MBSE CYBER SYSTEMS SYMPOSIUM 2025

EXTENDING THE CAPABILITIES OF CATIA MAGIC

DASSAULT SYSTEMES

MBSE CYBER SYSTEMS SYMPOSIUM 2025

Tim Anderson
Co-founder/COO
Enola Technologies

DASSAULT SYSTEMES

ENOLA TECHNOLOGIES

Outline

- Promises of MBSE
- What Can We Make CATIA Magic Do?
- Where Can the API Be Used?
- What Is the Open API?
- Managing Extensions
- Demo of Macros

ENOLA TECHNOLOGIES

Promises of MBSE

- Model Analysis
- Full Traceability
- Improved Content
  - Validation
  - Verification
- Trade Studies
- Custom Reporting
- Automations

CATIA Magic provides all of this, but it is capable of so much more!

ENOLA TECHNOLOGIES

What Can We Make CATIA Magic Do?

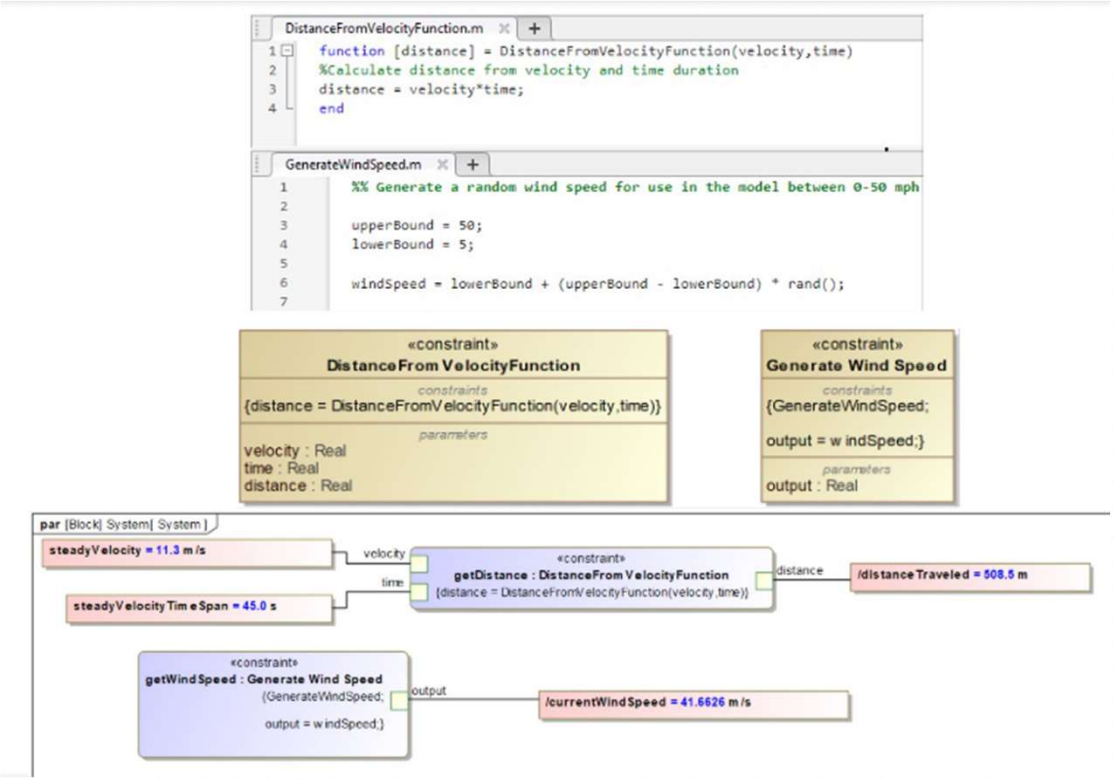
- Anything we can imagine!*
  - *and develop
- CATIA Magic utilizes a Java Application Programming Interface (API) for its main application and plugins
  - Not the same as TWC's REST API
- This API is full of software designed to help build the functionality that makes CATIA Magic so powerful
- Some of the API is Closed (subject to change) while the rest is Open (relatively stable between releases)
- CATIA Magic provides several methods for accessing and utilizing its Open API

Where Can the API Be Used?

