Advanced Customization - Overview

• Add-in versioning
• DAML dependencies across add-ins
  - How to modify one add-in from another via daml
• Categories
  - Pro’s usage of categories
  - Custom categories for plug-and-play UIs
Advanced Customization: Add-In versioning

- There may be scenarios where you want to deploy:
  - Newer versions of the *same* Add-in
    - Fix bugs
    - Augment with additional capabilities
  - …or both
Advanced Customization: Add-In versioning scheme

- Deploy bug-fixes and feature enhancements
- Pro will always load the latest version of an Add-in regardless of its location
  - Update the `version` attribute on `<AddInInfo>` in the Config.daml

```xml
<AddInInfo ...
version="1.0"
</AddInInfo>
<AddInInfo ...
version="1.0.0.2"
</AddInInfo>
<AddInInfo ...
version="1.0.1"
</AddInInfo>
<AddInInfo ...
version="1.3"
</AddInInfo>
<AddInInfo ...
version="2.0"
</AddInInfo>
```

Earliest

Major.Minor.Patch.Build Number

Latest
Advanced Customization: DAML Dependencies

• Deploy functionality incrementally across multiple Add-ins
  - “Core” or “Basic” Add-in + other Add-ins as optional packages or extensions
  - E.g. extra tools or UI components
Advanced Customization: DAML updates

- Use DAML to update the core Add-in module – tabs, groups, menus, etc.
  - Reference the module to be updated (“dependency”)
  - Specify the DAML element to be updated (tab, menu, etc)
  - Specify the update (typically an “insert”)

```xml
<AddInInfo id= .....>
  <Name>EnhancedOptionsAddin</Name>
  ...
  <updateModule refID="CoreOptionsAddin_Module">
    <tabs>
      <updateTab refID="CoreOptionsAddin_Tab1">
        <insertGroup refID="EnhancedOptionsAddin_Group1"/>
      </updateTab>
    </tabs>
  </updateModule>
</AddInInfo>
```

- Enhanced options add-in has a “dependency” on the Core Options Addin
  - Examples: https://github.com/Esri/arcgis-pro-sdk/wiki/ProSnippets-DAML
Advanced Customization: Add-In dependency

- Pre: ArcGIS Pro 2.2
  - Use a `dependency` tag in the Config.daml
  - Important if the “enhancements” add-in(s) processed before the “core” add-in it modifies
  - Use either add-in info id (guid) or daml module id

```
<dependencies>
  <dependency name="CoreOptionsAddin_Module"/>
  <dependency name="aacc0821-e371-407d-90c6-92fa5eb67b2f"/>
</dependencies>
```
Advanced Customization

- Demo
  - Multi-addin + Daml
Advanced Customization: Categories

- Categories
  - What are “Categories”? Why would I need them?
    - Overview of Pro’s usage of categories.
  - Using Custom Categories to dynamically configure our Add-in UIs
Advanced Customization: Categories

- What are Categories?
  - Mechanism for logically grouping together components that provide the same functionality
    - “Plug and Play” mechanism.
    - Declared and referenced in daml
    - Implemented in code
Advanced Customization: Categories

- Why would I need them?
  - Support scenarios where we want to dynamically configure our UIs
    - (doesn’t *have* to be UI related but that is the most common scenario)
  - No equivalent DAML mechanism
Advanced Customization: Pro Categories

- Pro has a variety of categories for its internal use.
  - For example: ...
- Construction tool categories to allow custom construction tools to be dynamically added to the Create Features palette in Pro.

- esri_editing_construction_point, esri_editing_construction_polyline
- esri_editing_construction_polygon, esri_editing_construction_multipoint
- esri_editing_construction_annotation, esri_editing_construction_dimension
Advanced Customization – Pro Categories

- **Embeddable Controls** - Custom UI associated with tools
  - Can provide a UI “overlay” on top of the MapView when the tool is activated

  https://github.com/Esri/arcgis-pro-sdk-community-samples/tree/master/Map-Exploration/BasicMapTool

  - Can host custom tool options on the Active Template pane (for construction tools)

Advanced Customization – Pro Usage of Categories

• Two techniques available for referencing or “registering” in a categories:
  - Plugins – mainly tools and buttons – use a categoryRefID attribute
    - E.g. Construction Tools
  - Everything else uses an `<updateCategory refID="..."/>` tag
    - E.g. Embeddable Controls
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - Plugins – mainly tools and buttons – add a categoryRefID attribute
  - categoryRefID should be set to the daml category id.

