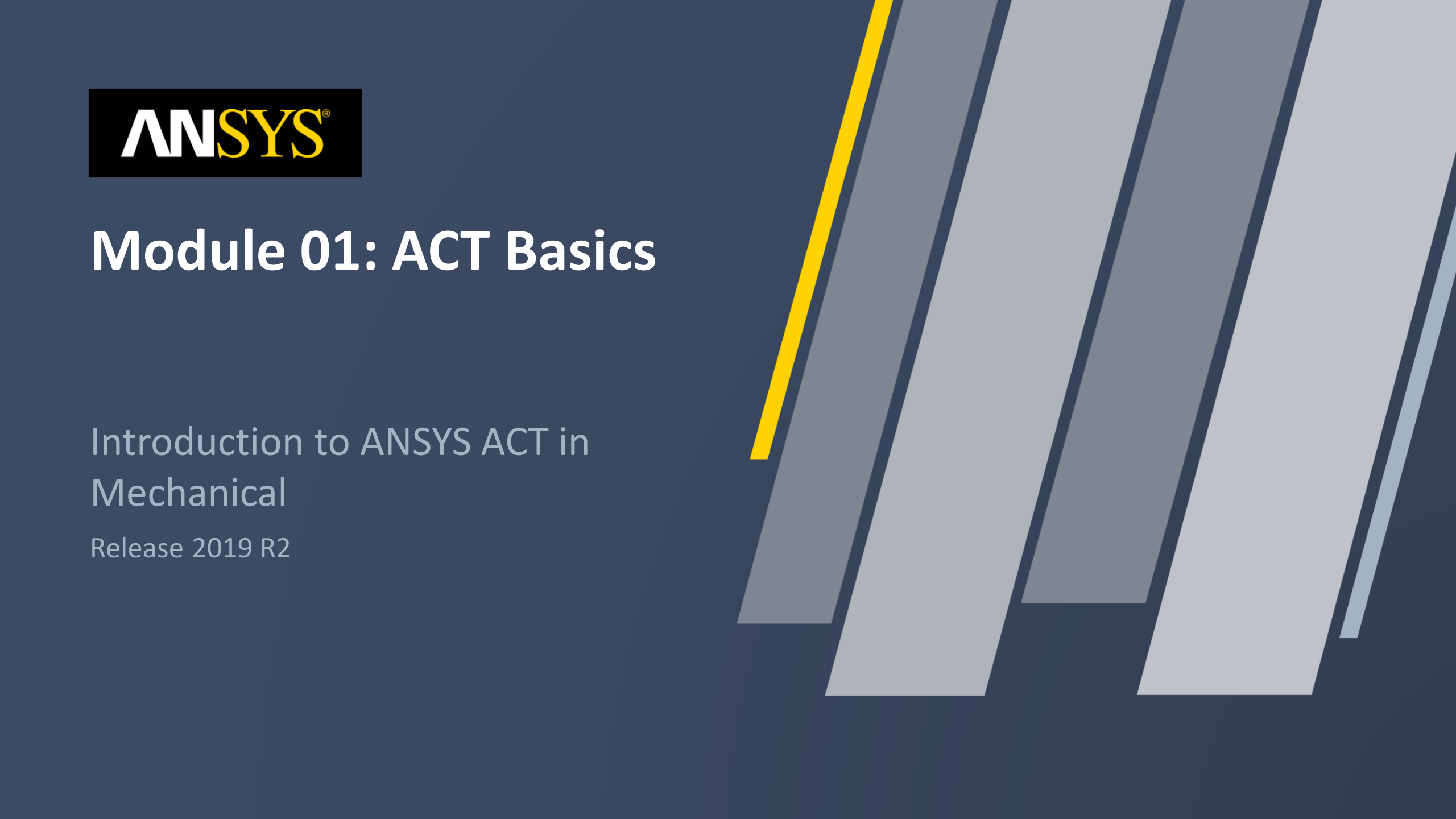


Module 01: ACT Basics

Introduction to ANSYS ACT in Mechanical

Release 2019 R2

Agenda

- ACT Overview
  - ACT for ANSYS Mechanical
  - ACT for DesignModeler
  - ACT Wizards
- Extension Basics
- Extension Types
- Menu « Extensions »
- Options « Extensions »
- Understanding an Example Extension
- Q/A

What is ACT?

- ACT = Application Customization Toolkit
- A consistent toolkit for customization
  - Available with various ANSYS products
  - Advanced capabilities for each targeted product
- An add-on module to the Workbench environment based on Python and XML
  - Programming is interactive and interpretive
  - Get, modify, and put data
  - Add new capabilities
  - Encapsulate process flow
  - Automate repetitive tasks
  - Integrate with external processes and codes
  - Be creative…

Major Steps Forward for ACT

- **At R15:**
  - Allow customization in:
    - ANSYS Mechanical
    - DesignXplorer
    - DesignModeler (Beta)
  - Works on Windows and Linux OS
- **At R16:**
  - DesignModeler released
  - Parameterize ACT properties
  - Project Schematic automation with ACT
  - Workbench wizards (Beta)
- **At R17:**
  - Custom workflows
  - Expanded API capabilities
  - New ACT Console
  - Easier access to ACT: ACT Start Page
  - Wizards released for a set of applications:
    - ANSYS Workbench, Mechanical, DesignModeler, DesignXplorer
  - Enhanced product coverage for wizards: (Beta)
    - Fluent, Fluent Meshing, SpaceClaim, Electronics Desktop
- **At R18:**
  - ACT Console enhancements
  - Automation API wrappers for custom workflow
  - Mechanical API enhancements
  - ACT App Builder (Beta)
  - ACT guided simulations (Beta)

ACT for ANSYS Mechanical

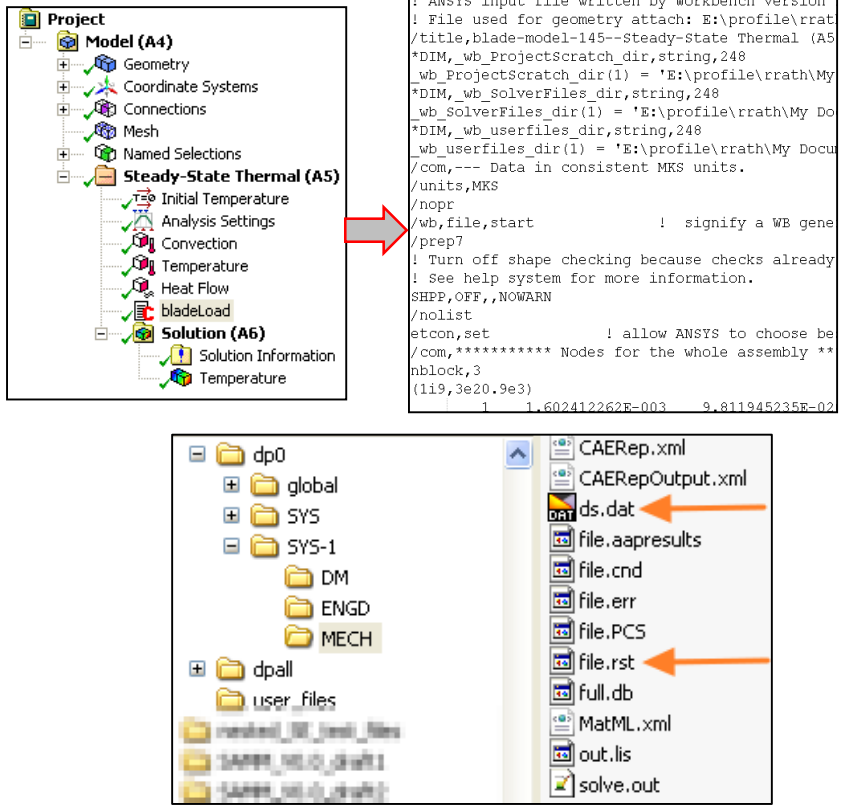
What ACT offers

- **Encapsulate APDL macros**
  - **Allows re-use of legacy APDL-scripts enabling transition to Mechanical application**
- **MAPDL exposure**
  - **Fills the gap between MAPDL solver capabilities and their exposure in ANSYS Mechanical**
- **Preprocessing features**
  - **Custom loads and boundary conditions**
- **Postprocessing features**
  - **Custom results**
- **In-house solver integration**
  - **Mechanical GUI**
- **Automation using the ACT Automation API**

ACT entry points in Mechanical

How standard Mechanical works

- When you “solve” an analysis in Mechanical, the following things happen:
  1. All items in the tree (Mesh, Contacts, Solver settings, Loads, and so on) are converted to APDL commands → written as “ds.dat”
  2. MAPDL is invoked in background, consumes this "ds.dat" and writes out the result to “file.rst”
  3. Mechanical reads back the result (file.rst) and displays contours, vectors, and so on



Where ACT comes into the picture

- ACT mostly comes into the picture in steps 1 and 3
- ACT also comes into the picture in step 2 if you use an in-house or third-party solver (instead of default MAPDL)

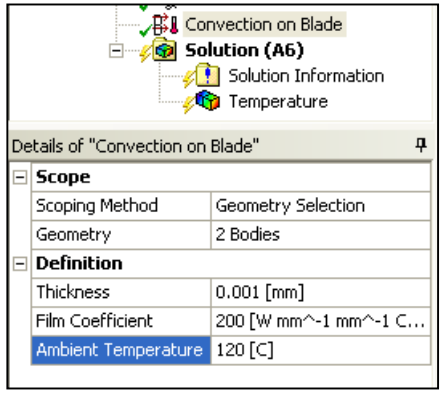
ACT entry points in Mechanical (2)

How ACT affects step 1

- **Custom loads created via ACT also need to be translated as APDL commands to go into “ds.dat”**
  - **Using Python APIs, you access all of the data (geometry, mesh, other data, inputs in custom load, and so on) and generate the desired APDL commands**
- **ACT provides a lot of control in accessing the data and where the additional commands need to go into the “ds.dat”**
- **This part requires knowledge of APDL commands**

When you “solve” an analysis in Mechanical, the following things happen:

1. **All items in the tree (Mesh, Contacts, Solver settings, Loads, and so on) are converted to APDL commands → written as “ds.dat”**
2. **MAPDL is invoked in background, consumes this "ds.dat" and writes out the result to “file.rst”**
3. **Mechanical reads back the result (file.rst) and displays contours, vectors, and so on**




```
/com,*********** Conv. Blade Load Convection on Blade ***********
/com,
thickness=0.0254
film_coefficient=200
temperature=120

/prep7
CMBLOCK,boltConv1,ELEMENT,      180
(8i10)
       191       192       193       194       195       196       197
       199       200       201       202       203       204       205
       207       208       209       210       211       212       213
```

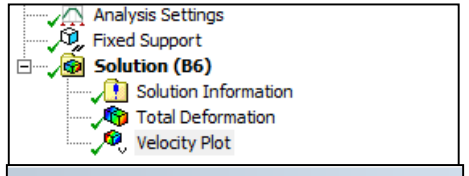
ACT entry points in Mechanical (3)

How ACT affects step 3

- Once results are available (from file.rst), custom result objects created via ACT access those results using ACT Python APIs
- ACT provides the mechanism to generate scalar/vector/tensor results at nodes/elements by using the available results
- You can use external code along with the Python APIs to generate required results
- APDL commands are NOT required
  - Although, optionally, you can use APDL commands to generate required results
    - This approach is less efficient than using ACT Python APIs directly

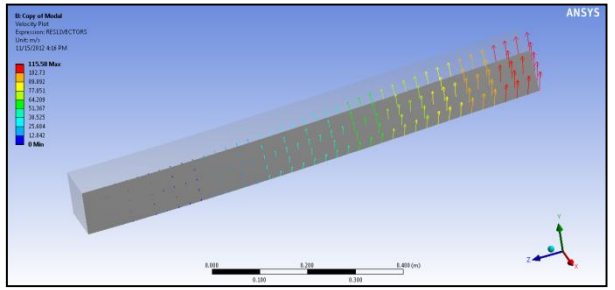
When you "solve" an analysis in Mechanical, the following things happen:

1. All items in the tree (Mesh, Contacts, Solver settings, Loads, and so on) are converted to APDL commands → written as "ds.dat"
2. MAPDL is invoked in background, consumes this "ds.dat" and writes out the result to "file.rst"
3. Mechanical reads back the result (file.rst) and displays contours, vectors, and so on



Details of "Velocity Plot"

Geometry	
Scoping Method	Geometry Selection
Geometry	1 Face
Definition	
Substep	1.
Results	
Minimum	0. m/s
Maximum	115.58 m/s



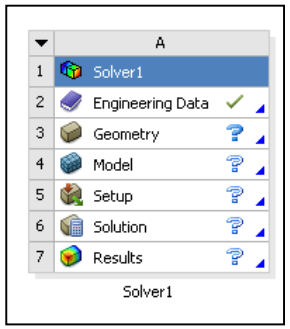
ACT entry points in Mechanical (4)

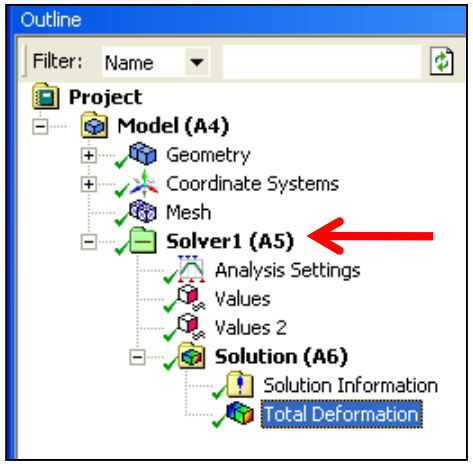
How ACT affects step 2

- **In this step, you can invoke any in-house or third-party solver (instead of MAPDL) for the solution**
  - **This step is not of any importance when the standard MAPDL solver is used**
- **You can essentially create your own input file format (instead of "ds.dat" for MAPDL) from the items in the tree**
- **ACT provides the mechanism and controls in writing the input, invoking the external solver, and reading results back**
- **As the solver is not MAPDL, APDL commands are NOT required**

When you "solve" an analysis in Mechanical, the following things happen:

1. **All items in the tree (Mesh, Contacts, Solver settings, Loads, and so on) are converted to APDL commands → written as "ds.dat"**
2. **MAPDL is invoked in background, consumes this "ds.dat" and writes out the result to "file.rst"**
3. **Mechanical reads back the result (file.rst) and displays contours, vectors, and so on**





What's more…

Apart from these 3 steps, ACT also provides access to:

- Mechanical Graphics for
  - Displaying desired information (such as drawing lines, surfaces, texts, and so on)
  - Displaying load variation for an custom load
- Mechanical Automation for
  - Automating a process in Mechanical (model setup, postprocessing, and so on)
  - Getting or setting data relative to native Mechanical objects
- Mechanical Application for
  - Invoking JScript commands if required
  - Publishing warnings and errors from ACT objects
- Workbench application for
  - Access to Engineering Data
  - Invoking Python commands to get data, perform automation, or so on
- External Python modules for many different purposes, including:
  - Displaying GUI panels
  - Plotting graphs

Mechanical Automation with ACT

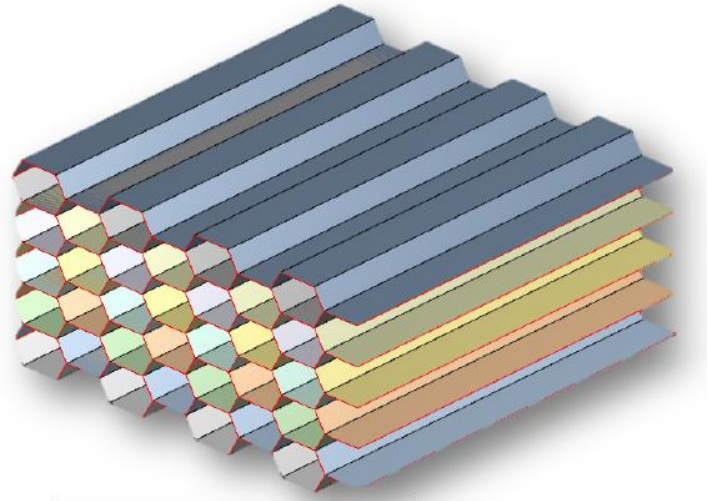
- The ACT Automation API allows you to automate a process in Mechanical, which previously required the use of Jscript
- Automation can be addressed using IronPython only
- ACT creates a set of customization possibilities
  - ACT typically adds preprocessing and postprocessing features
- ACT is able to manage standard features of Mechanical using the Automation API
  - ACT typically allows the use of standard preprocessing or postprocessing features
- JScript can still leverage new features exposed via ACT when missing
- Thanks to the Automation API, the three examples below can be addressed using ACT only:
  - In your workflow, you want to automatically insert a few mesh controls and already available boundary conditions [fixed support, pressure load, and so on]
  - In your workflow, you want to expose some new type load [such as super-element, acoustics loads, and so on] or functionality [typically for users to use interactively]
  - In your workflow, you want to expose some new features and automate existing features and these new features