ENOLA TECHNOLOGIES

Simulation

- Accessible in Many Forms:
  - Activity Diagrams
    - Opaque Actions
    - Guards
  - State Machine Diagrams
    - Behaviors (Entry/Do/Exit/Effect)
    - Guards
    - Change Events
  - Parametric Diagrams
    - Constraint Blocks
- Can simplify simulation models by shortcutting formal constructs with code



ENOLA TECHNOLOGIES

Structured Expressions

- Accessible in Many Forms:
  - Custom Columns / Derived Properties
  - Dynamic Legends
  - Validation Rules
- Script is selectable as an operation in the Structured Expression window
- Parameters can be added to the operation at will and referenced directly in the script
- Scripts are often used to more easily collect data for queries and to perform mathematical operations (e.g. calculating percentages)

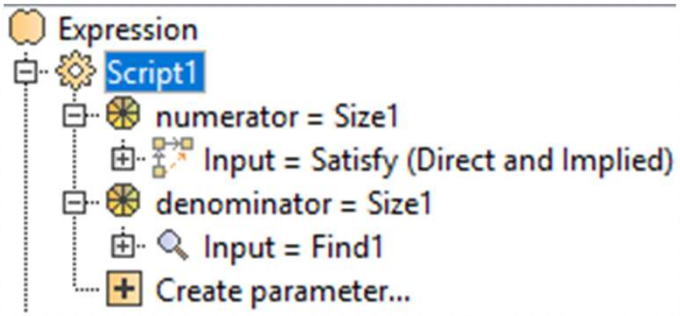
Language:

Groovy

Body:

Script1(numerator, denominator)

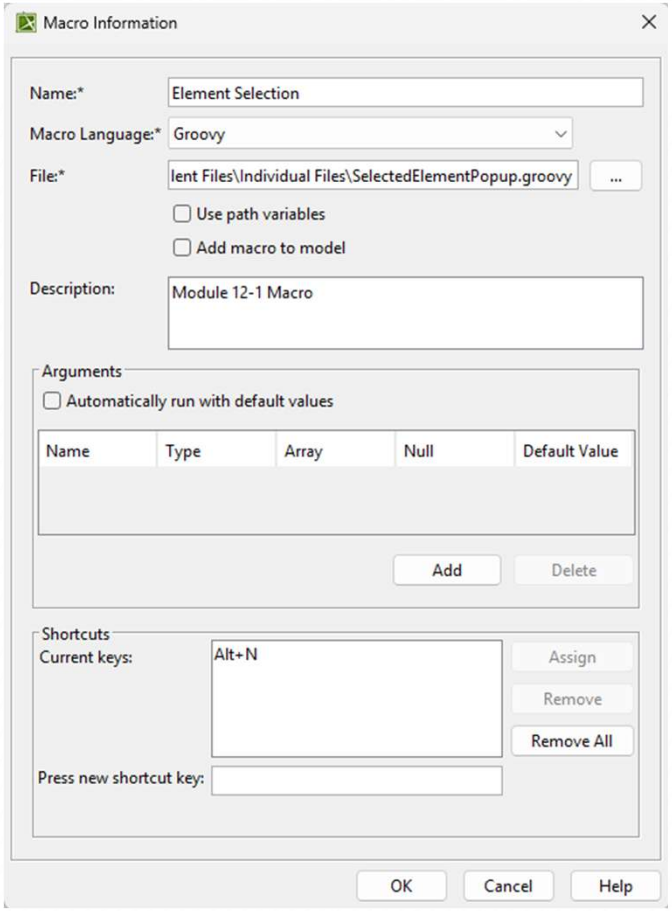
```
1  output = "";
2  if(denominator == 0)
3  {
4      output = "0%";
5  }
6  else
7  {
8      ratio = (numerator/denominator)*100;
9      output = "" + ratio + "%";
10 }
11 return output;
```



ENOLA TECHNOLOGIES

Automations and Feature Extensions

- Plugins and macros provide additional functionality to the already feature-heavy CATIA Magic application
- Both utilize the API and much of what can be done could be written as a plugin OR macro
- Macros are generally used for automating tasks and scalably transforming model content
- Plugins are generally used to add new features, often in the menu bar, like wizards and or event-listeners



Plugins vs Macros

Plugins

- Pros:
  - Can Add Custom Menu Options
  - Is Multi-threaded
  - Can Listen For, Catch, and React to Events
  - Can be Bundled into an Installation Package for a Team
- Cons:
  - Must Be Compiled in Java
  - Must Be Installed on each User's Cameo Application

