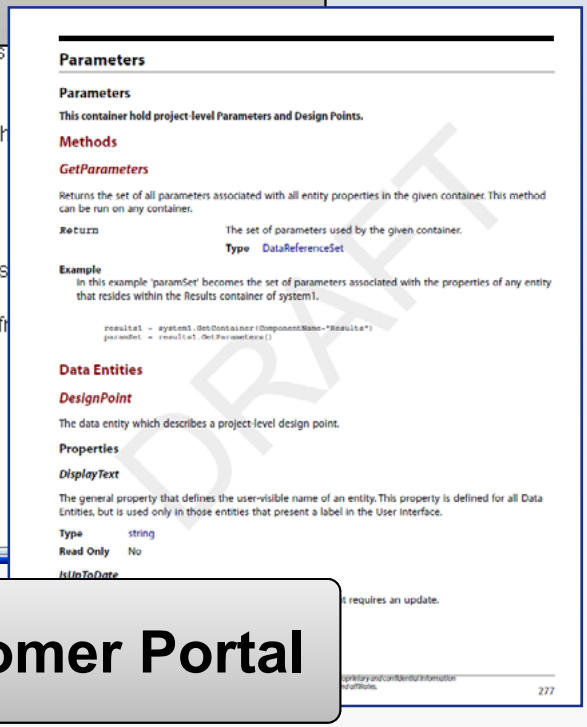
```
# create a geometry system
system = GetSystem(Name="Geom")
# query the data container of the geometry system
geometry = system.GetContainer(ComponentName="Geometry")
# send a JScript command to DM to create an ellipse
geometry.SendCommand(Command = """var ps1 = new Object();
  ps1.Plane = agb.GetActivePlane();
  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.El7 = Ellipse( 8.0, 10.0, 9.0,
  6.0, 5.0, 12.0); }
  agb.Regen();""")
```

Documentation



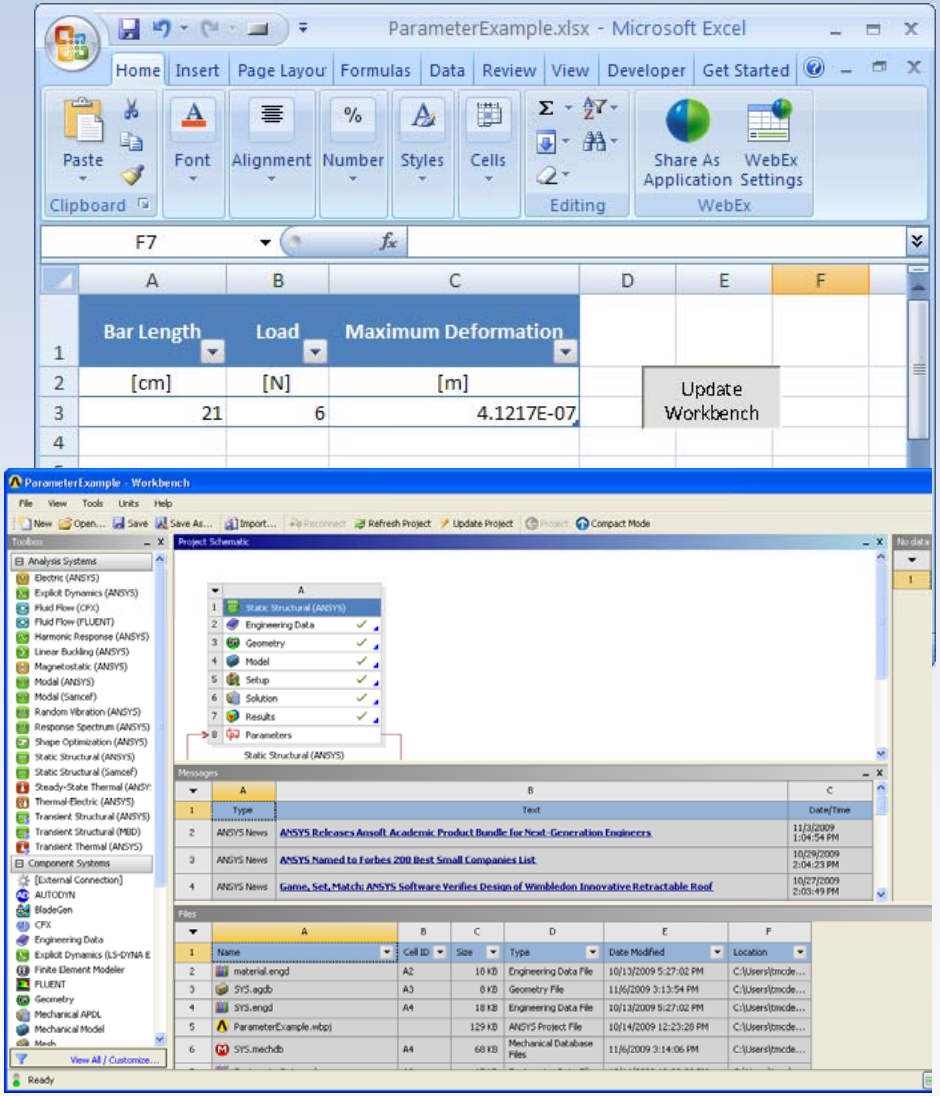
Online Help includes:

