# Tekla Internal API

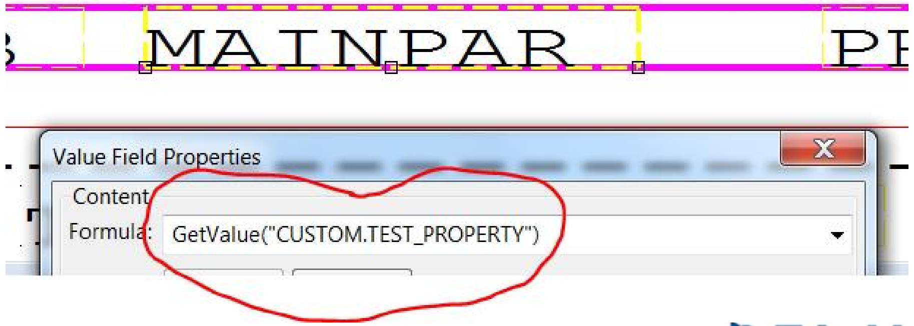
## Custom Property Plug-in Basics

## Objective

- Understand the basics of Custom Property Plug-ins
  - How to define
  - How to add your Plug-in to Tekla Structures
  - How to debug a Plug-in
  - How to connect to Template
- Understand Plug-in logical structure
  - Value types
  - Input

# What are Custom Property plug-ins

- Enables external calculation for template fields
  - I.e. for custom areas, special product codes, custom marks in drawings
- Identification based on “CUSTOM.” in property name
- Are implemented as .NET class libraries (dll) with specific metadata
- Currently internal API, not meant for public use
- Execution is synchronous
  - Asynchronous actions forbidden!

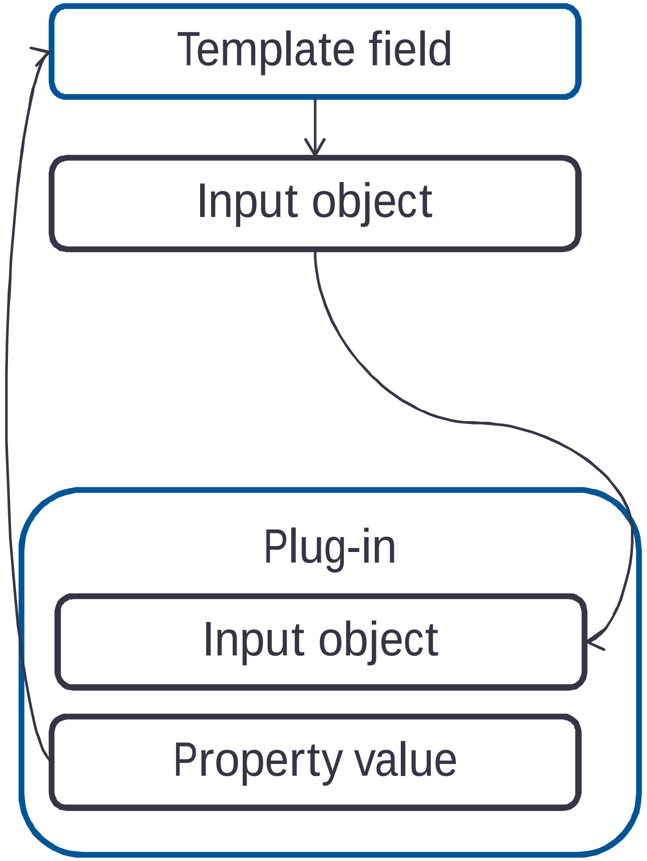


GetValue("CUSTOM.TEST_PROPERTY")