Macros

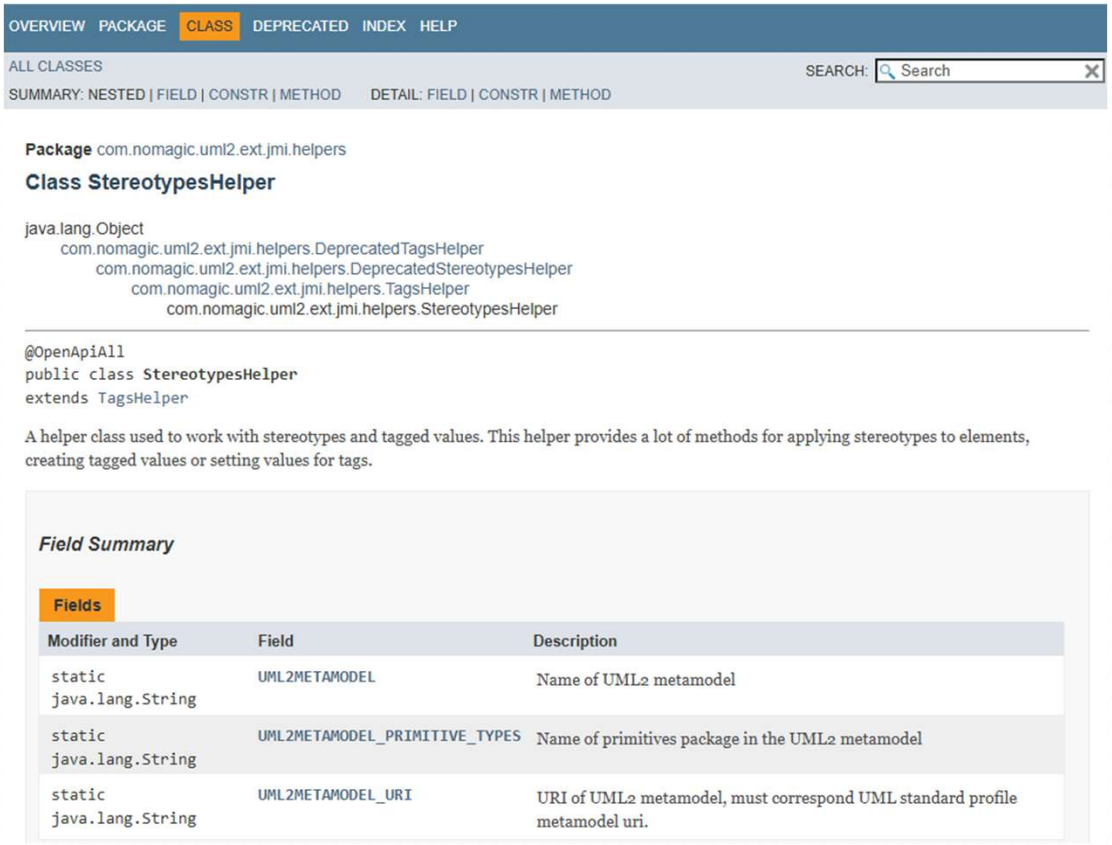
- Pros:
  - Multiple Languages Supported
  - Easy to Execute
    - Can be Hot-keyed
  - No Installation Needed
  - Can Be Added to Projects and Managed in TWC
- Cons:
  - Is Single-threaded
  - Cannot Catch Events

What is the Open API?

ENOLA TECHNOLOGIES

Java API

- An API is a way for two computer applications to communicate with each other via software code
  - It's a type of software interface that offers "services" to other pieces of software
  - An application's API consist of a set of classes that are made available to developers to use in their applications
    - APIs don't include ALL classes for application (Open vs Closed API), just the ones made available for use for interfacing with the application
    - API's hide the internal details of how a system or application works, but expose parts that may be useful for developers



OVERVIEW PACKAGE CLASS DEPRECATED INDEX HELP

ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package com.nomagic.uml2.ext.jmi.helpers

Class StereotypesHelper

java.lang.Object
  com.nomagic.uml2.ext.jmi.helpers.DeprecatedTagsHelper
    com.nomagic.uml2.ext.jmi.helpers.DeprecatedStereotypesHelper
      com.nomagic.uml2.ext.jmi.helpers.TagsHelper
        com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper

```
@OpenApiAll
public class StereotypesHelper
extends TagsHelper
```

A helper class used to work with stereotypes and tagged values. This helper provides a lot of methods for applying stereotypes to elements, creating tagged values or setting values for tags.