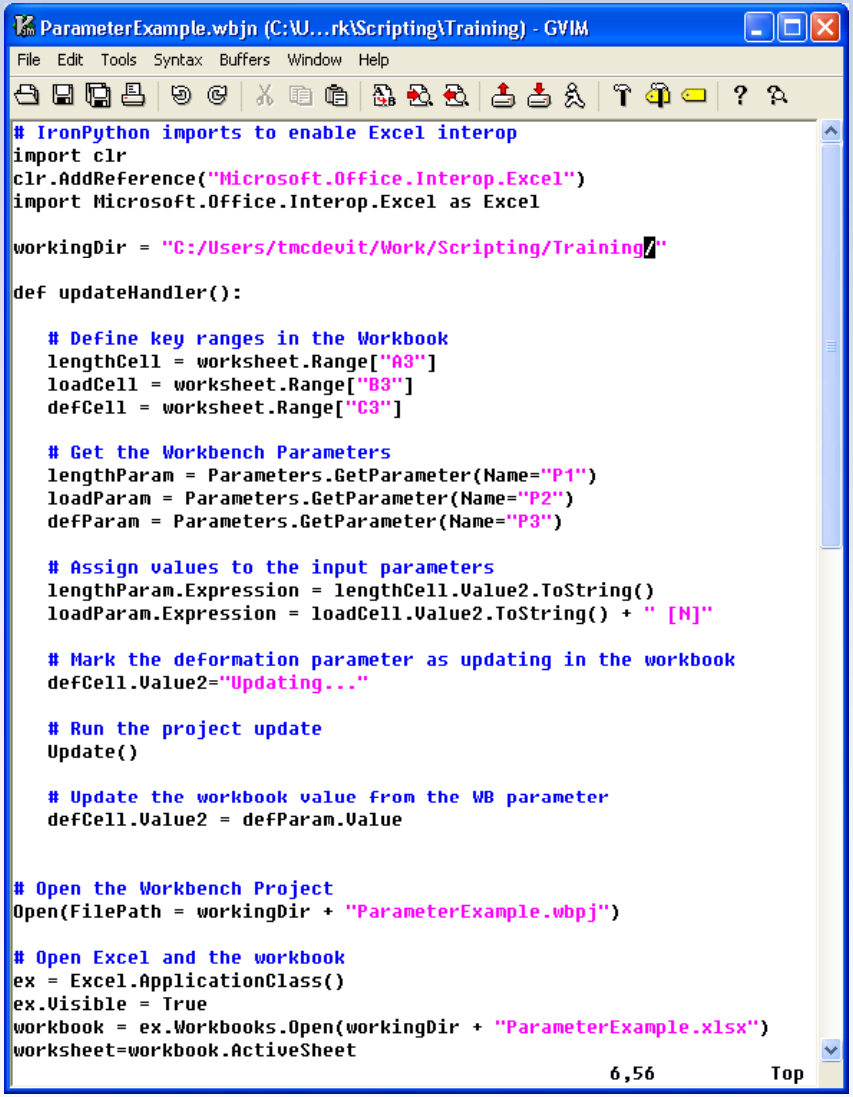
- **Overview**
- **How-to**
- **Complete command reference**
- **Many examples**