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
  <!--or use esri_editing_construction_polyline, _polygon, _annotation, etc. as needed-->
  ...
</tool>
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - Plugins – mainly tools and buttons – add a categoryRefID attribute
  - categoryRefID should be set to the daml category id.

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...

<!--or use esri_editing_construction_polyline, _polygon, _annotation, etc. as needed--> ...
</tool>
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - Plugins – mainly tools and buttons – add a categoryRefID attribute
  - categoryRefID should be set to the daml category id.

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
  <!--or use esri_editing_construction_polyline, _polygon, _annotation, etc. as needed-->
  ...
</tool>
```
Advanced Customization : Pro usage of Categories

• Polygon construction tool example:
  - `categoryRefID` set to “esri_editing_construction_polygon”
  - Dynamically adds the tool to the `polygon` Create Features palette.

```xml
<tool id="Example_ConstructionTool2" categoryRefID="esri_editing_construction_polygon"/>
<!—or use esri_editing_construction_polyline, _polygon, _annotation, etc. as needed-->...
</tool>
```
Advanced Customization : Pro usage of Categories

• Polygon construction tool example:
  - categoryRefID set to “esri_editing_construction_polygon”
  - Dynamically adds the tool to the **polygon** Create Features palette.

```xml
<tool id="Example_ConstructionTool2" categoryRefID="esri_editing_construction_polygon" ...
<!--or use esri_editing_construction_polyline, _polygon, _annotation, etc. as needed-->
... 
</tool>
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - "Non" Plugins – such as embeddable controls or custom components must use an
    `<updateCategory` element with a refID set to the daml category id

```xml
<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className="...">
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>

//Tool references the embeddable control
internal class BasicMapTool: MapTool {
  public BasicMapTool() {
    this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
  }
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - “Non” Plugins – such as embeddable controls or custom components must use an <updateCategory> element with a refID set to the daml category id

```xml
<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...">
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```

```java
//Tool references the embeddable control
internal class BasicMapTool: MapTool {
  public BasicMapTool() {
    this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
  }
}
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - “Non” Plugins – such as embeddable controls or custom components must use an `<updateCategory>` element with a `refID` set to the daml category id

```xml
<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...">
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>

//Tool references the embeddable control
internal class BasicMapTool: MapTool {
  public BasicMapTool()
  {
    this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
  }
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - “Non” Plugins – such as embeddable controls or custom components must use an
    `<updateCategory` element with a refID set to the daml category id

```xml
<updateCategory refID="esri_embeddableControls">
    <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...">
        <content className="EmbeddableControl1View" />
    </insertComponent>
</updateCategory>
```

//Tool references the embeddable control
internal class BasicMapTool: MapTool {
    public BasicMapTool()
    {
        this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
    }
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  
  "Non" Plugins – such as embeddable controls or custom components must use an `<updateCategory` element with a refID set to the daml category id

```xml
<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className="...
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```

//Tool references the embeddable control
internal class BasicMapTool: MapTool {
  public BasicMapTool()
  {
    this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
  
```
Advanced Customization – Pro Usage of Categories

- Two techniques available for referencing categories:
  - “Non” Plugins – such as embeddable controls or custom components must use an `<updateCategory` element with a refID set to the daml category id

```html
<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className="...
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```

//Tool references the embeddable control
internal class BasicMapTool: MapTool {
  public BasicMapTool()
  {
    this.OverlayControlID = "BasicMapTool_BasicEmbeddableControl";
  
```
Advanced Customization: Categories:

Custom Categories
- How do I define them?
- How do I use them for my own UIs?
Advanced Customization: Categories:

• On github we have two samples for custom categories:

  **Scenario 1**: Using Categories to group items in a ArcGIS Pro Gallery UI element.

  [Image of ArcGIS Pro Gallery UI with categories]


  **Scenario 2**: Enhance a base add-in with extra functionality from additional add-ins.

  [Image of add-in window with extra reports]

Advanced Customization: Categories:

**Scenario:**
Dynamically add plugin content to a gallery.

- Content may be contained across multiple add-ins
- Different users may have different combinations (of addins)