Trimble

# Data, input, and execution

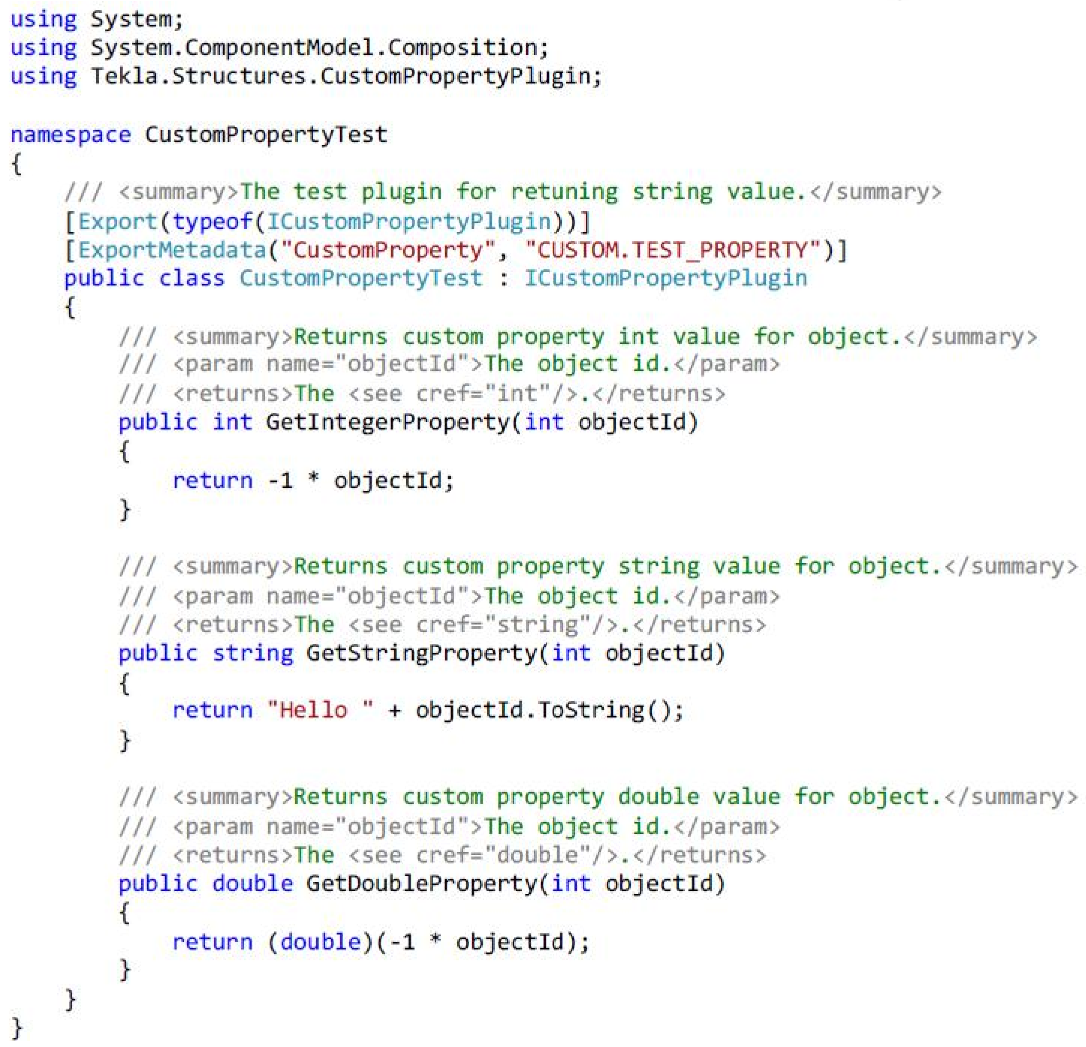
- Custom property value is needed for template
- New Plug-in started
  - Constructor method runs internally
  - Correct method based on value type (int, double, string) is called
  - Input object id based on context is passed to plug-in as argument in call method
- Custom property value is calculated and passed back to template field



# Basic requirements of Custom property

- References to System.ComponentModel.Composition and Tekla.Structures.CustomPropertyPlugin.dll
- Class implements ICustomPropertyPlugin interface
  - Interface is defined in Tekla.Structures.CustomPropertyPlugin.dll
- Needed Custom Attributes
  - Export attribute for MEF: [Export(typeof(ICustomPropertyPlugin))]
  - Property type and name : [ExportMetadata("CustomProperty","CUSTOM.TEST_PROPERTY")]
- Unit conversions defined in template setting files