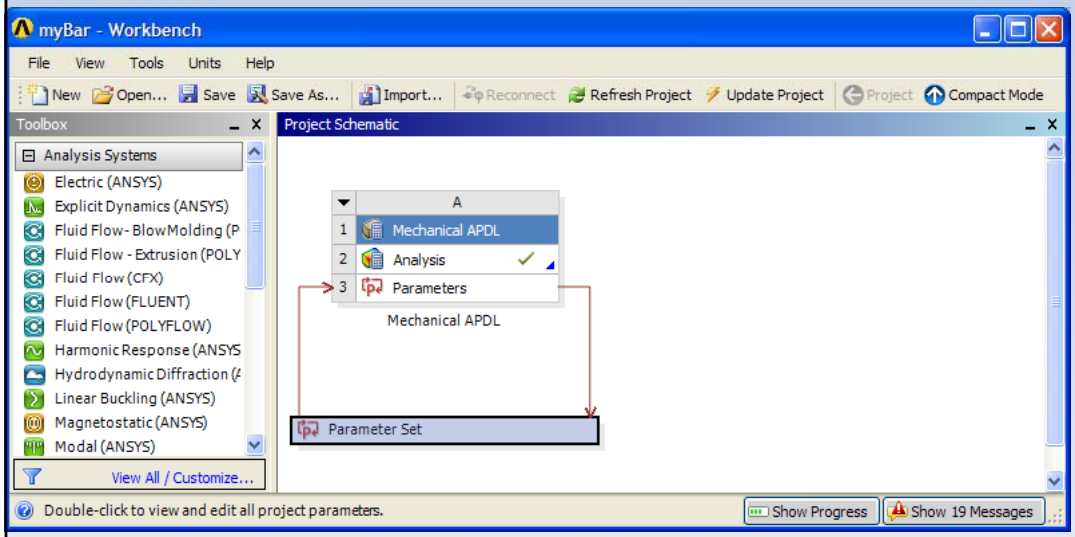
PDF version available on ANSYS Customer Portal

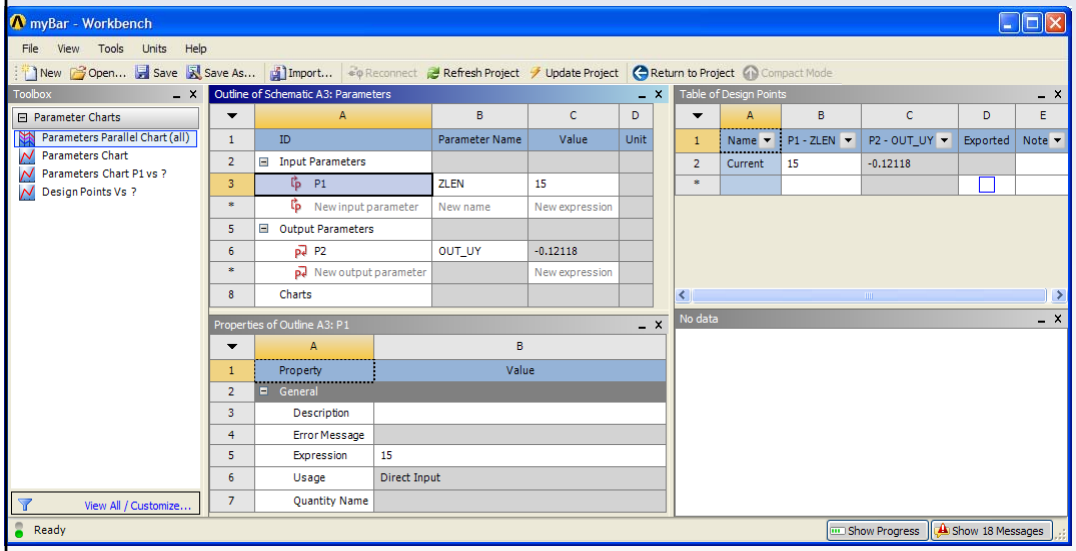
Example & Demo: Parametric Update from MS Excel