Field Summary

Fields

Modifier and Type	Field	Description
static java.lang.String	UML2METAMODEL	Name of UML2 metamodel
static java.lang.String	UML2METAMODEL_PRIMITIVE_TYPES	Name of primitives package in the UML2 metamodel
static java.lang.String	UML2METAMODEL_URI	URI of UML2 metamodel, must correspond UML standard profile metamodel uri.

UML Metamodel Classes

- Every class in UML is represented by a Java class in the API that conforms to the UML specification by the OMG
- These classes provide access to get or set the fields in the Specification Window
- Notably, we do NOT use Constructors to create these elements
  - The ElementsFactory is needed for this
- Once the UML element is created, stereotypes are placed on top using a helper class

- Element
  - Parent of All
  - element.get_relationshipOfRelatedElement() returns all relationships for an element
  - element.getOwnedElement() returns all owned elements underneath this element (DOWN)
  - element.getOwner() returns the owner of this element (UP)
  - element.setOwner(someElement) places the element in the Containment Tree
- Relationship
  - Parent of All Relationships
- NamedElement
  - Provides the Name property (not all elements have a name)
- Package / Model
  - Containers within UML
- Class / Property
  - Common UML structural elements
- Diagram
  - Any diagram in Cameo (the type is determined by a property called DiagramType)
- PresentationElement / DiagramPresentationElement
  - Symbols on diagrams
  - presentationElement.getElement() returns the element for the symbol

ENOLA TECHNOLOGIES

Application Classes (1)

- The Application Classes consist of the main classes in the User Interface
- Nearly every automation begins with accessing the Application Instance, its GUILog, and its active Project.
- If the model is to be edited, a session must first be created and then closed when completed.

- Application
  - The main class for current instance of CATIA Magic running
  - Getting the Instance of from Application is usually the first thing done in plugins/macros
- MainFrame
  - The primary graphical user interface (GUI) of CATIA Magic
  - Provides access to the menus and for syncing windows created
- GUILog
  - The log file class for CATIA Magic
- Project
  - The instance of the project(s) active in the instance of CATIA Magic
  - Provides access to the root of the Containment Tree
- ElementsFactory
  - The element creation mechanism for a project
- SessionManager
  - Manages sessions for actions taken in CATIA Magic (for undoing/redoing)

ENOLA TECHNOLOGIES

Application Classes (2)

- Useful functions:
  - Application.getInstance().getProject()
    - Get active project element
  - Application.getInstance().getMainFrame()
    - Get access to the GUI of CATIA Magic
  - Application.getInstance().getGUILog().log("Hello World")
    - Print to the log file
  - project.getPrimaryModel()
    - Returns the root Model element in the Containment Tree
    - A great starting point for traversing the entire model
  - project.getElementsFactory()
    - Gets the elementsFactory to create new elements
  - elementsFactory.createMETACLASSInstance()
    - Creates a new element of type METACLASS
  - mainFrame.getMainMenuBar()
    - Gets the File -> Edit -> View etc. Menu Bar to add new buttons
  - SessionManager.getInstance().createSession("Session Name Here")
    - Creates a new session for making edits to the model
  - SessionManager.getInstance().closeSession()
    - Closes the active session

Helper Classes (1)

- The Helper Classes provide useful features for working in the UML framework
- The most useful is typically StereotypesHelper
  - This applies stereotypes, searches for elements, and gets and sets tagged values

- StereotypesHelper
  - Primary helper for working with stereotypes on elements and their tagged values
- CoreHelper
  - Gets/sets the documentation of elements
  - Gets/sets the source and target (client/supplier) of relationships
- Finder
  - A utility class for finding elements within the Containment Tree meeting given criteria
- ElementSelectionDlgFactory
  - A factory for creating a Select Element Window in CATIA Magic
  - Many options available for single/multi-select, default options selected, and element creation allowed in window

Helper Classes (2)