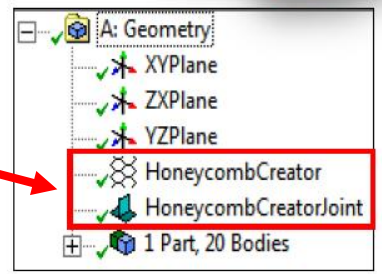
ANSYS®

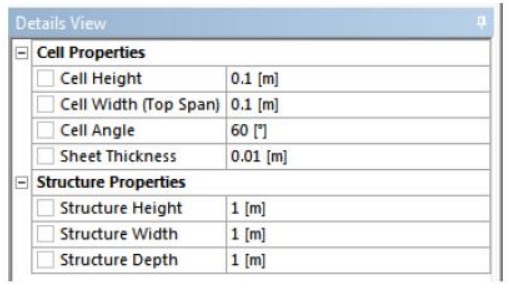
ACT for DesignModeler

What ACT for DesignModeler offers

- **New objects for specific geometry creation**
  - **Family of parts or assemblies**
  - **Industry-specific or company-specific geometry creation customization**
    - **Appears as a single ACT object in the tree**
  - **Automated geometry simplification**
  - **Customized application-specific mesh preparation**



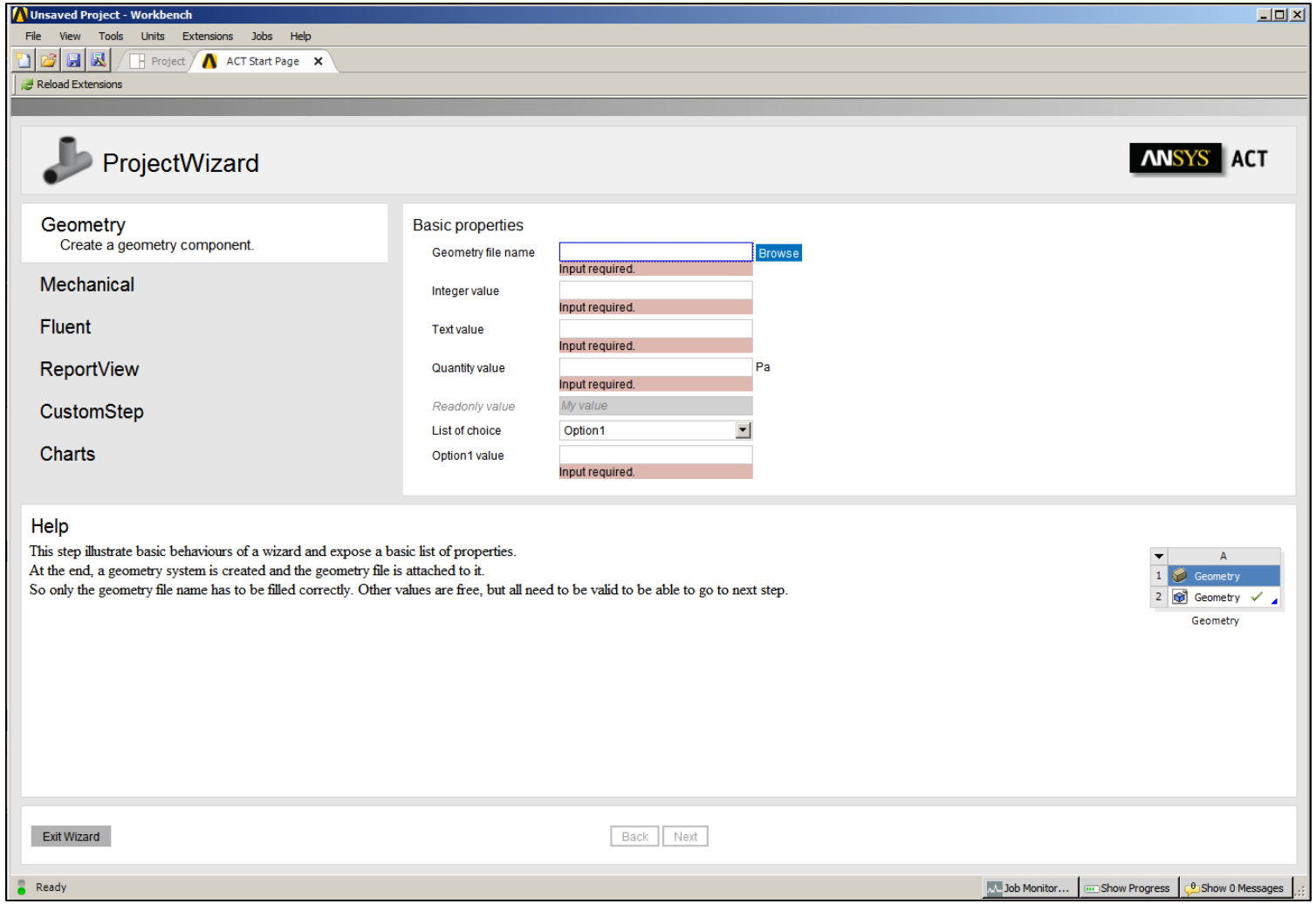




ACT Wizards

What ACT Wizards offers

- Can capture industry-specific or company-specific simulation best practices
- Enables you to integrate, automate, and streamline company-specific best practices
- Can be deployed as an app for engineers, designers, and analysts

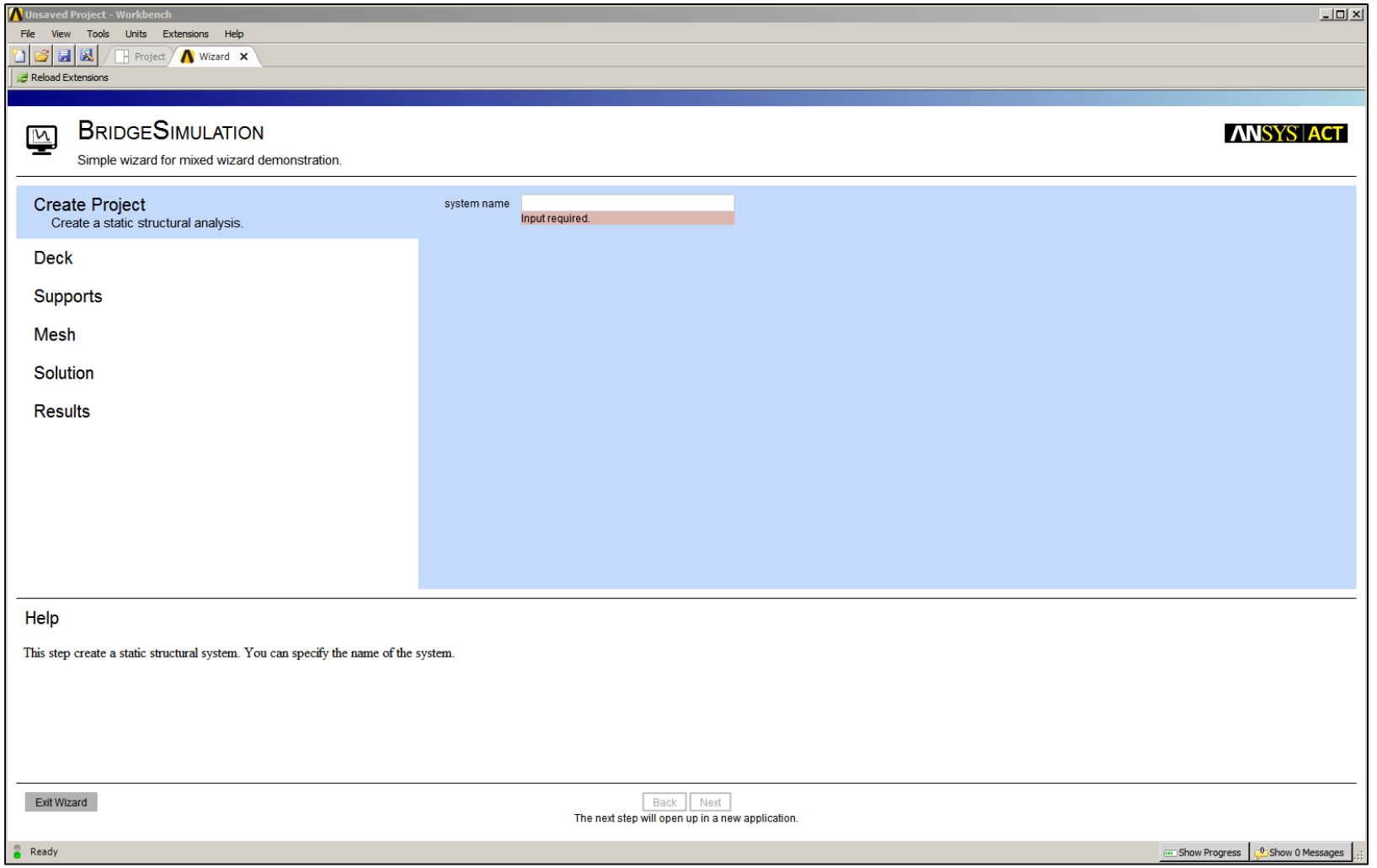


Wizards at a glance…

Below are a few ideas to keep in mind before starting with wizards:

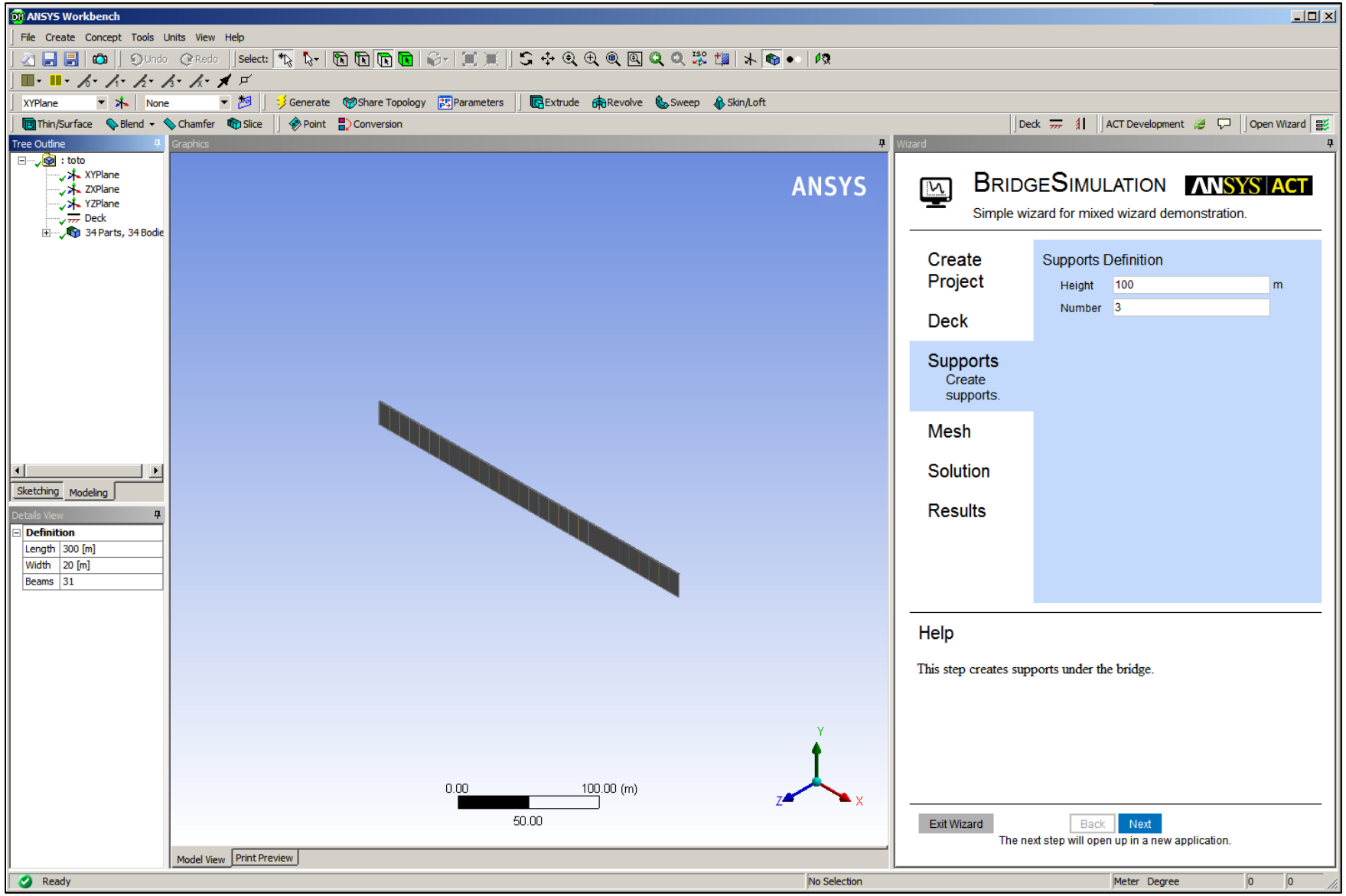
- **By nature, a wizard is an extension**
- **The extension that defines the wizard needs first to be loaded using the Extension Manager**
- **Wizards are launched from the Wizard Page, which is available from the ACT Start Page**
- **Wizards can be implemented at the Project page level or in a set of target applications: Mechanical, DesignModeler, SpaceClaim, Electronics Desktop, Fluent, Fluent Meshing, and AIM**
- **Wizards can combine different applications**