No DAML support for “declaring” content for a gallery
i.e. no such things as `<updateGallery ….><insertButton …>`
Advanced Customization: Custom Categories:

**Solution:**
- Define a custom category for our gallery

“Host or Owner”
- Gallery add-in will be responsible for defining and reading the category
  - Load category “contents”

“Consumer” or “Provider”
- Plugin content will register in the custom category
  - Use categoryRefID
Advanced Customization: Define Category – Host or Owner

Host

• Step 1:

```xml
<categories>
  <insertCategory id="AcmeCustom_AnalysisTools"/>
</categories>
```
Advanced Customization: Category AcmeCustom_CategoryTools

Host

• Step 2:
  - Identify category requirements. This is called “the contract”
    - Items must be Buttons or Tools – (i.e. be “Plugins” - implement IPluginWrapper)
    - Items must provide a version
    - Items can optionally identify a group

  - It is the responsibility of the “Host” or “Owner” of the category to enforce the contract.
Advanced Customization – Category Contract - Consumer

Consumer

• Step 3:
  - Add-ins register their relevant content for use in the gallery
  - Registered “content” must satisfy our Category **contract** requirements:
    1. Be “plugins”
    2. Have a version attribute
    3. Optionally have a group name
Advanced Customization – Register in Category - Consumer

Consumer

• Step 3:
  - In the Config.daml:
    - We add a `categoryRefId` attribute on the button or tool DAML declaration
    - We add a content element with a version (required) and group (optional) attribute to each button or tool.
Advanced Customization: Consumer - Config.daml

- One of our buttons in the Config.daml

```html
<controls>
  <button id="..." className="...">
    <tooltip ...></tooltip>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Add a categoryRefID attribute to the `<button ...>`

```xml
<controls>
  <button id="..." className="..." categoryRefID="">
    <tooltip ...></tooltip>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Set `categoryRefID = “Your category id”`

```xml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip ...></tooltip>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Add a `<content .../>` child element to `<button .../>`

```daml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip ...></tooltip>
    <!-- Use a content element for custom attributes-->
    <content/>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Add a version attribute and (optionally) a group attribute.
  - version is **required** by our AcmeCustom_AnalysisTools contract.

```xml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip ...></tooltip>
    <!-- Use a content element for custom attributes-->
    <content version="1.0"/>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Add a version attribute and (optionally) a group attribute.
  - version is **required** by our AcmeCustom_AnalysisTools contract.

```xml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip ...></tooltip>
    <!-- Use a content element for custom attributes-->
    <content version="1.0" group="Special"/>
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Repeat for any DAML command that we want to add to our gallery

```daml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip .../></tooltip>
    <!-- Use a content element for custom attributes-->
    <content version="1.0" group="Special"/>
  </button>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
  </button>
</controls>
```
Advanced Customization: Consumer - Config.daml

- Repeat for any DAML command that we want to add to our gallery

```xml
<controls>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <tooltip ...></tooltip>
    <!-- Use a content element for custom attributes-->
    <content version="1.0" group="Special"/>
  </button>
  <button id="..." className="..." categoryRefID="AcmeCustom_AnalysisTools">
    <content version="1.0"/>
  </button>
</controls>
```
Advanced Customization: Consume Category – Host or Owner

Host

• Step 4:
  - Read the category components
    - Implement in code in our gallery initialization.
  - Verify the contract:
    - Each component is a plugin
    - Each component must have a version attribute
    - Each component may have a group attribute
  - If any component fails our contract we do not add it to our gallery
Advanced Customization: Pro API Categories class

• To read the category components we use:
  - Categories.GetComponentElements("Your_Category_DAML_Id")
    - Returns an enumeration of ArcGIS.Desktop.Framework.ComponentElement
    - Each ComponentElement represents a component record (read from add-in Config.daml)
      - In our case, these are our <button ... categoryRefId=""> records.