- Useful functions:
  - StereotypesHelper.addStereotype(someElement,someStereotype)
    - Adds a stereotype to an element
  - StereotypesHelper.hasStereotype(someElement,someStereotype)
    - Checks if the element has a specific stereotype
  - StereotypesHelper.getStereotypePropertyValue(someElement,someStereotype,”TagName”)
    - Gets the Tagged Value of the element
  - StereotypesHelper.setStereotypePropertyValue(someElement,someStereotype,”TagName”,value)
    - Sets the Tagged Value of the element
  - StereotypesHelper.getStereotypedElements(someStereotype)
    - Returns ALL elements with that stereotype applied
  - CoreHelper.getComment(someElement)
    - Gets the documentation for an element
  - CoreHelper.getClientElement(someRelationship)
    - Gets the client/source of a relationship
  - CoreHelper.getSupplierElement(someRelationship)
    - Gets the supplier/target of a relationship
  - Finder.byQualifiedName().find(project,”QUALIFIEDNAME”)
    - Finds the element with the given qualified name

Requirement Creation Sample

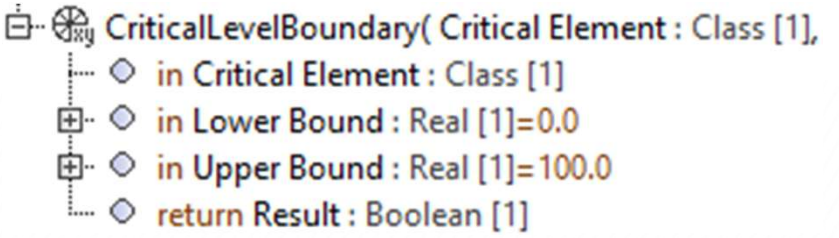
```
import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.openapi.uml.SessionManager;
import com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;

project = Application.getInstance().getProject();
mainModelPackage = project.getPrimaryModel();
elementsFactory = project.getElementsFactory();

sysmlProfile = StereotypesHelper.getProfile(project,"SysML");
requirementStereotype = StereotypesHelper.getStereotype(project,"Requirement",sysmlProfile);

Class createRequirementElement(name,id,text,owner)
{
    newRequirement = elementsFactory.createClassInstance();
    newRequirement.setName(name);
    StereotypesHelper.addStereotype(newRequirement,requirementStereotype);
    StereotypesHelper.setStereotypePropertyValue(newRequirement,requirementStereotype,"Id",id);
    StereotypesHelper.setStereotypePropertyValue(newRequirement,requirementStereotype,"Text",text);
    newRequirement.setOwner(owner);
}

void print(charArray)
{
    Application.getInstance().getGUILog().log(charArray);
}

try
{
    SessionManager.getInstance().createSession("New Requirement");

    requirementName = "My Created Requirement";
    requirementId = "MyReq-1";
    requirementText = "The system shall...";

    newRequirement = createRequirementElement(requirementName,requirementId,requirementText,mainModelPackage);
    print("New Requirement Created")
}
finally
{
    SessionManager.getInstance().closeSession();
}
```

Managing Extensions

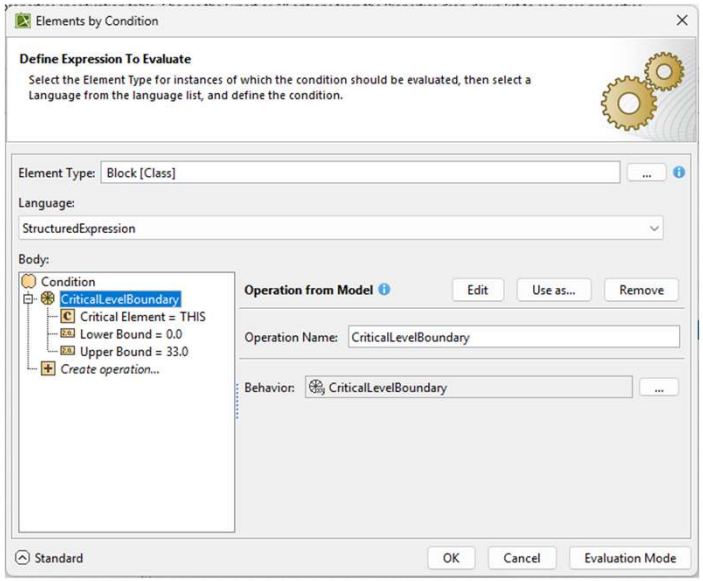
ENOLA TECHNOLOGIES

Managing Extensions

- Structured Expressions
  - Structures expressions can be stored and reused within profiles as OpaqueBehaviors
  - Parameters for inputs are created within OpaqueBehaviors and are assignable within the Structured Expression
  - The Structured Expression is stored within the Body and Language property of the OpaqueBehavior
- Simulation
  - Code for simulations is stored within the behaviors/constraints of an Activity, State Machine, or Parametric Diagram.