Wizards at the project page level

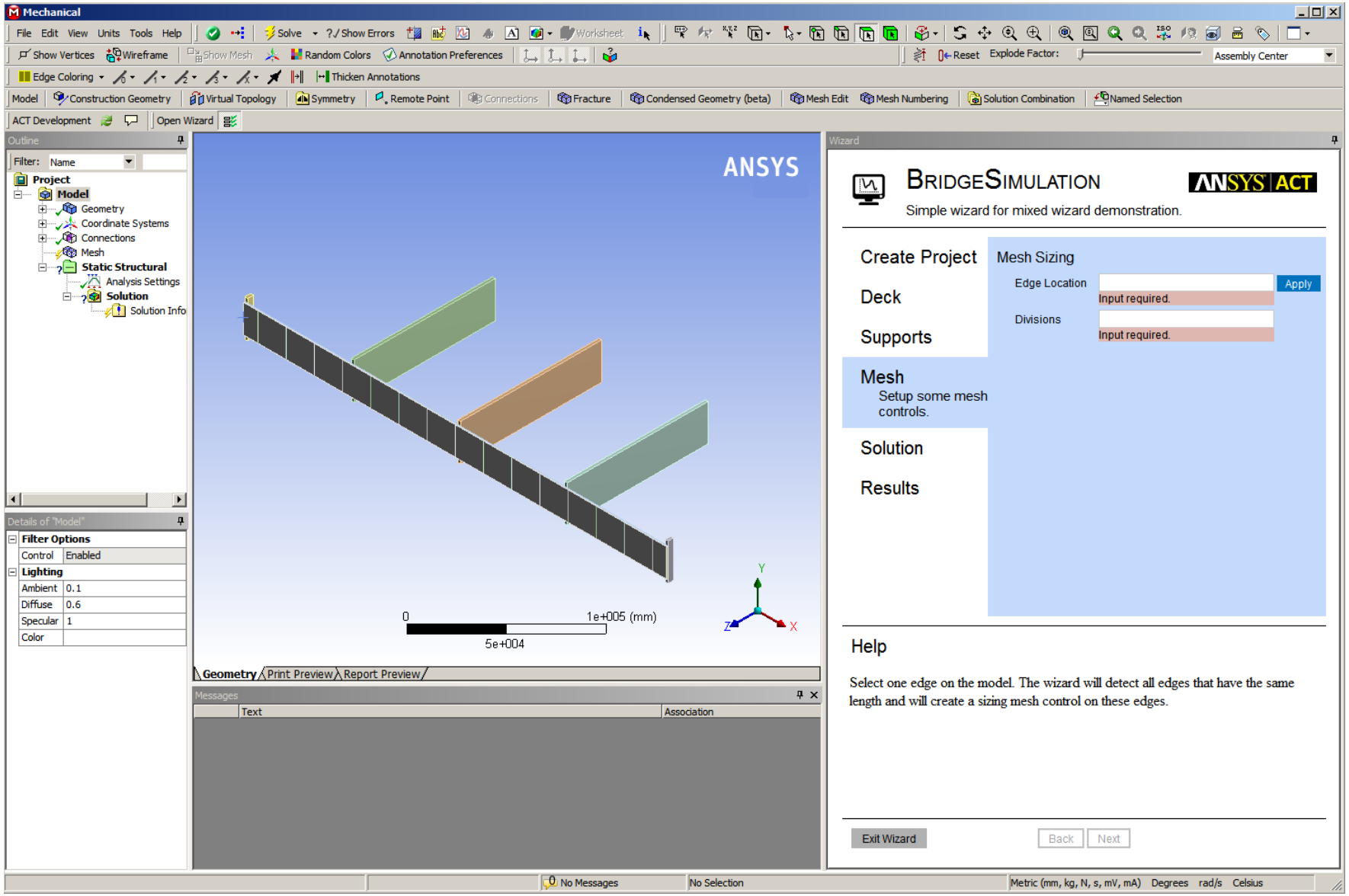


HTML-based generic layout

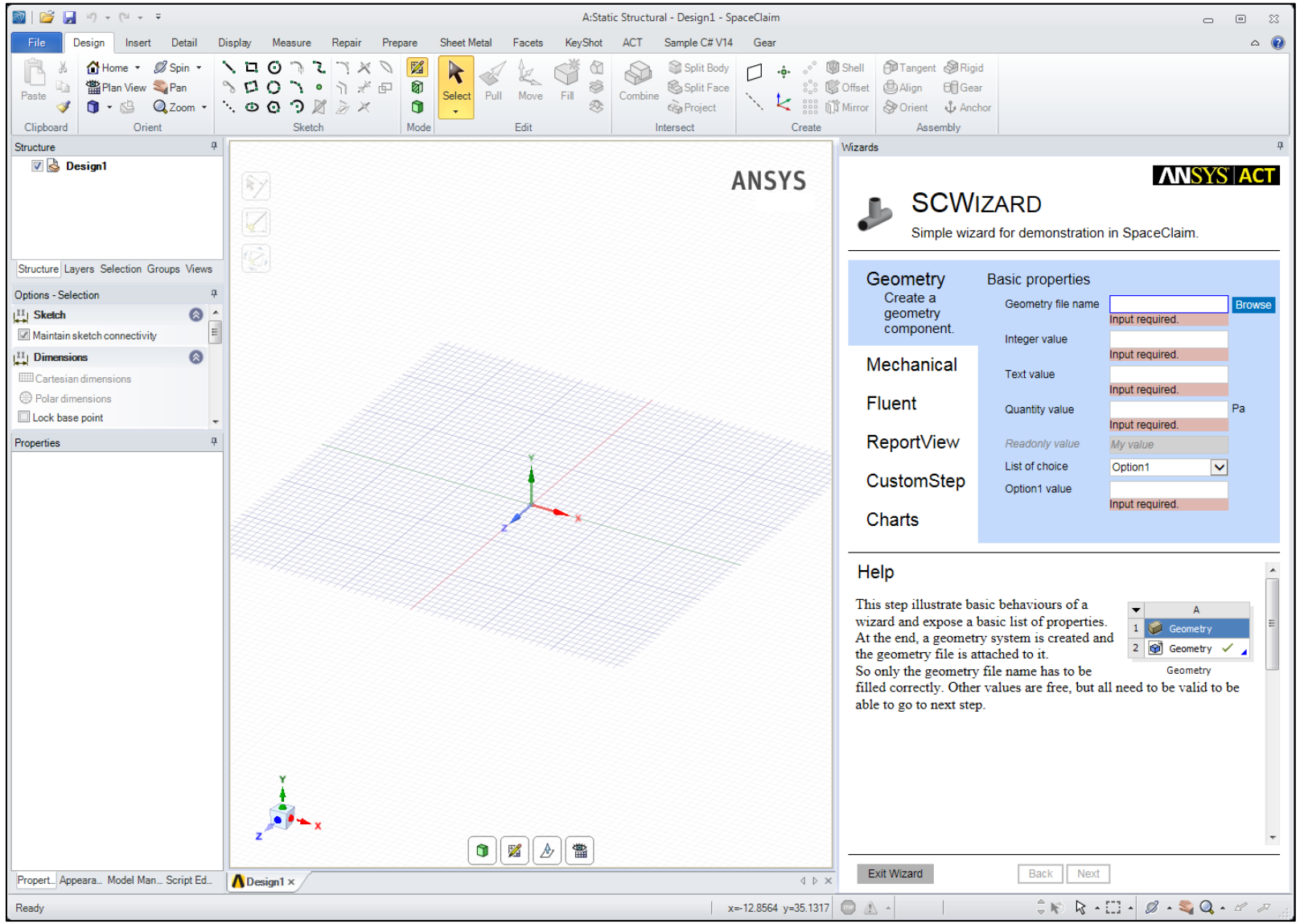
Wizards in DesignModeler



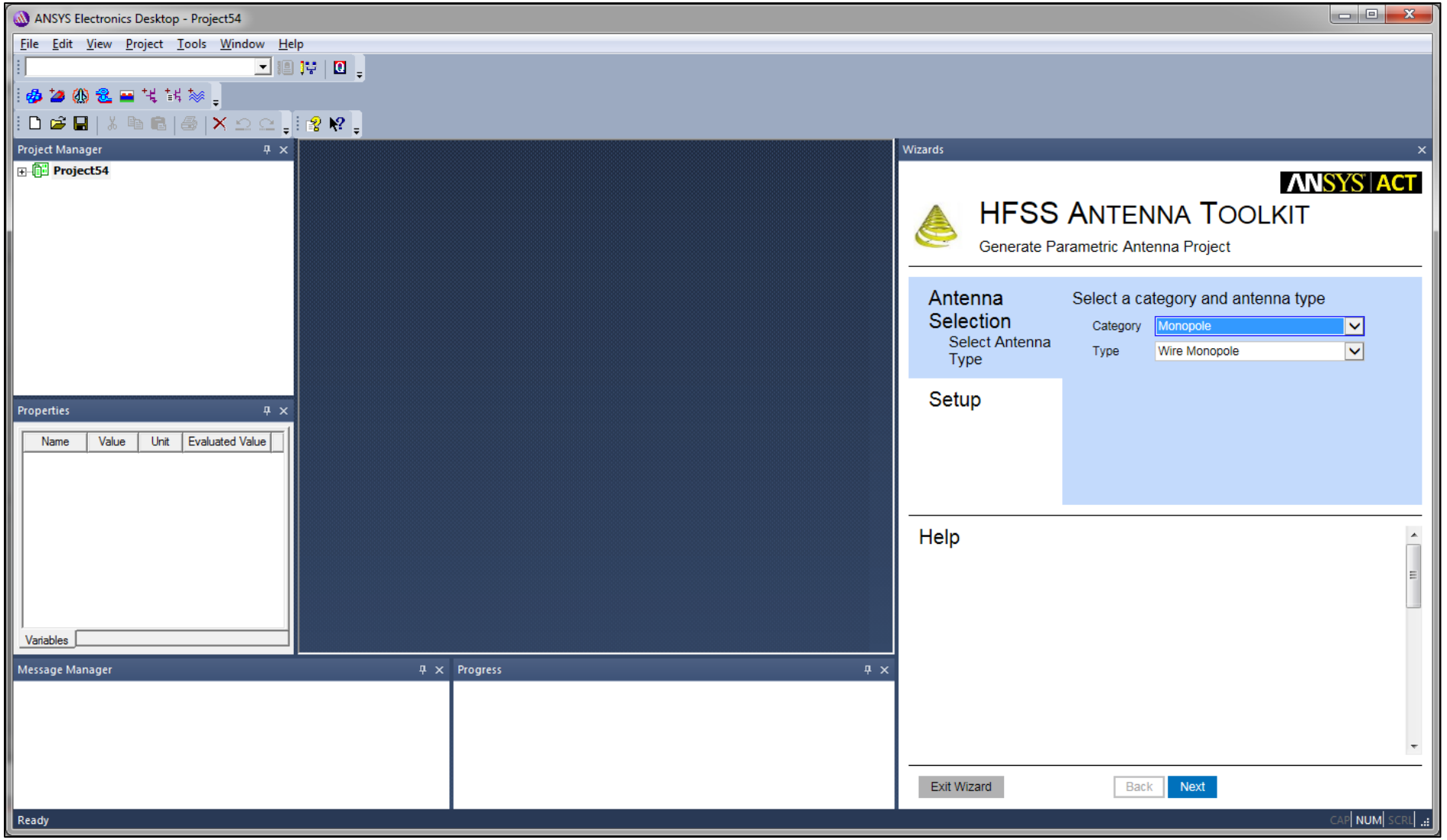
Wizards in Mechanical



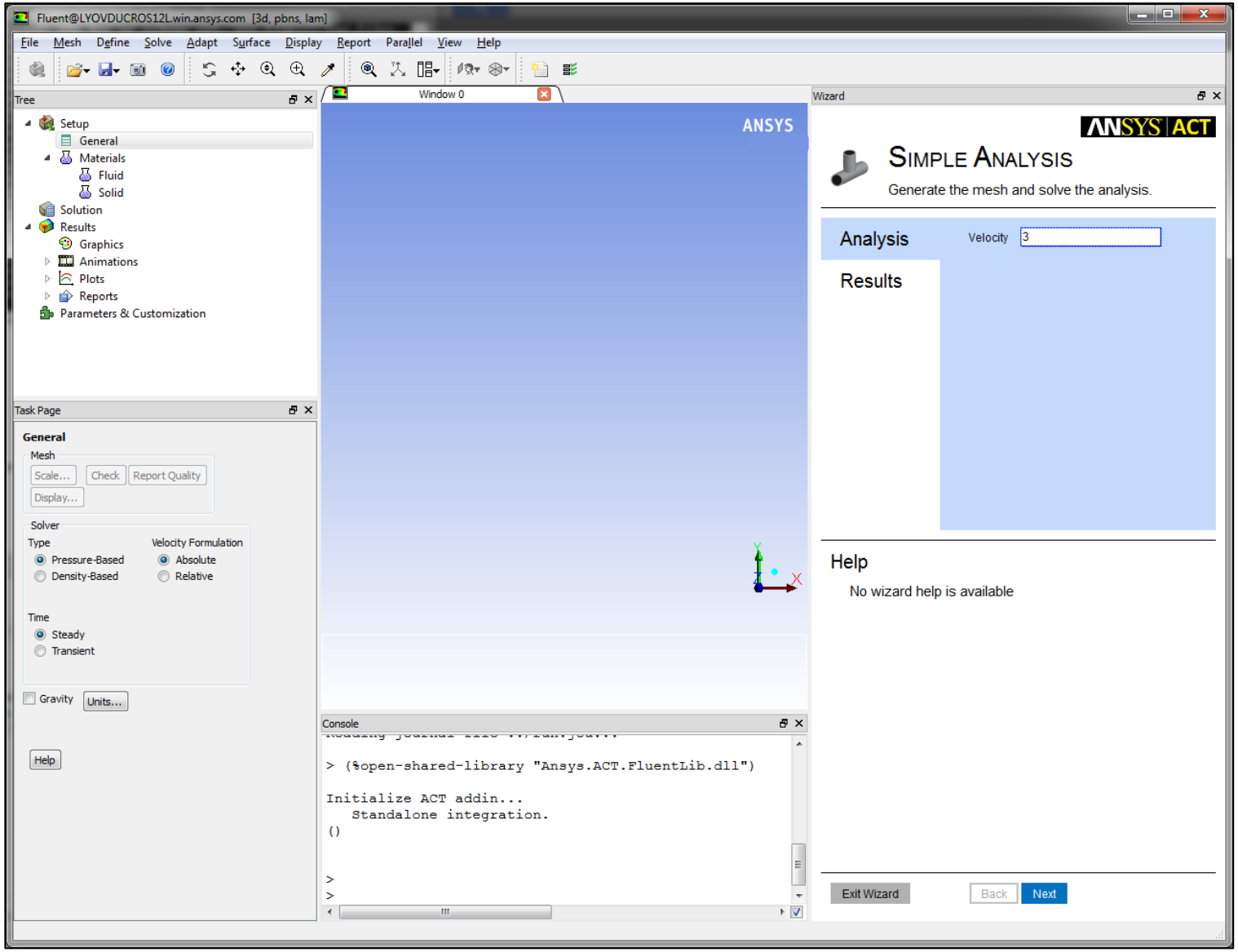
Wizards in SpaceClaim



Wizards in Electronics Desktop

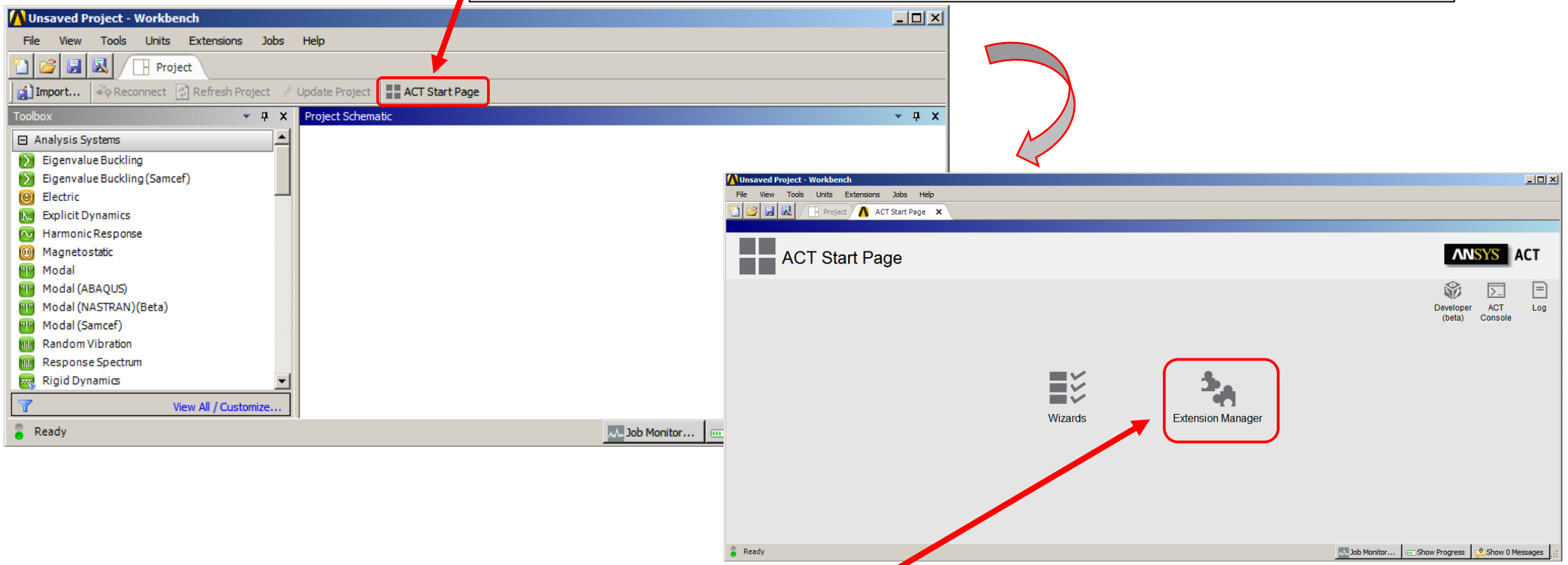


Wizards in Fluent



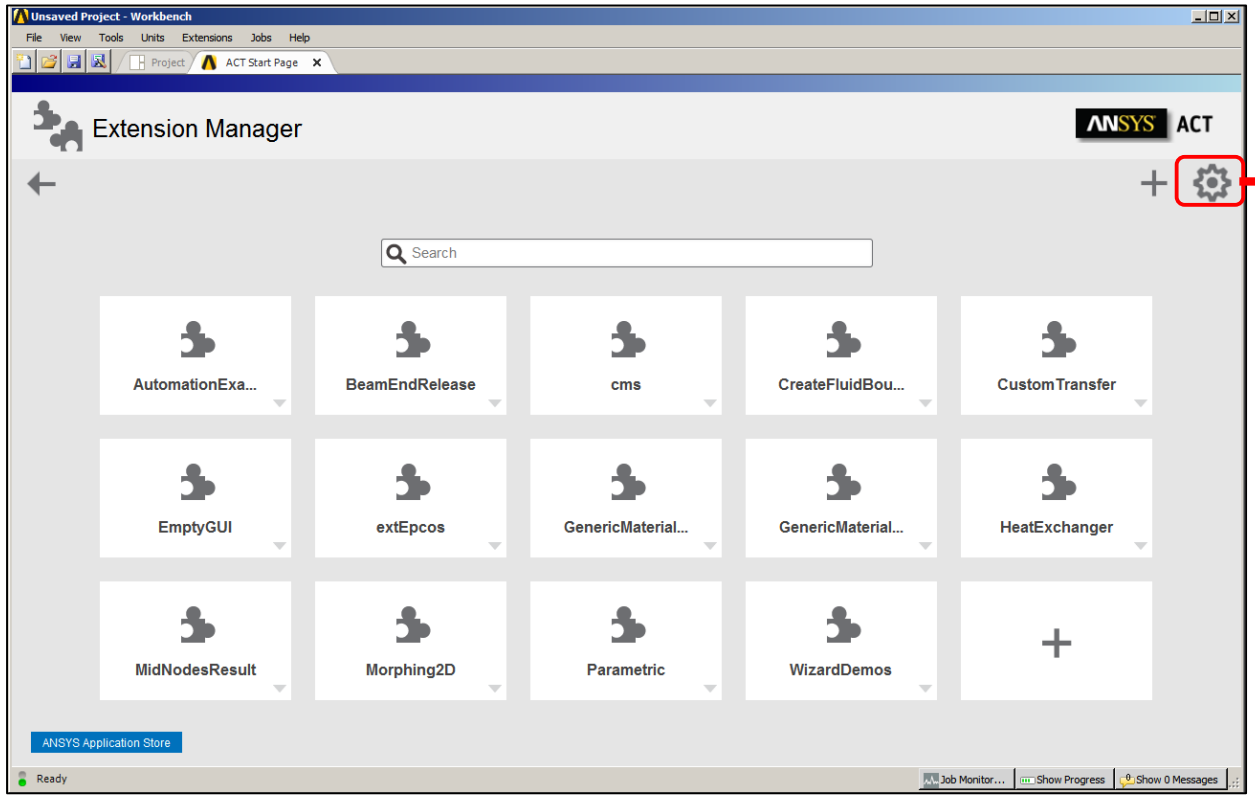
Get access to wizards

1 – Launch the ACT Start Page by clicking the ACT Start Page button

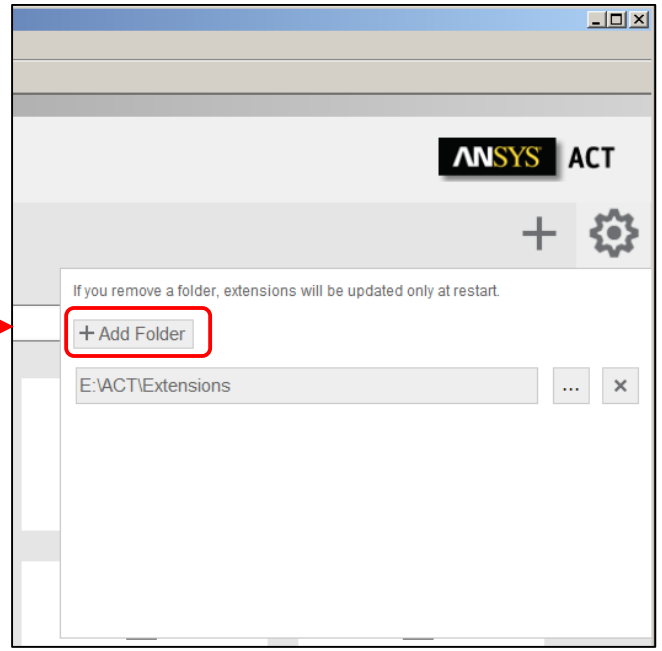