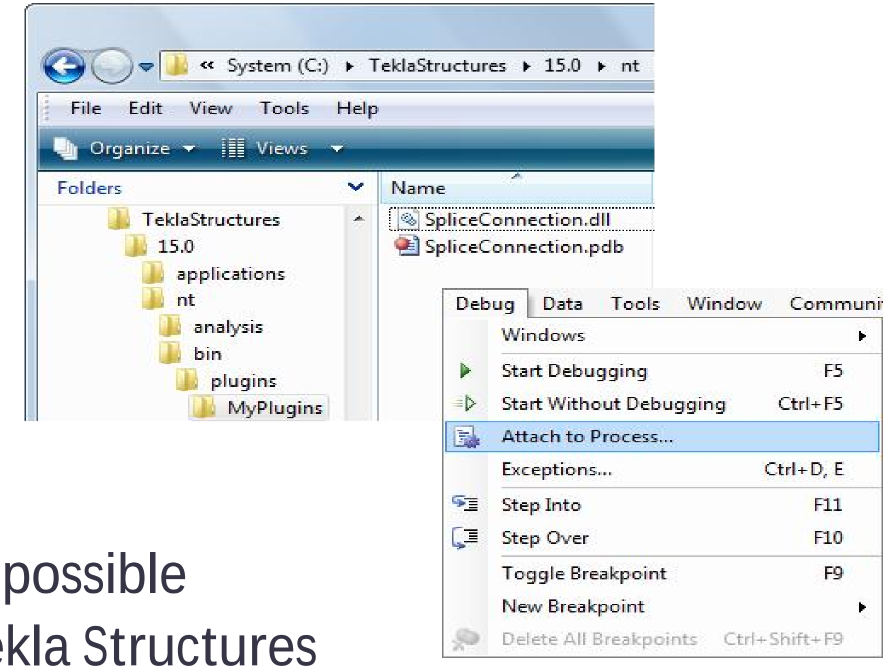
# Example:



```csharp
using System;
using System.ComponentModel.Composition;
using Tekla.Structures.CustomPropertyPlugin;

namespace CustomPropertyTest
{
    /// <summary>The test plugin for retuning string value.</summary>
    [Export(typeof(ICustomPropertyPlugin))]
    [ExportMetadata("CustomProperty", "CUSTOM.TEST_PROPERTY")]
    public class CustomPropertyTest : ICustomPropertyPlugin
    {
        /// <summary>Returns custom property int value for object.</summary>
        /// <param name="objectId">The object id.</param>
        /// <returns>The <see cref="int"/>.</returns>
        public int GetIntegerProperty(int objectId)
        {
            return -1 * objectId;
        }

        /// <summary>Returns custom property string value for object.</summary>
        /// <param name="objectId">The object id.</param>
        /// <returns>The <see cref="string"/>.</returns>
        public string GetStringProperty(int objectId)
        {
            return "Hello " + objectId.ToString();
        }

        /// <summary>Returns custom property double value for object.</summary>
        /// <param name="objectId">The object id.</param>
        /// <returns>The <see cref="double"/>.</returns>
        public double GetDoubleProperty(int objectId)
        {
            return (double)(-1 * objectId);
        }
    }
}
```

## Running and Debugging a Plug-in

- Preparation
  - Copy the project dll and pdb file to the Plug-ins folder or a sub folder
  - Run Tekla Structures
  - Set breakpoints in the code
- Debugging
  - Debug > Attach to process
  - Run or modify the Plug-in
  - Debug > Stop debugging
- Changes
  - On the fly code changes are not possible
  - A new dll requires a restart of Tekla Structures



# Notes

- Visual Studio
  - Plug-in projects are a ‘Class Library’ (e.g. dll)
- Plug-in dlls
  - More than one Plug-in can be created in the same dll under the same project
- No message boxes and pop-up dialogs
  - Any dialogs and message boxes will be shown again for each Plug-in instance
- Trouble shooting
  - Information about problems loading Plug-ins or problems with the dialog can be found from the session history log

## Limitations and known problems

- Performance issues
  - Plug-ins load when first one is called
- Error handling
  - Information level in log file is basic

## Exercise

Custom Property Plug-ins

## Objective: Create a Custom property plug-in for custom part mark

1. Create new class library project CustomPartMark
2. Add references to System.ComponentModel.Composition, Tekla.Structures.CustomPropertyPlugin.dll
3. Add needed custom attributes before class definition

   [ExportMetadata("CustomProperty", "CUSTOM.PART_POS")]
4. Implement methods in ICustomPropertyPlugin
5. In GetStringProperty(int objectId) return new part mark string
   - Either based on given id or
   - Add reference to Tekla.Structures.Model.dll, select the model object using new Model().SelectModelObject(new Identifier(id)), get report property (”PART_POS”) and return modified property to core
6. Modify a report (i.e. Part_List.rpt) and change GetValue(“PART_POS”) -> GetValue(“CUSTOM.PART_POS”) in “PART_POS” field
7. Copy dll to subfolder of Plugins, start TS and run report

– Fetches UDAs of parts and
– Writes UDAs to Xml file
– Filename given as parameter for plug-in
– Inp used for dialog definition

# Thank You