Operation from Model



ENOLA TECHNOLOGIES

Managing Plugins

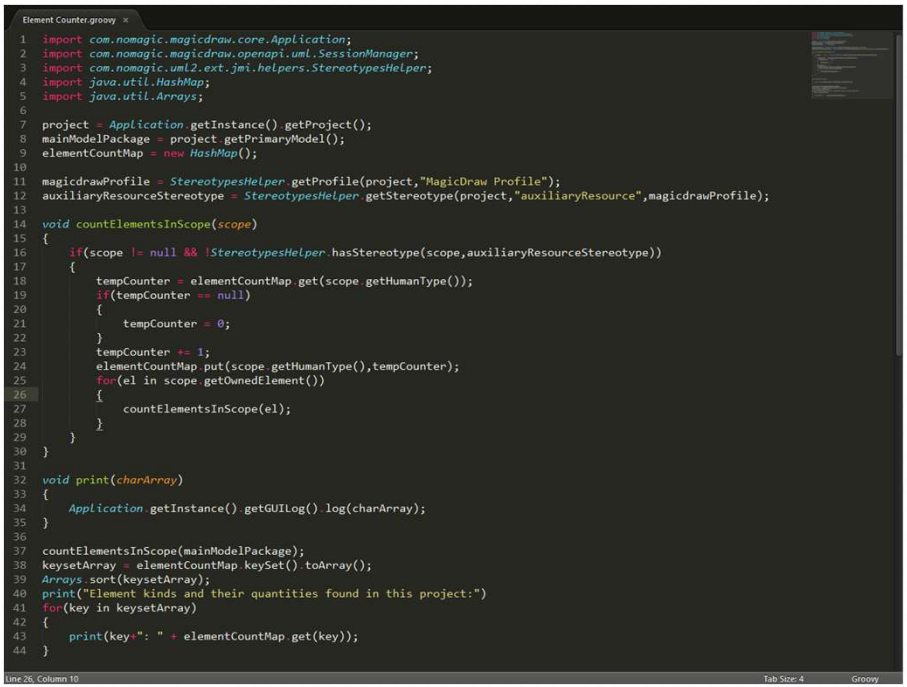
- Plugins
  - Plugin code should be managed with an external repository such as GitHub
  - Once compiled and installed, the code stays locked within the plugins directory of the installation
  - Plugin code is often developed in IDEs such as IntelliJ, Eclipse, and NetBeans
  - Most users will require IT support to install plugins into CATIA Magic

GitHub

ENOLA TECHNOLOGIES

Managing Macros

- Macros
  - Macro code can be stored within the application OR a project.
  - Macros stored to the application are added to the Tools -> Macros menu and can be run on any project within THAT instance of CATIA Magic.
    - Each user needing to run the macro would load it into their application using Tools -> Macros -> Organize Macros
    - This does NOT require elevated privileges
  - Macros stored to a project are maintained in a Macro type element
    - The MacroEngine profile is loaded automatically
  - All Macros in a project are automatically available to all users of the project under Tools -> Macros
  - Macros can be developed in IDEs (like Plugins) or in advanced text editors such as Sublime or Notepad++



ENOLA TECHNOLOGIES

Best Practices

- When developing any extension for CATIA Magic, always consider the range of users for the extension
  - The broader the audience, the more resilient the extension needs to be to users improperly using the code or bad data being used with it
  - For very specific applications, only key users should have access to extensions
- Macros stored in projects are model elements and are thus subject to version controlling in TWC
- Plugins should be CMed per the organization's standard operating procedure
- For wizards and large functionality used by a team, plugins are best suited
- For automations, transformations, or reports, macros are best suited

Demo of Macros

QUESTIONS?

ENOLA TECHNOLOGIES