2 – Get access to the Extension Manager

Get access to wizards (2)



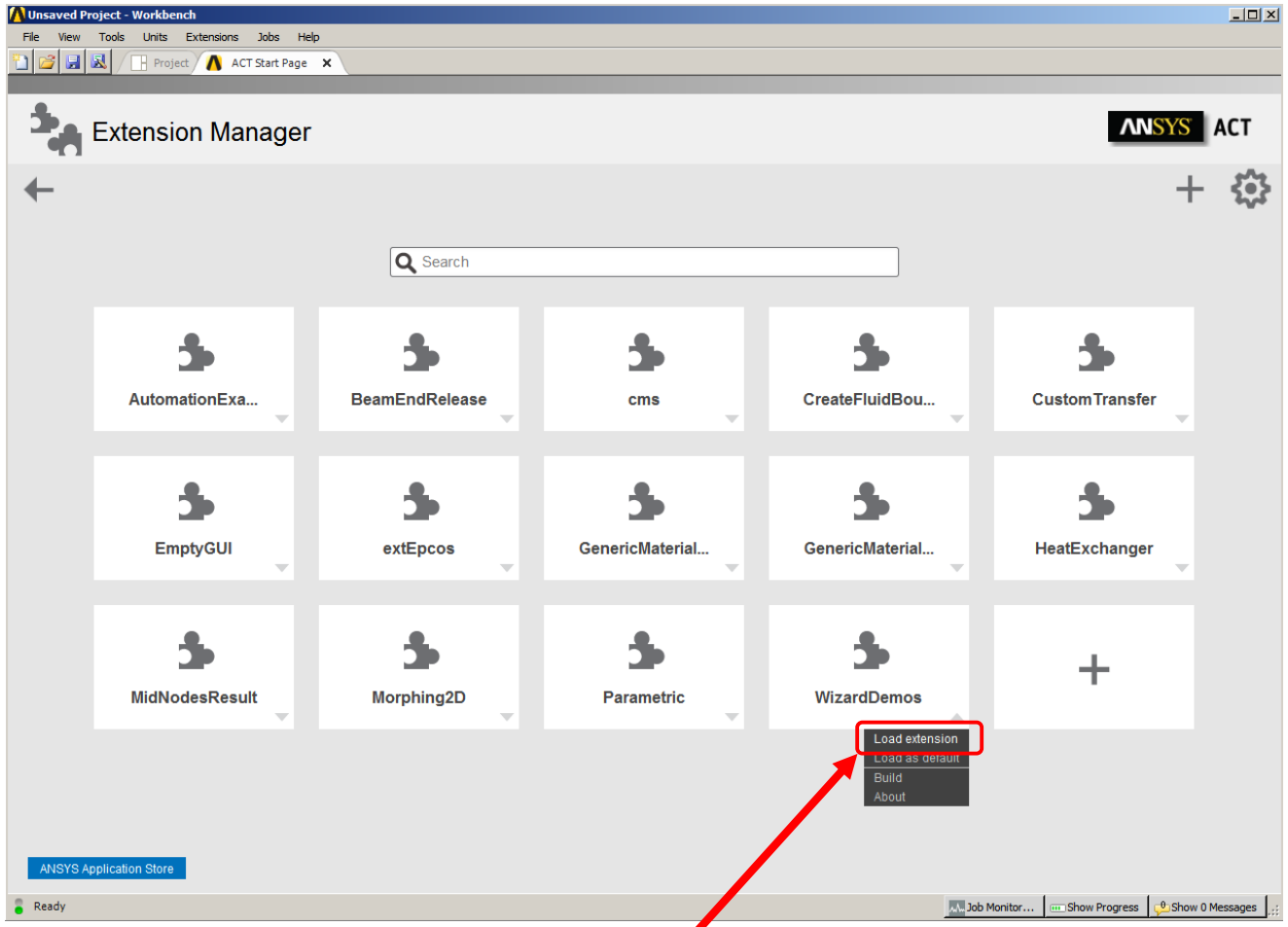
1 – Click the settings button to add new folders if needed

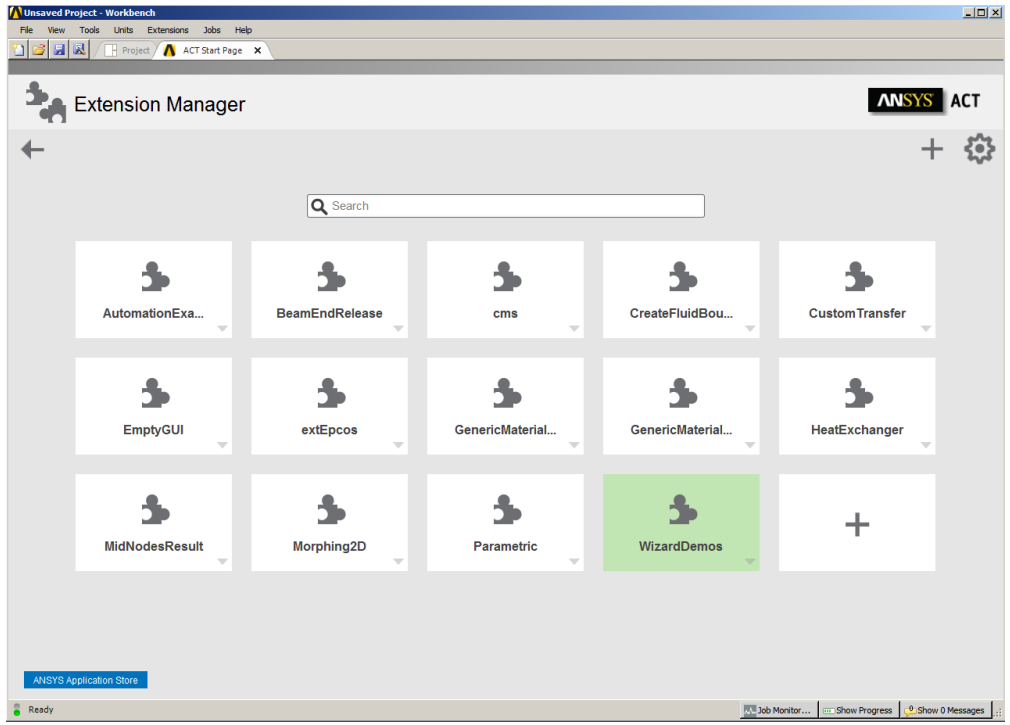


2 – Click the "Add Folder" button to insert a new folder

Extension Manager launched from the ACT Start Page

Get access to wizards (3)

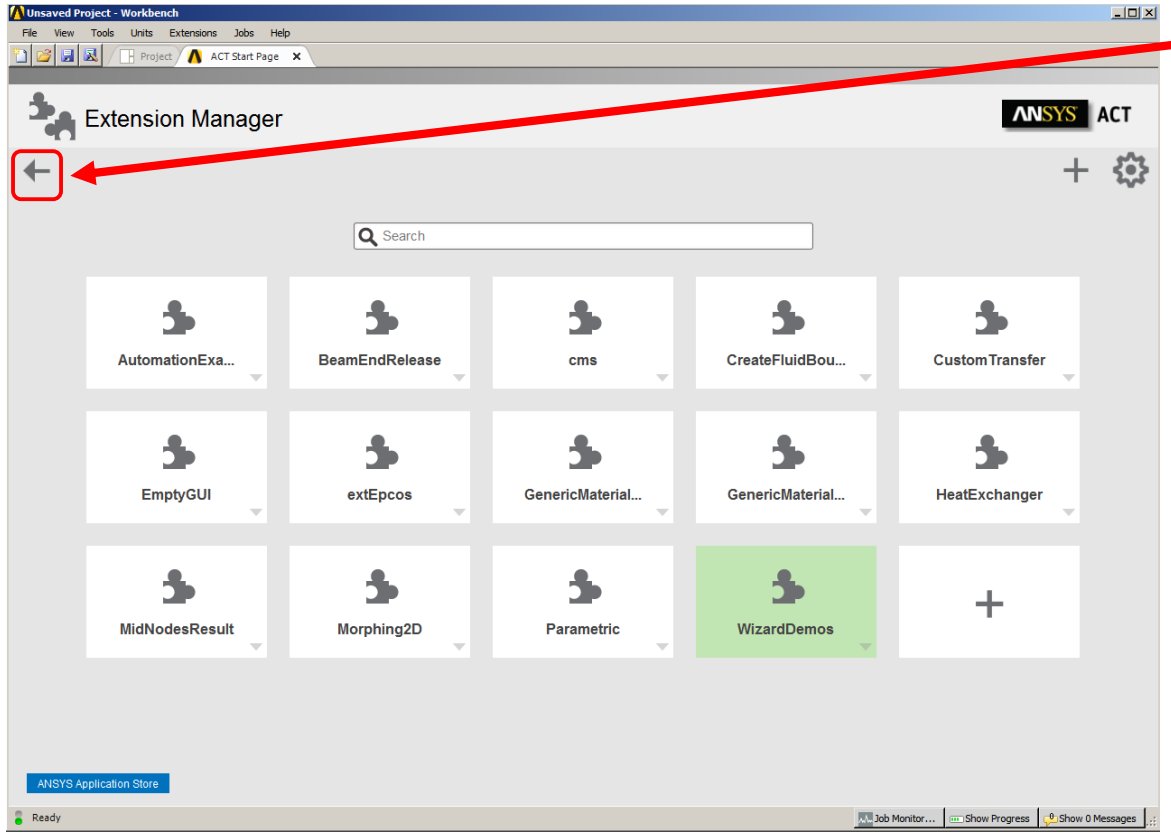




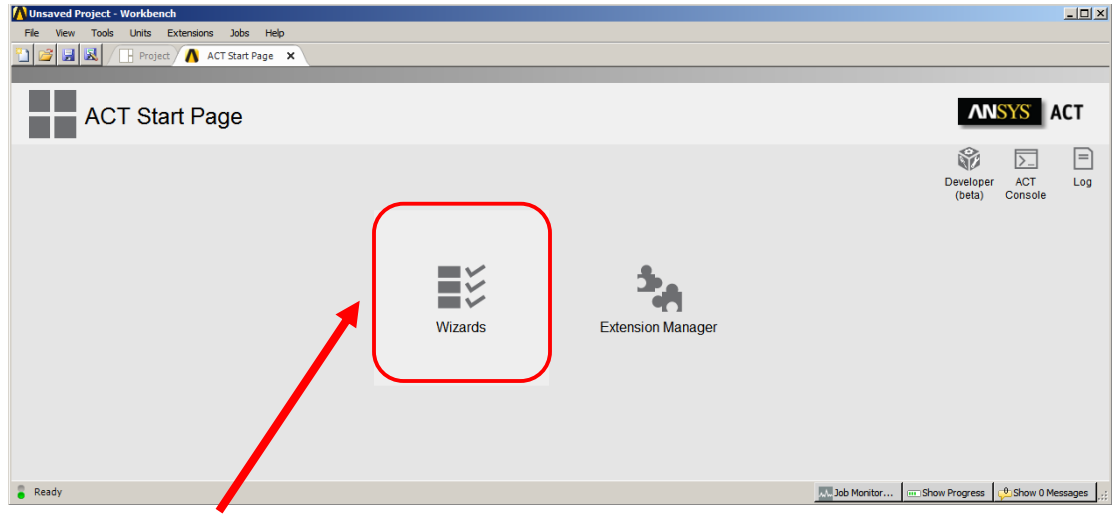
Click the panel of the extension directly or select the “Load extension” option to load the extension

Once loaded, extensions are marked in green

Get access to wizards (4)