Example & Demo: Stress Analysis via APDL



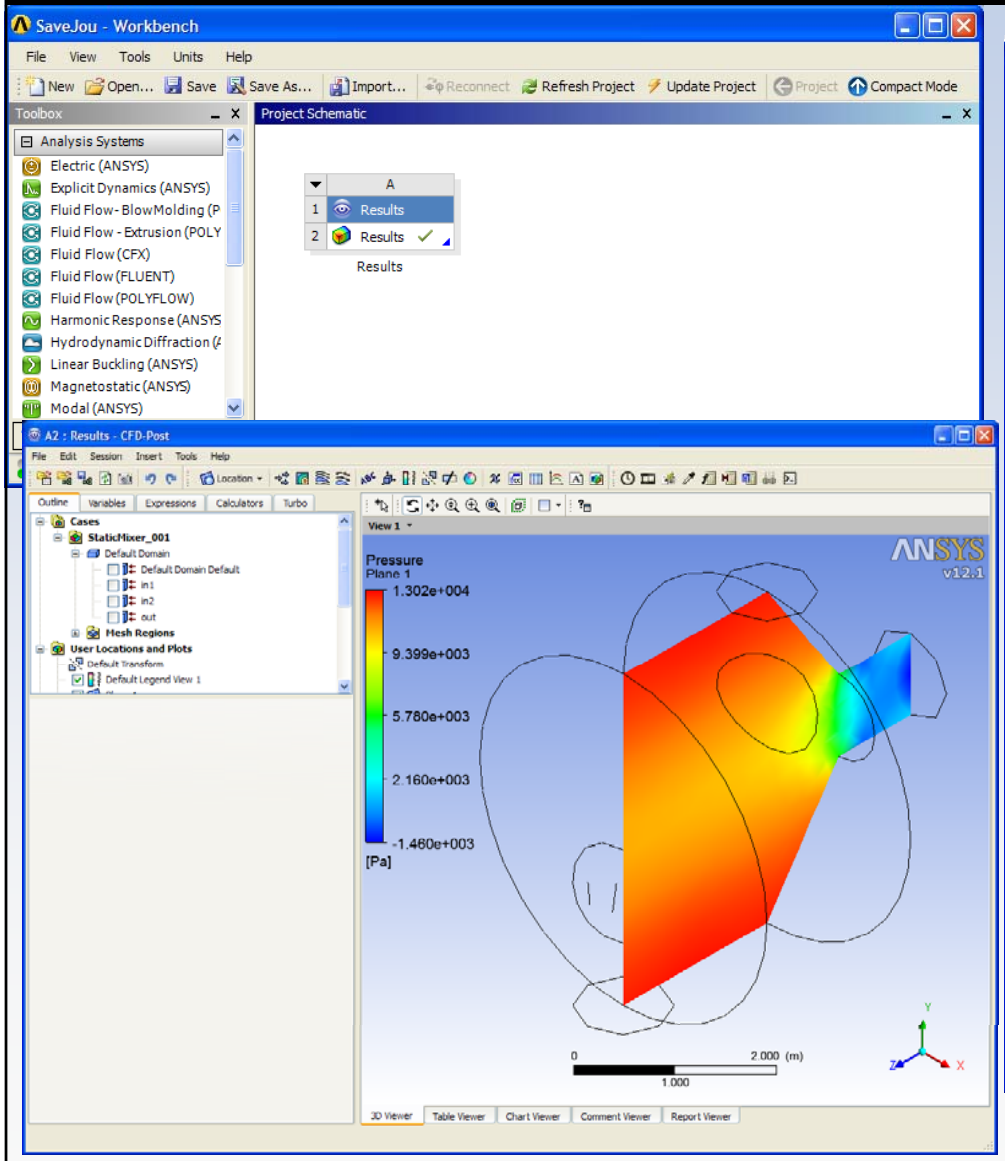


apdl.wbjn - Notepad

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 promting us about overwrite.
    if os.path.exists(cdbName):
        os.remove(cdbName)

    # Generate the APDL command to save the CDB file and send it.
    apdlCmd = "cdwr,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)
```