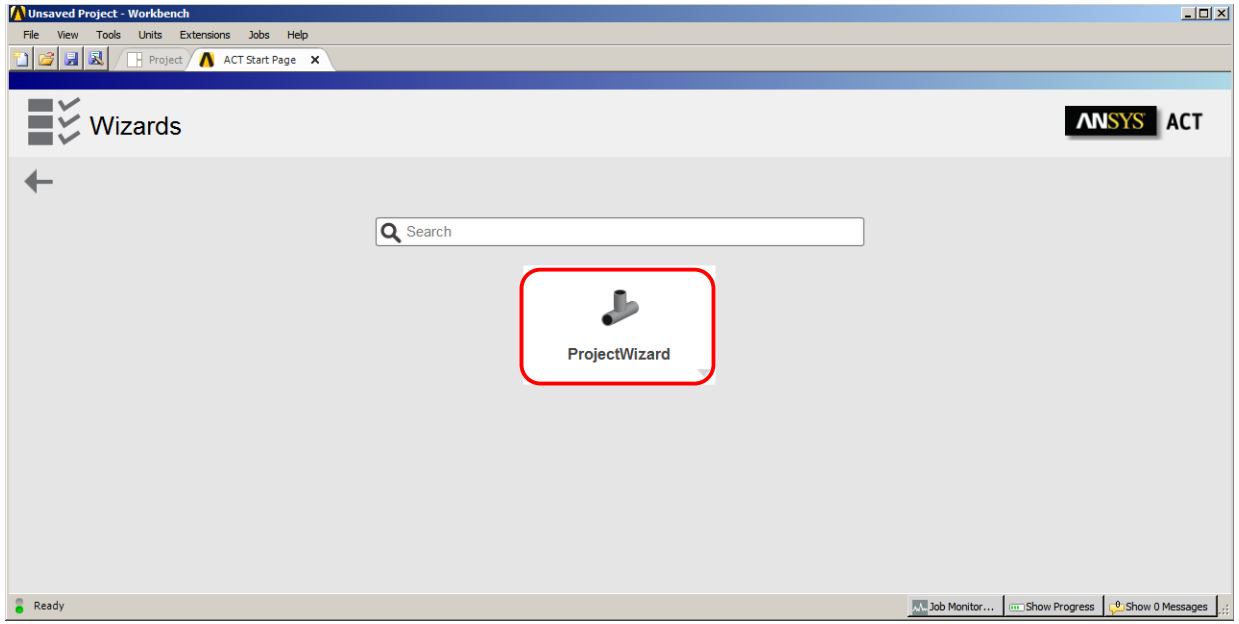


Return to the ACT Start Page

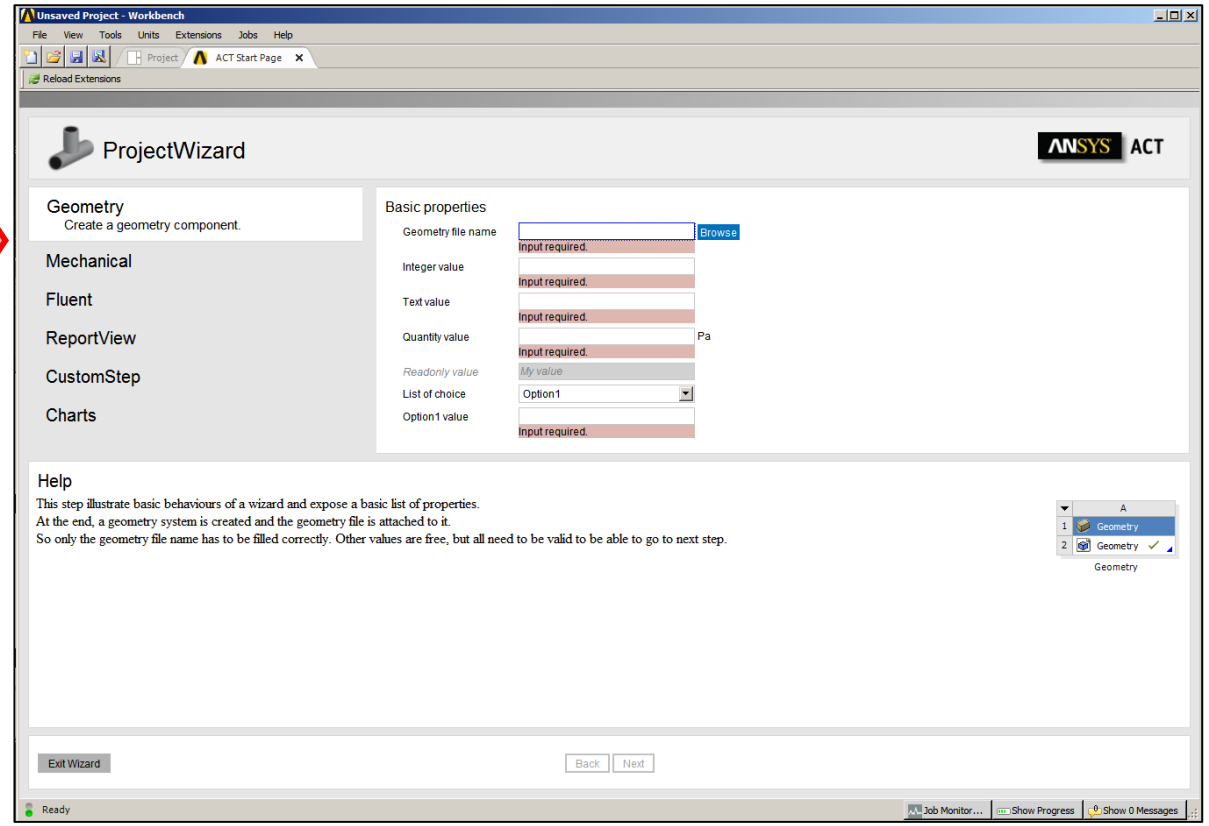


Click the Wizards icon to start the Wizards manager

Get access to wizards (5)



The Wizards manager displays all wizards that are currently loaded. You use the Extension Manager to install and load (and unload) wizards.

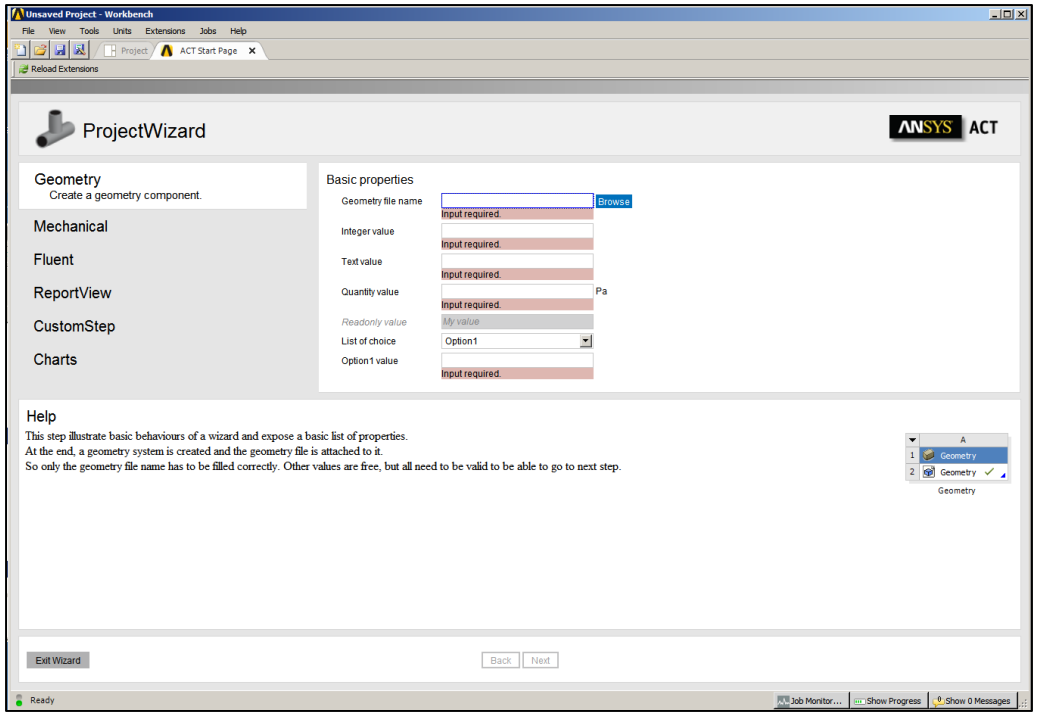


Congratulations! Your wizard is launched.

Wizards Implementation

- **By nature, a wizard is an extension following the same rules:**
  - **XML file with definitions for steps, properties, and callbacks**
  - **IronPython functions to describe the process to be launched for each callback**
- **Layout is generated by default but it can be customized using additional tags in the XML file**
- **One unique implementation whatever the application where the wizard resides**
- **Tags for layout definition (optional):**

```
<extension>
    <uidefinition>
        <layout>
    </uidefinition>
</extension>
```



Wizards Implementation

- **Two mandatory elements for wizard definition: <wizard> and <step>**

```
<wizard>
    <step>
        <callbacks> ... </callbacks>
        <componentdata>
        <componentstyle>
        <description>
        <property>
        <propertygroup>
        <propertytable>
    </step>
</wizard>
```

- **Only a few callbacks are needed:**
  - **One mandatory callback <onupdate> to go to the next step**
  - **Two optional callbacks:**
    - **<onreset> to go back to the previous step**
    - **<onrefresh> to insert a custom treatment at the initialization of a step**

ACT Approach and Extensions

Typical Use Case

A “super” user (or a tools/methods group) develops an ACT-based extension that encapsulates expertise, know how…

and provides it to “everyday” users for expanded usage of simulation within the organization.

- **A “super user” needs to have the ACT development module to develop extensions.**
- **The “everyday” users don’t need the ACT development module for using extensions.**

ACT Components

ACT Module (development toolkit)

- Used to create ACT-based customizations or “extensions”
- License-managed (ACT license)
- Available with the Enterprise license
- Maintained and supported by ANSYS

ACT Extensions (resulting customizations)

- What becomes visible to the end user in the GUI (similar to UDF)
- Create extensions under binary format (no license)
- Outside of ANSYS’s standard support model
- Don't need a license (beyond the one to run ANSYS Mechanical) to use extensions

ACT Extension Development and Use

- Typically, a “super” user (or tools/methods group) needs to *standardize a simulation process*. This involves:
  - Need to create new features (not existing in Mechanical already)
  - Need to automate existing features and new features in Mechanical
  - Use of other Workbench capabilities such as CAD Connection, DP/DX, Meshing, and so on

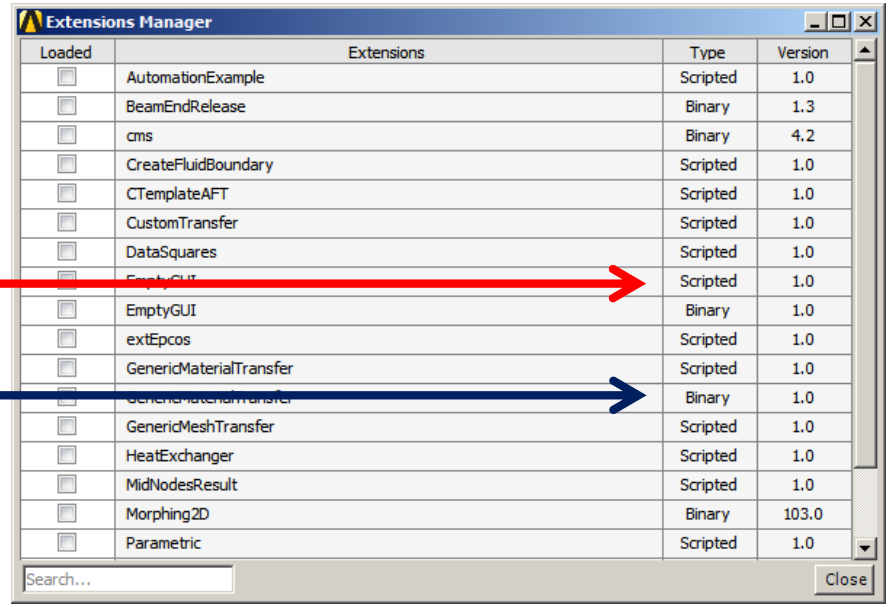
- Also, there is a need to:
  - Reduce the learning curve for new engineers (so they don’t have to be APDL experts)
  - Avoid a solution where a user:
    - Can deviate from the standard process
    - Has to change things and adapt to his or her needs
  - Be able to upgrade the solution and deploy it easily
    - Avoid multiple versions

An ACT-based customization addresses all of these requirements very well

- A “super” user develops an ACT-based extension that encapsulates expertise, know how…
- And provides it to “everyday” users for a larger usage of simulation within the organization.

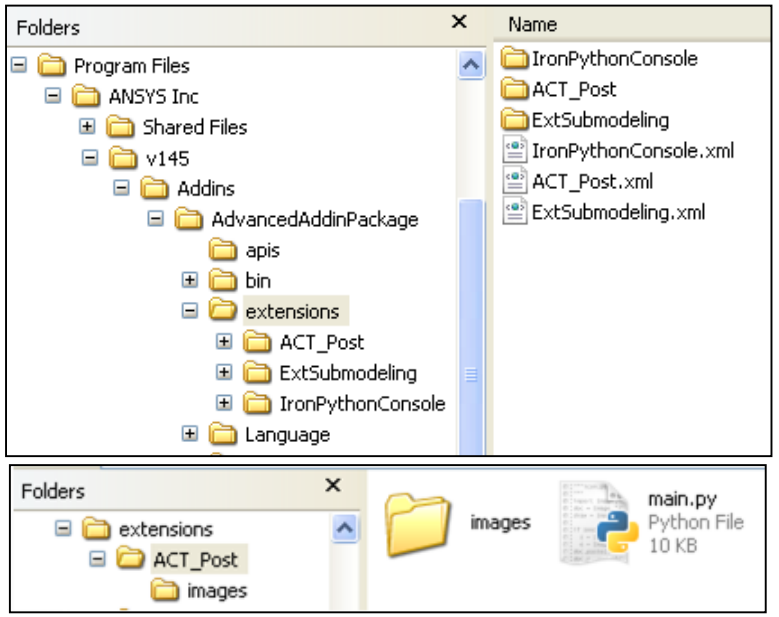
ACT Extension Types

- ACT handles two different types of extensions
  - Scripted extensions
    - Used during the development phase
    - Need ANSYS Customization Suite license to use
    - Made of XML and IronPython functions
  - Binary extensions
    - Provided by the developer to the users (once the extension is completed)
    - Doesn't need a license (beyond the one to run ANSYS Mechanical) to use
    - Result of Build Binary Extension… command for one scripted extension → WBEX file
    - This WBEX [WorkBench EXtension] file is installed from the Extensions menu



ACT Extension Basics

- A scripted ACT extension consists of at least:
  - ➢ *One XML file*
    - Configures the UI content
    - Defines the extensions properties
    - Binds application events to IronPython script functions
    - Configures behaviors for custom loads and results

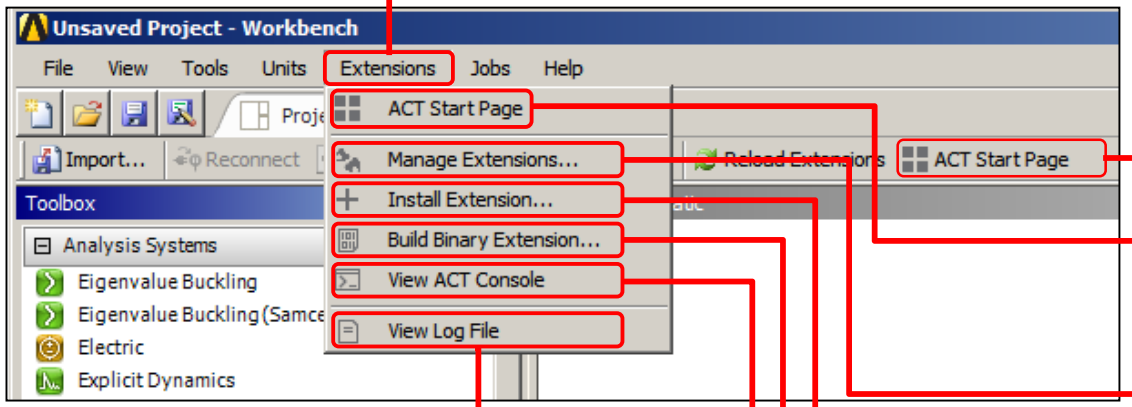


  - – ***One IronPython script file***
    - **Implements the extensions functionality**
    - **Event-driven: Functions are invoked by application-generated events**
    - **Supports access to external libraries**
    - **The script file is typically placed in a folder having the same name as the XML file**

- **You can have additional files and folders to organize the content better**
  - – **A separate folder can be used for images, other resources, and so on**
- **An extension can potentially be made of additional components**
  - – **External Python libraries, C# code, and so on**

A binary ACT extension is a WBEX file that you need to install through Workbench

Extensions Menu



The *Extensions* option is available in the menu bar of the Project page

Launch the ACT Start Page

Open the Extension Manager to load and unload extensions

Install extensions

Build binary extensions

Launch the ACT Console

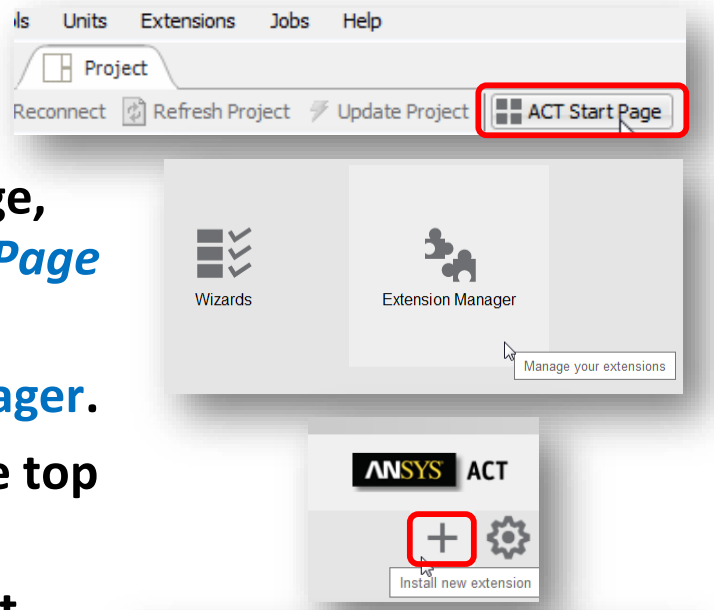
View log file to review messages generated from extensions

⚠

- No additional installation is required for the ACT module.
- The ACT module is installed along with Workbench

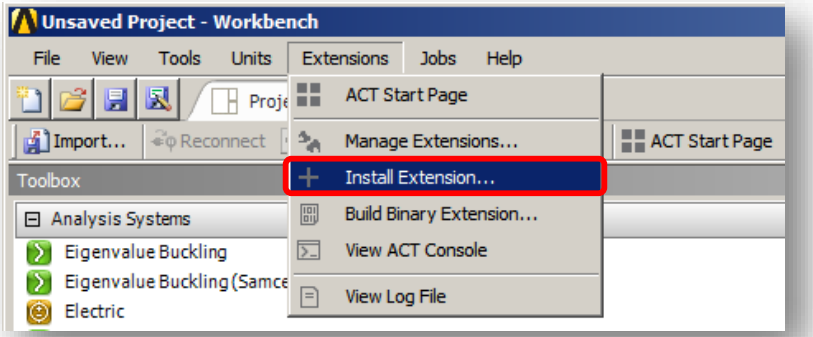
Install binary extensions

Installing from the ACT Start Page:

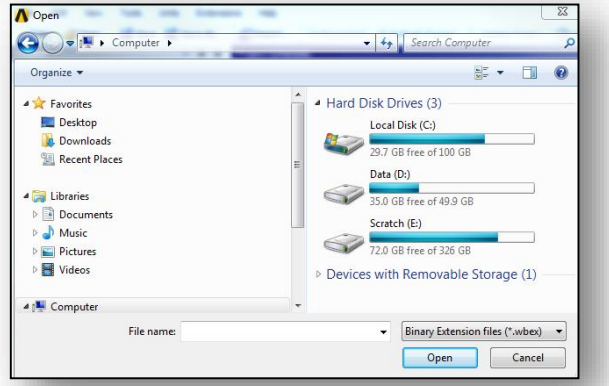


1. From the Project page, select the *ACT Start Page* option.
2. Click Extension Manager.
3. Press + symbol in the top right corner.
4. In the dialog box that opens, select the appropriate **.wbex* binary file.
5. Click Open to install the extension.

Installing from the Extensions menu:



1. From the *Extensions* menu, select the *Install Extension…* option.
2. In the dialog box that opens, select the appropriate **.wbex* binary file.
3. Click Open to install the extension.

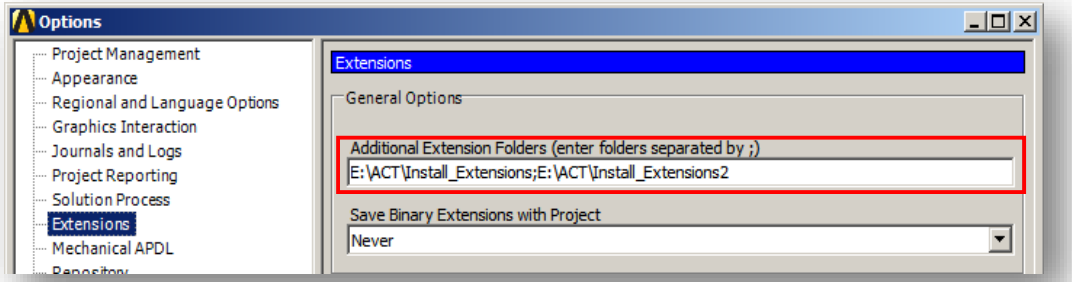


Install scripted extensions

- Paste the XML file and the corresponding folder on your computer. You can paste them in a user-defined path: Any location on your machine or a shared drive.

- If the files are located in the default path, the extension is automatically available in the Extension Manager.

- If the files are in a user-defined path, you must define the *Additional Extension Folder…* property under Workbench menu (Tools → Options…) to make it available in the Extension Manager.

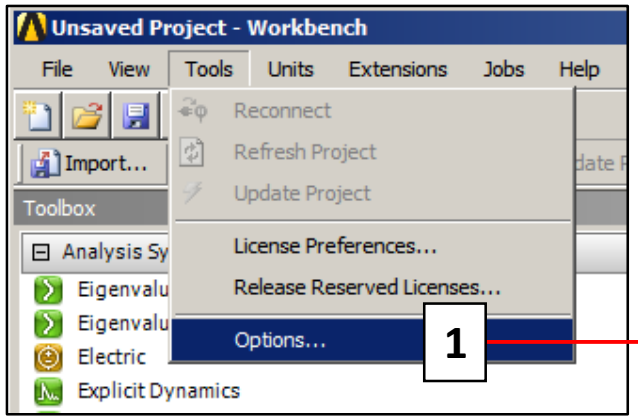


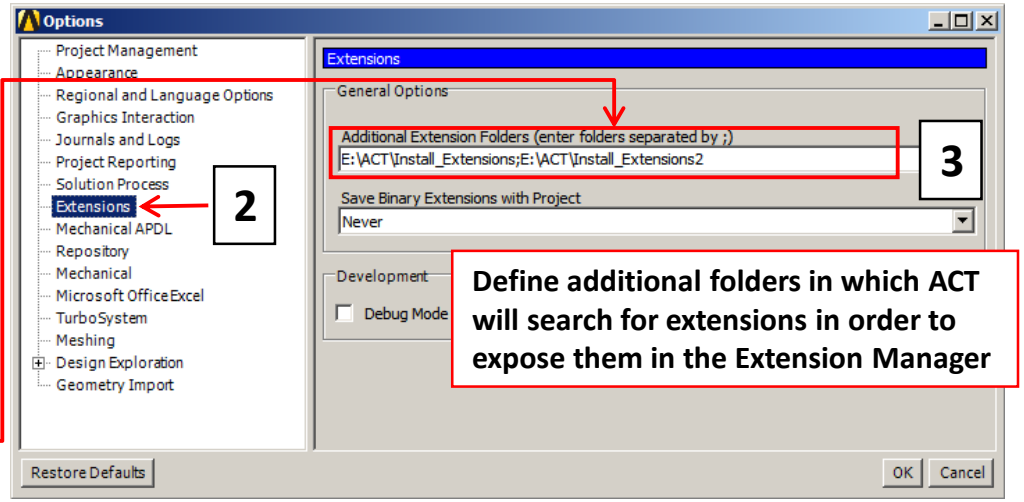
Configure the installation of an extension

- Once the binary extension is installed at the default location, you can move the *.wbex and the folder to any other location
  - Default path: *%AppData%\Ansys\[version]\ACT\extensions*
  - New path: Any location on your machine or a shared drive.

- All users who want to use the extension must include that path in their Workbench Options

1. In the *Tools* menu, select the *Options…* command.
2. Select *Extensions* in the pop-up panel.
3. Add the path in the *Additional Extensions Folder …* property.





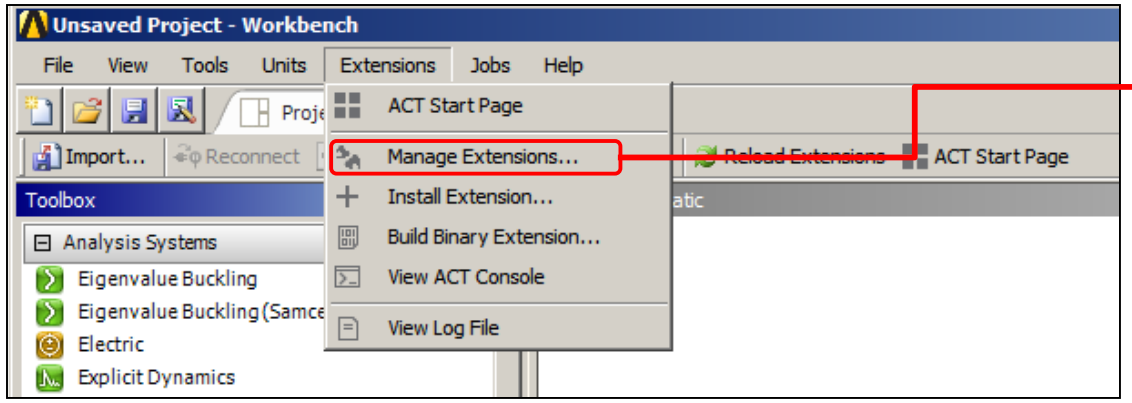
Notes:

• During the scan of the available extensions, the folders will be analyzed in the following order:

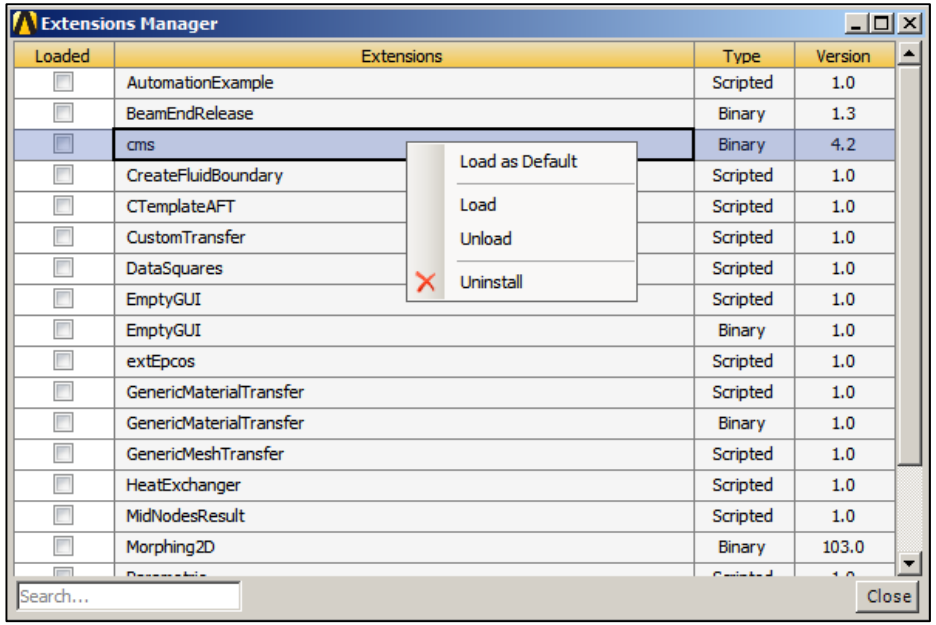
1. Application data folder (*%AppData%\Ansys\[version]\ACT\extensions)*
2. Additional folders defined in the "*Additional Extension Folders* property.
3. Installation folder
4. The "extensions" folder part of the current Workbench project (if the project was previously saved with the extension)

• If an extension is available in more than one of these locations, the first one according to the above order is used.

Manage Extensions Menu (Legacy Extensions Manager)



Open the Extension Manager to load and unload extensions

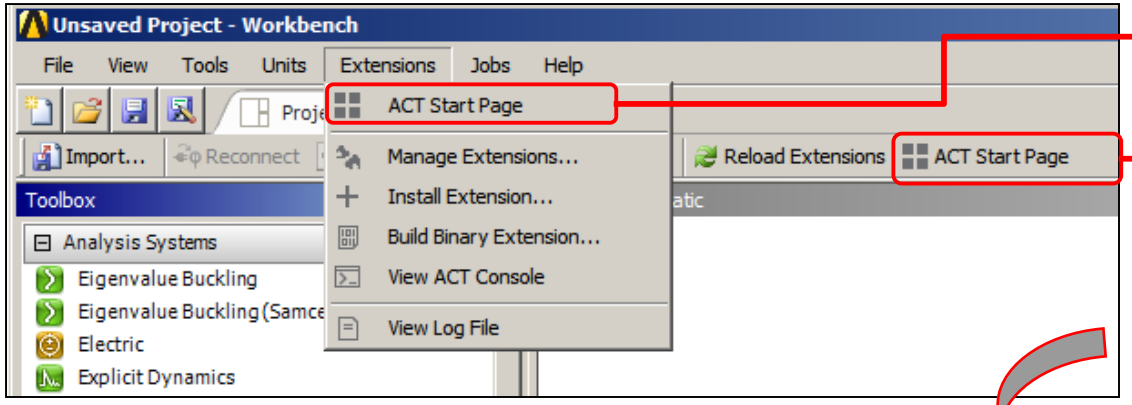


Launch the Extensions Manager (legacy version)

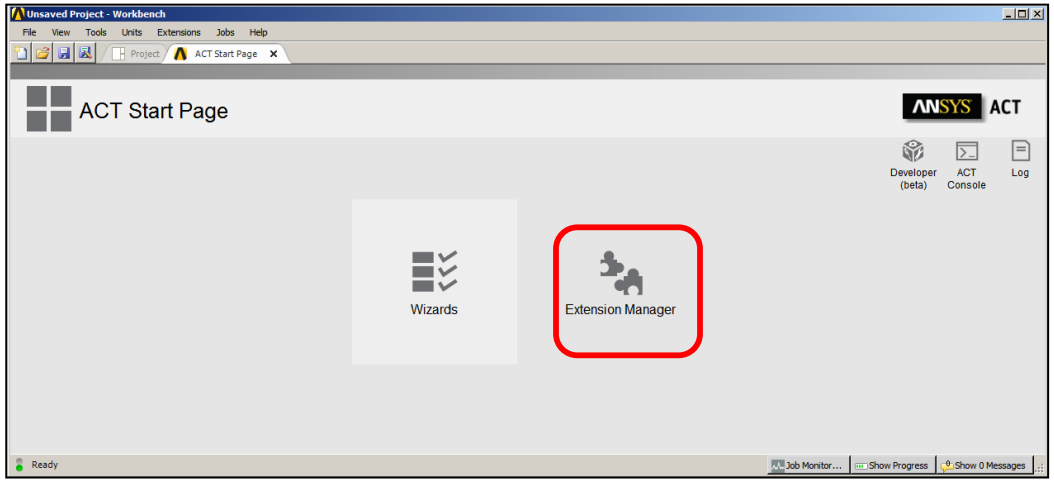
- Extensions are loaded with an application if they are preselected in the Workbench Extension Manager
- An extension can be loaded and unloaded by selecting and clearing the check box
  - Extensions that were used in a specific project are automatically selected on opening the project
- Additional options are available by right-clicking an extension
  - To load certain extension as default (such as a new project)
  - To uninstall a binary extension

Manage Extensions Using Extension Manager from ACT Start Page

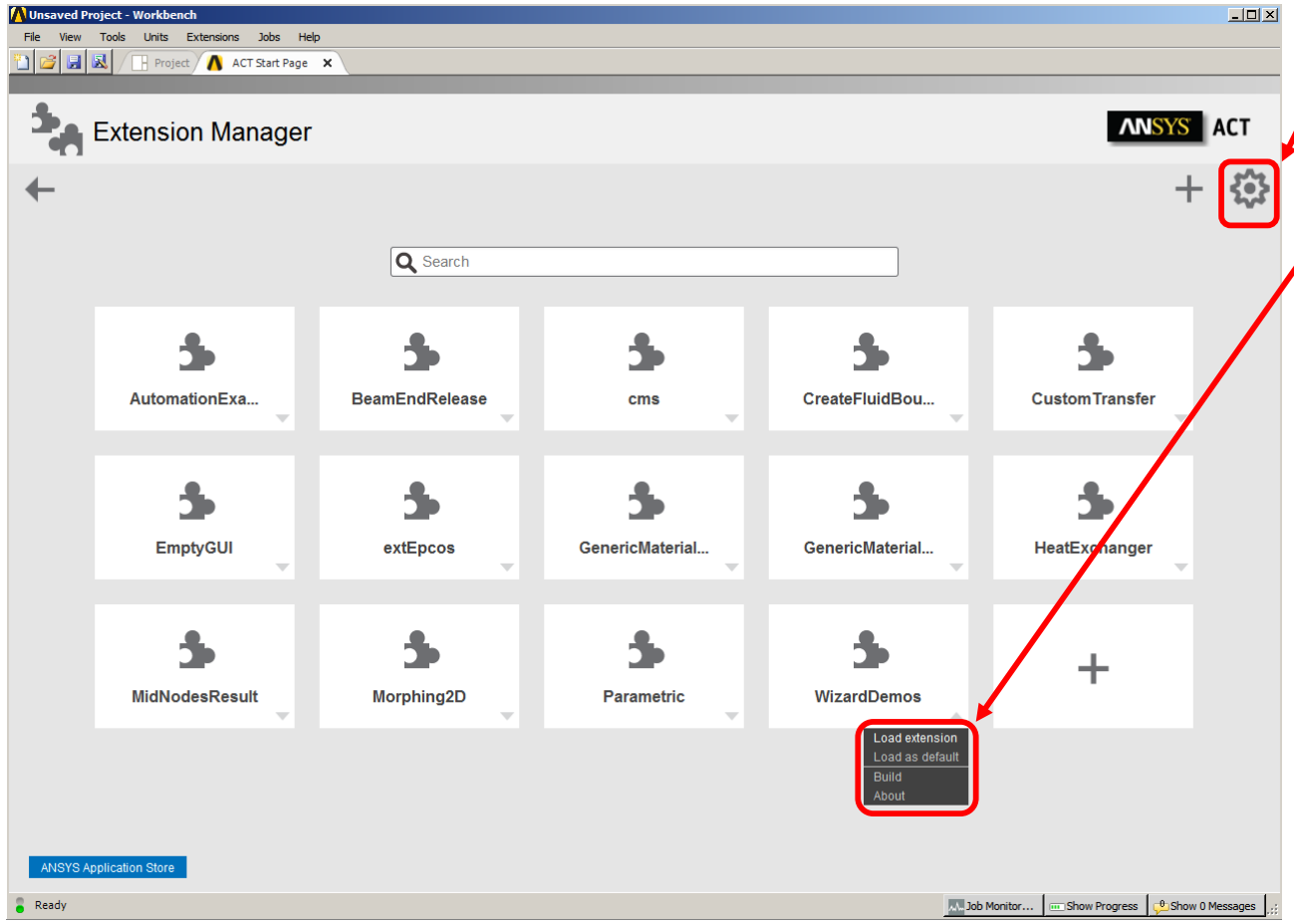
- **Newer version of the Extension Manager is available from the ACT Start Page**
- **Features remain the same as the legacy Extensions Manager**
- **The ACT Start Page, with Extension Manager access, is made available in all products, including standalone products**



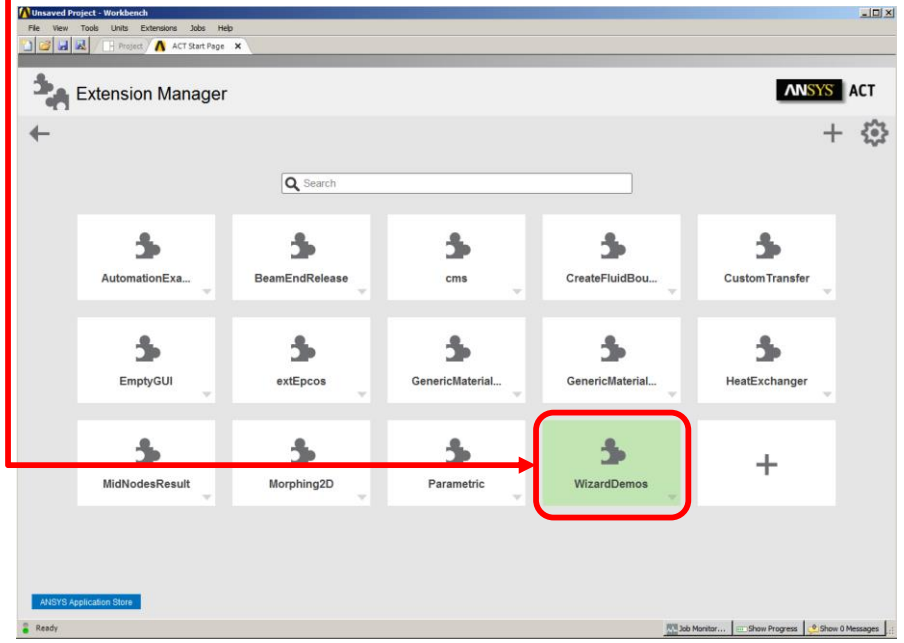
Launch the ACT Start Page from the *Extensions* menu or *ACT Start Page* button



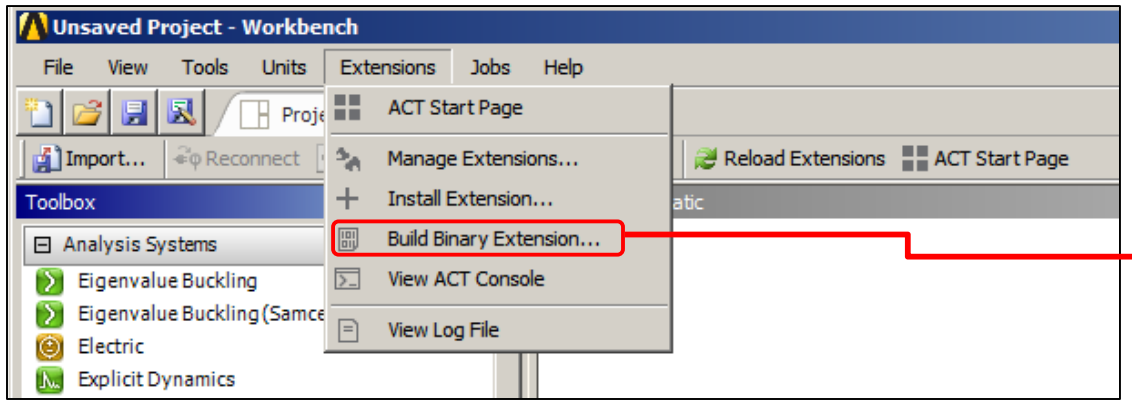
Extension Manager from ACT Start Page



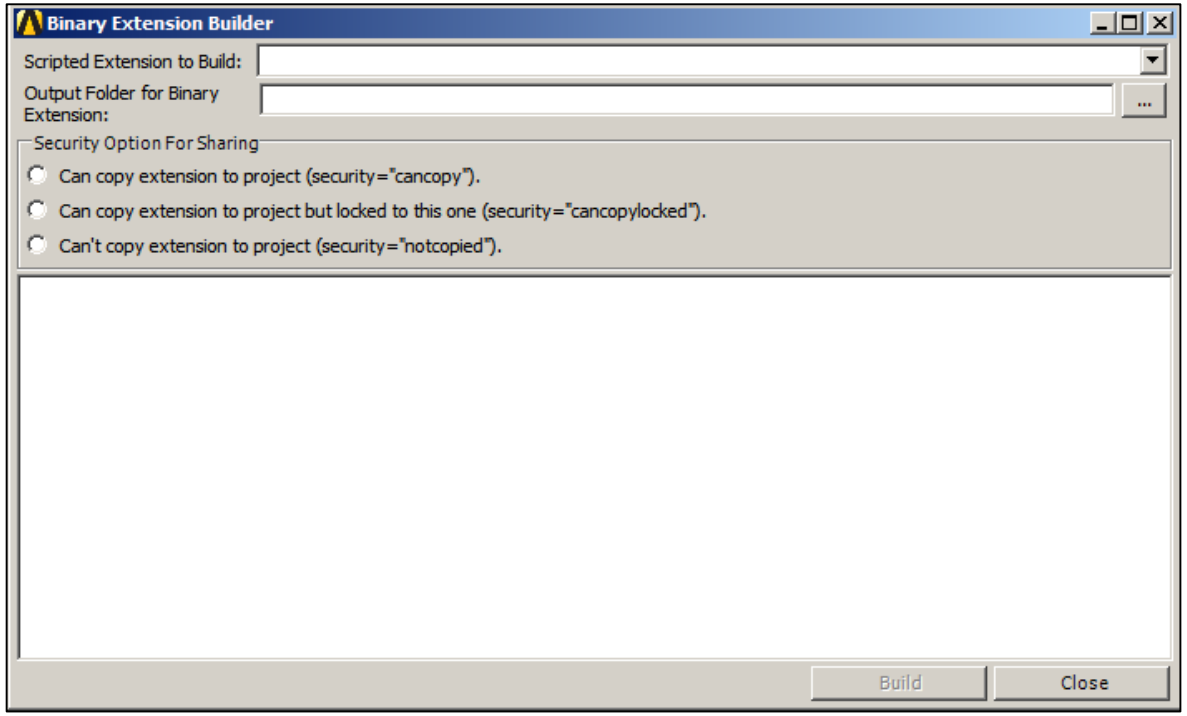
- Click the settings (gear) symbol to define the new folders in which to find extensions
- Load and unload extensions by clicking the extension panel directly or by using the drop-down menu
- Loaded extensions appear in green



Build Binary Extension

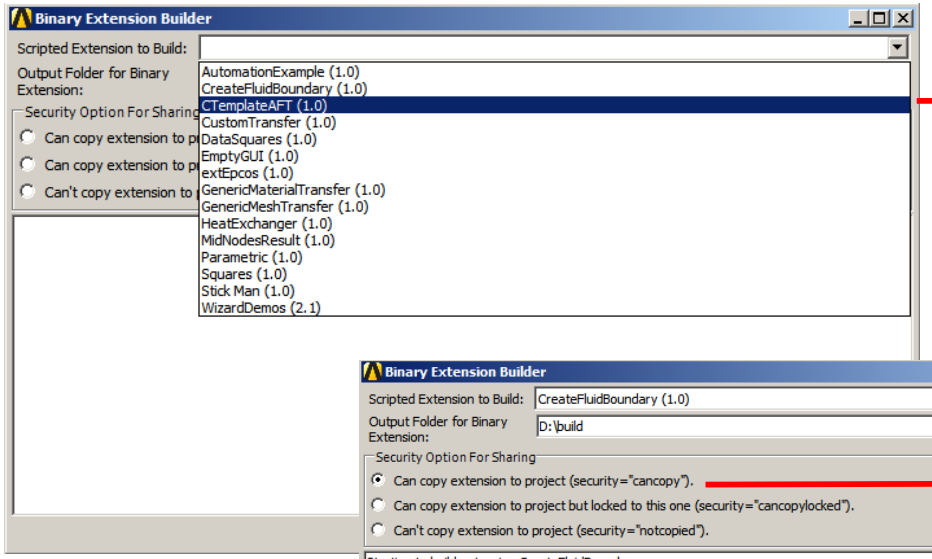


Build binary extensions

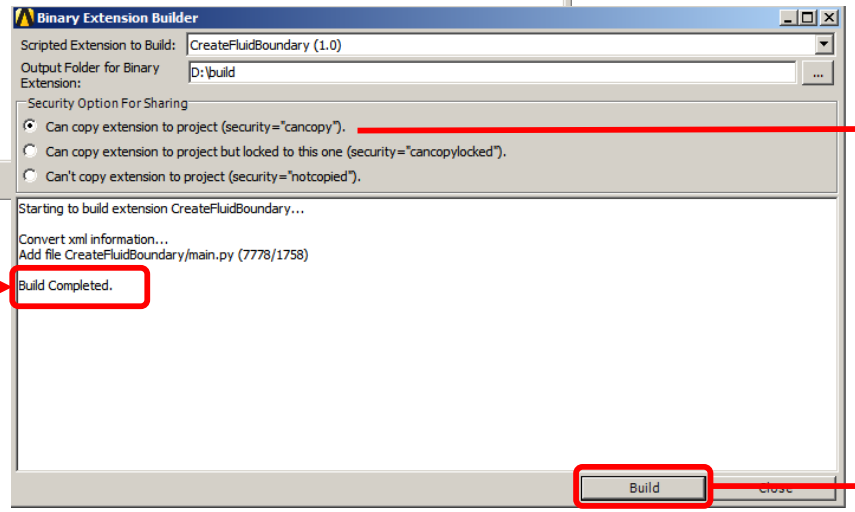


Launch the Binary Extension Builder to migrate an extension from the scripted to binary format

Build Binary Extension (2)



1. Select the scripted extension to be built. All scripted extensions currently found by the Extension Manager are listed.



2. Select the security option for controlling how the binary extension can be used later.

3. Launch the build action to generate the binary extension.

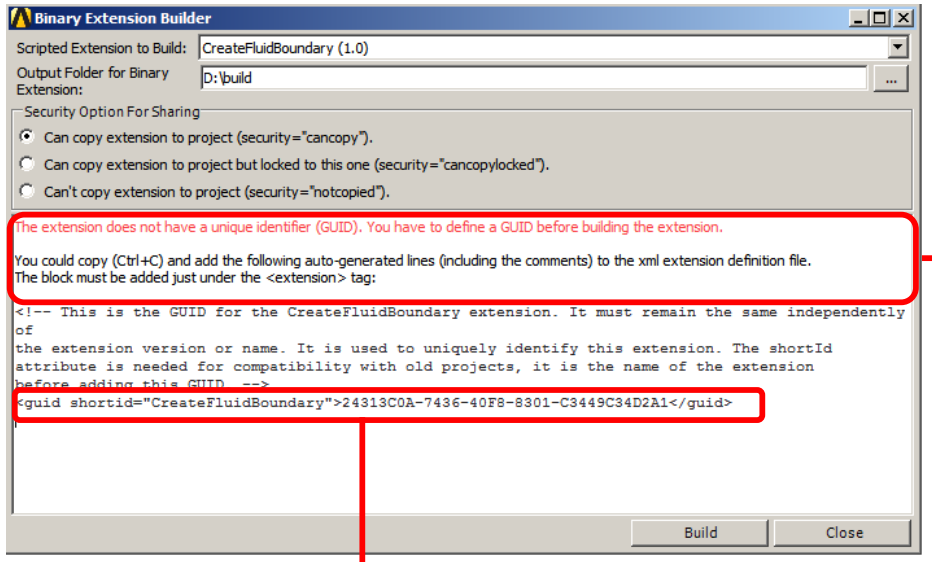
The build is complete. The binary extension has been generated successfully.

Build Binary Extension (3)

- Three security levels are available:
  - *Can copy extension to project*. Each time a user asks to save this extension with a project, the extension itself is copied into the project and consequently is available each time this project is opened. This extension can also be used with other projects.
  - *Can copy extension to project but locked to this one*. The extension can be saved within a project, as with the previous option, but the use of the extension is limited to the current project. If a user opens a new project, this extension is not available.
  - *Can't copy extension to project*. The extension is not saved with the project and is not shared with other users with the project. Note that the extension can be sent in addition to the project.

- These security options apply only if the user wants to save the extension with the project. Otherwise, they are not applicable.

Build Binary Extension (4)



Warning or error messages can occur during the build. Check the log carefully.

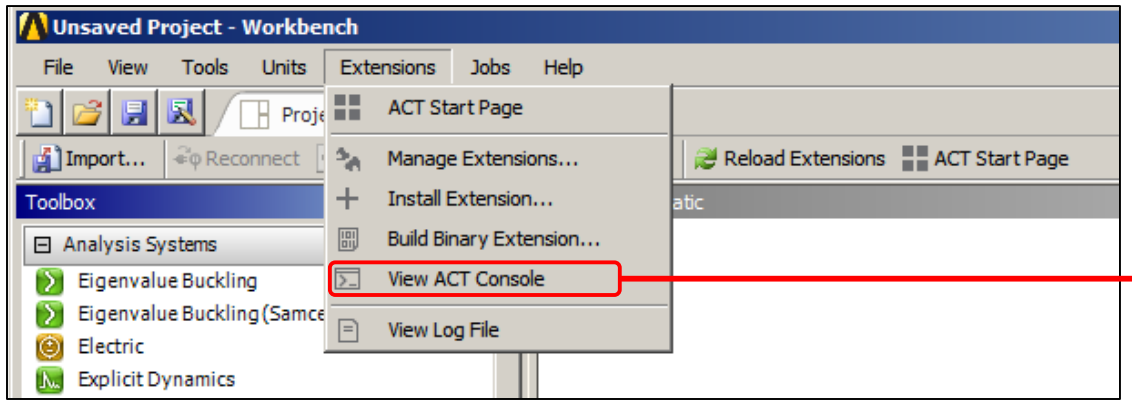
If the "guid" is not defined in the XML file, a message warns the developer to insert it as indicated in the instructions given.

```
<extension version="1" name="CreateFluidBoundary">
    <guid shortid="CreateFluidBoundary">24313C0A-7436-40F8-8301-C3449C34D2A1</guid>
    <script src="main.py" />
    <wizard name="CreateFluidBoundary" version="1" context="Study">
        <description>Simply extension to create fluid BC</description>
        <step name="Step1" caption="Define Process Inputs" version="1" context="Project">
            <callbacks>
                <onupdate>action</onupdate>
            </callbacks>
            <property name="Velocity" caption="Velocity" control="float" unit="Velocity" default="1 [m/s]" />
        </step>
    </wizard>
</extension>
```

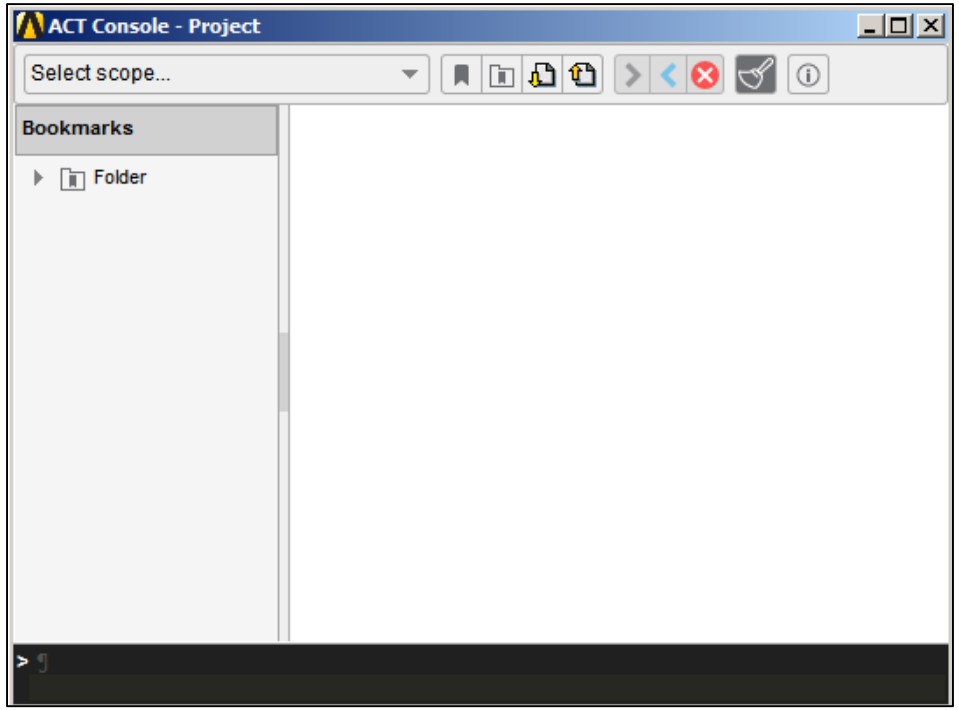
Copy and paste the "guid" definition as proposed in the log window in the XML file to avoid the warning message and launch the build operation again.

As result, the *extension_name*.wbex file is created.

View ACT Console

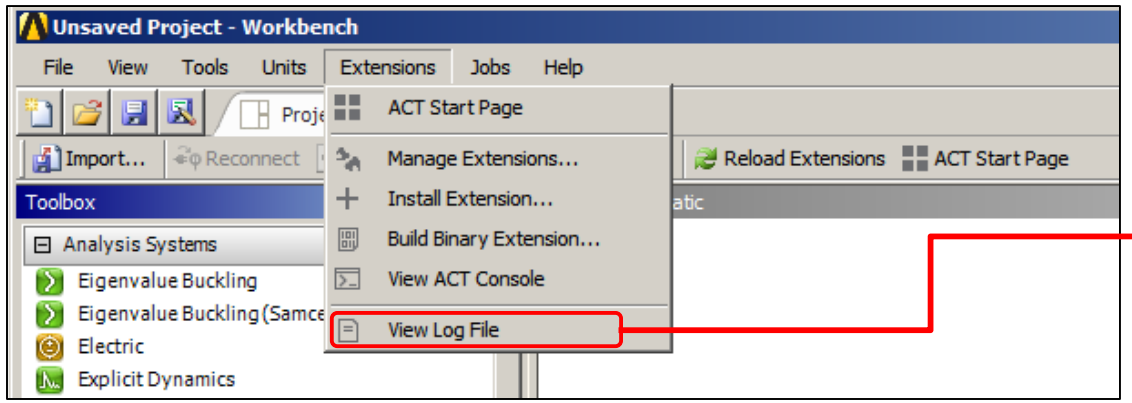


Launch the ACT Console

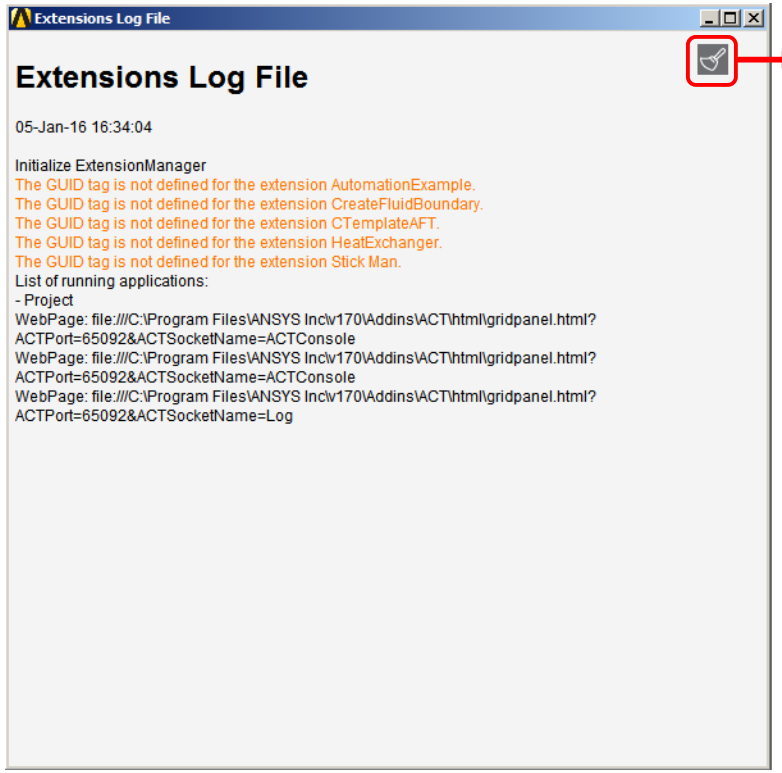


The ACT Console provides access to the APIs

View Log File



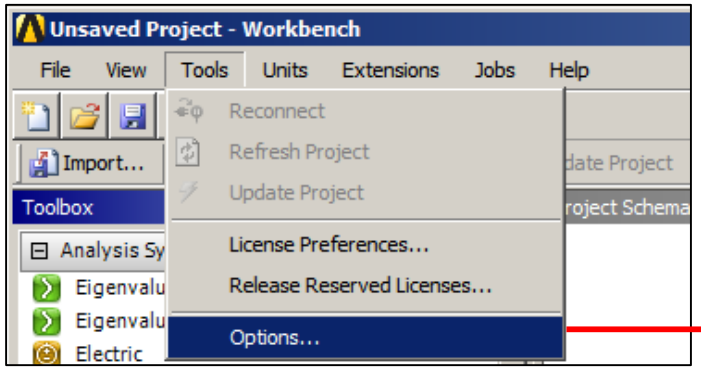
View log file to review messages generated from the extensions



Click here to reset the content of the window

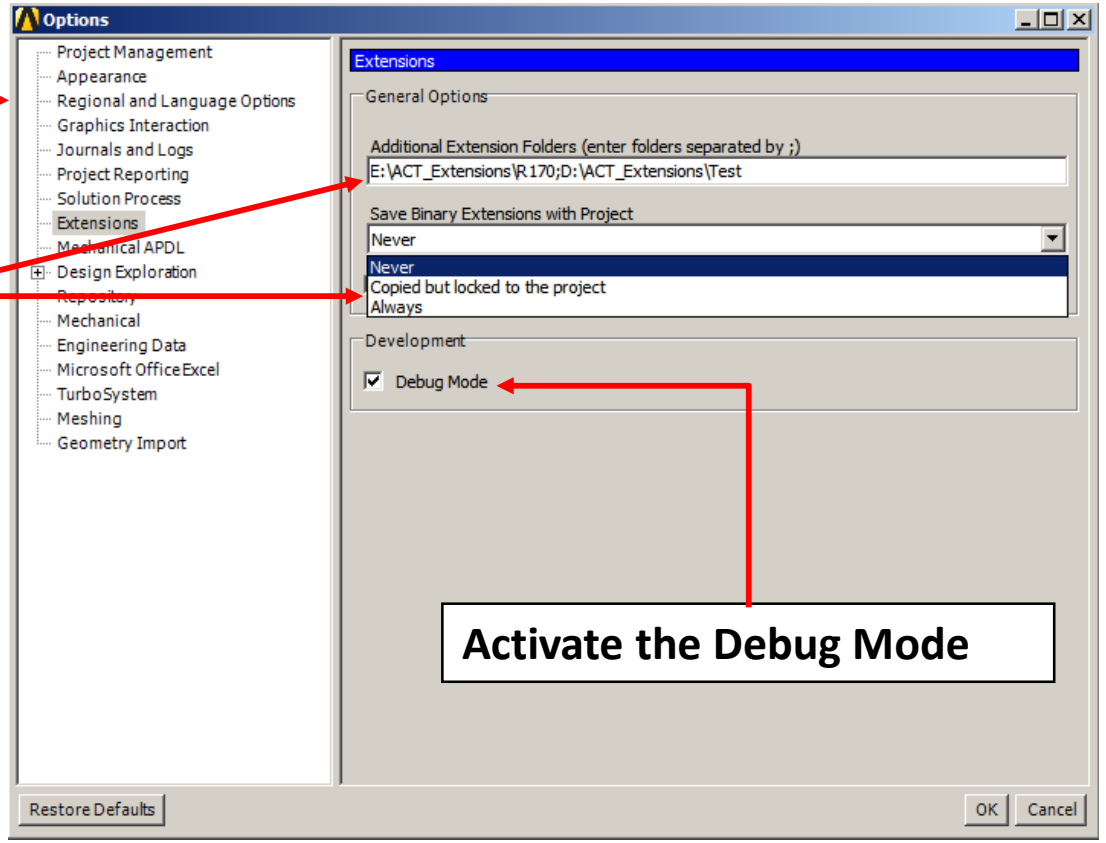
The Extensions Log File window displays log, warning, and errors messages

Options “Extensions”



- **Define additional folders that ACT is to search for extensions to expose them in the Extension Manager**
- **Folder names have to be separated by semicolons**

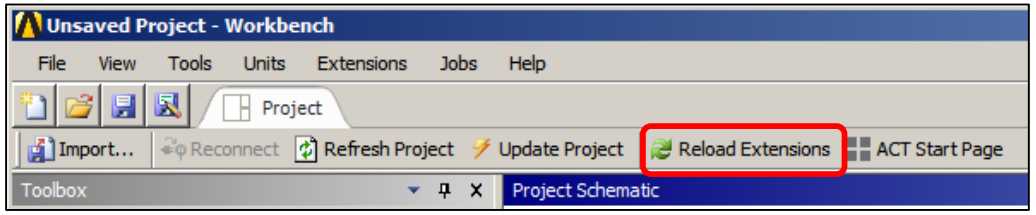
- **Define if the loaded extensions need to be saved within the project**
- **Locked option available if needed**



Debug Mode Option

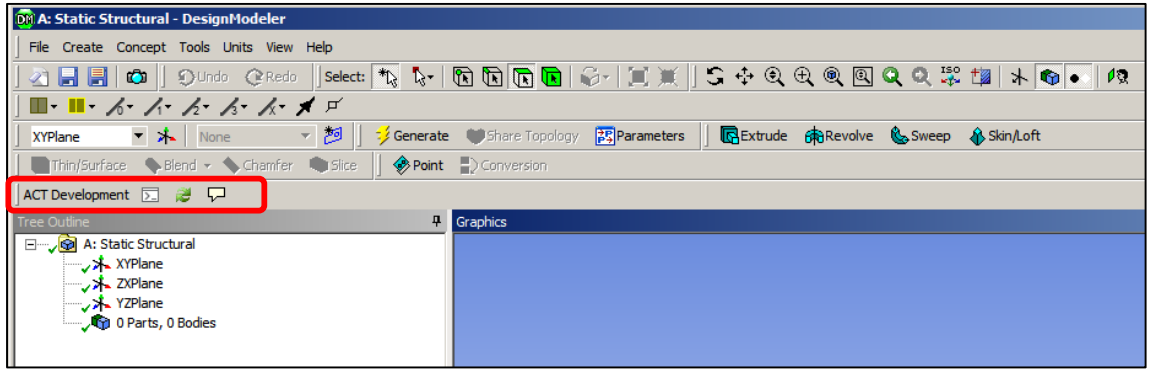
Once the Debug Mode is active, you get two additional features to help you debug your extension

- From the Project page:



Reload Extensions feature is available

- **In DesignModeler:**



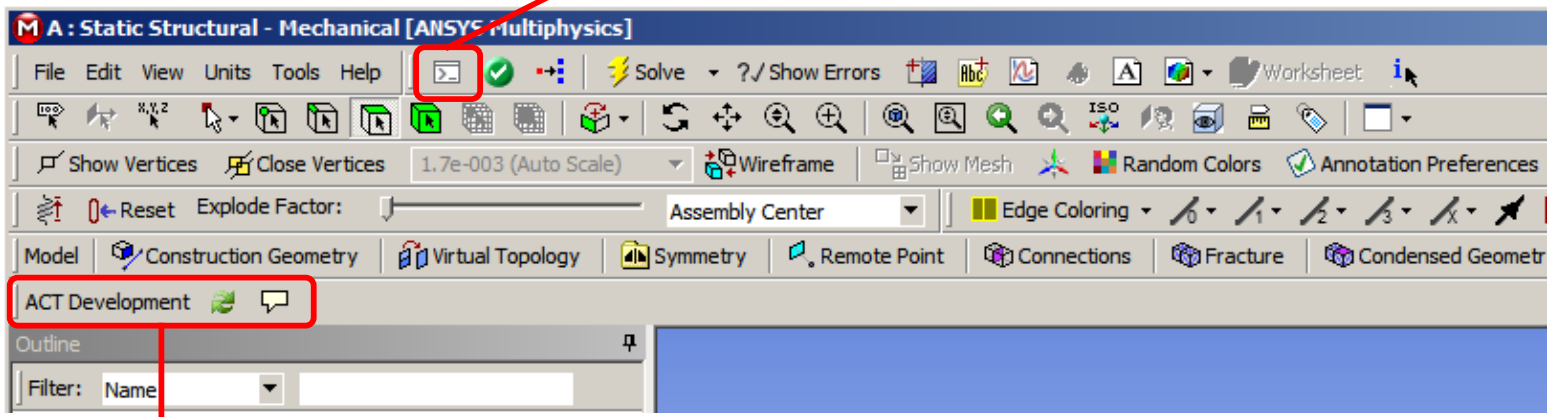
ACT Development toolbar is available in DesignModeler. This toolbar includes:

- **ACT Console**
- **Reload Extensions feature**
- **Access to the Log file window**

Debug Mode Option (2)

- **In Mechanical:**

ACT Console is always available, regardless of Debug Mode



ACT Development toolbar in Mechanical. This toolbar includes:

- **Reload Extension feature**
- **Access to Log file window**

ANSYS®

Q/A