• For each ComponentElement:
  - Use component.GetContent() to return the <content/> element
    - Returned as a System.Xml.Linq.XElement.
    - Query it for custom attributes and child elements
    - Or use component.ReadAttribute("attribute_name")
Advanced Customization: Loading the components in code

• Load the Gallery, e.g. in our Gallery class constructor:

```csharp
//read all registered components from the category
foreach (var component in Categories.GetComponentElements("AcmeCustom_AnalysisTools")) {
}
```
Advanced Customization: Loading the components in code

- Test each component for our contract – only add the ones that are “good”

```csharp
//read all registered components from the category
doctest (var component in Categories.GetComponentElements("AcmeCustom_AnalysisTools")) {

    var content = component.GetContent();
    var version = content.Attribute("version").Value; //check the version attribute...

    var group = component.ReadAttribute("group") ?? ""; //group is optional

    //check our components are indeed plugins
    var plugin = FrameworkApplication.GetPlugInWrapper(component.ID); //IPluginWrapper
    if (plugin != null)
        //if we have a valid component, make a gallery item our of it
        Add(new AcmeGalleryItem(component.ID, group, plugin));

```
Advanced Customization

Scenario 1: Configure Gallery

- Demo
Advanced Customization

Summary

• Add-in versioning scheme for deploying successive versions of the same add-in

• Using DAML across multiple add-ins
  - How to modify one add-in from another via daml

• Categories
  - Pro’s usage of categories
  - Custom categories for plug-and-play Uis
    - Host and Consumer roles
## ArcGIS Pro SDK for .NET – Technical Sessions

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Session</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon, Nov 04</td>
<td>1:00 pm – 1:45 pm</td>
<td>Learning Customization and Extensibility</td>
<td>Salon Durieux</td>
</tr>
<tr>
<td>Tue, Nov 05</td>
<td>9:00 am – 9:45 am</td>
<td>Understanding the CIM, a Guide for Developers</td>
<td>Salon Humboldt</td>
</tr>
<tr>
<td></td>
<td>2:00 pm – 2:45 pm</td>
<td>Intermediate Editing</td>
<td>Salon Durieux</td>
</tr>
<tr>
<td></td>
<td>4:00 pm – 4:45 pm</td>
<td>Advanced Customization Patterns</td>
<td>Salon Durieux</td>
</tr>
<tr>
<td>Thu, Mar 07</td>
<td>2:00 pm – 2:45 pm</td>
<td>Learning Customization and Extensibility</td>
<td>Salon Humboldt</td>
</tr>
</tbody>
</table>
Advanced Customization Patterns

• Questions?

Please Take Our Survey on the App

Download the Esri Events app and find your event

Select the session you attended

Scroll down to find the feedback section

Complete answers and select “Submit”
Advanced Customization – Pro Usage of Categories

- Category components can also add a custom `<content .../>` tag:
  - Allows components to have custom attributes related to the parent category

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
  <content insert="before" placeWith="esri_editing_SketchPointTool" />
</tool>

<tool id="Example_Editing_Tool1" categoryRefID="esri_editing_CommandList" ...>
  <content L_group="Intermediate Editing" gallery2d="true" gallery3d="false" />
</tool>

<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...>
    <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```
Advanced Customization – Pro Usage of Categories

- Category components can also add a custom `<content .../>` tag:
  - Allows components to have custom attributes related to the parent category

```
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
    <content insert="before" placeWith="esri_editing_SketchPointTool" />
</tool>

<tool id="Example_Editing_Tool1" categoryRefID="esri_editing_CommandList" ...>
    <content L_group="Intermediate Editing" gallery2d="true" gallery3d="false" />
</tool>

<updateCategory refID="esri_embeddableControls">
    <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...">
        <content className="EmbeddableControl1View" />
    </insertComponent>
</updateCategory>
```
Advanced Customization – Pro Usage of Categories

• Category components can also add a custom `<content .../>` tag:
  - Allows components to have custom attributes related to the parent category

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
  <content insert="before" placeWith="esri_editing_SketchPointTool" />
</tool>

<tool id="Example_Editing_Tool1" categoryRefID="esri_editing_CommandList" ...>
  <content L_group="Intermediate Editing" gallery2d="true" gallery3d="false" />
</tool>

<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...
  <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```
Advanced Customization – Pro Usage of Categories

- Category components can also add a `<custom .../>` tag:
  - Allows components to have custom attributes related to the parent category

```xml
<tool id="Example_ConstructionTool1" categoryRefID="esri_editing_construction_point" ...>
  <content insert="before" placeWith="esri_editing_SketchPointTool" />
</tool>

<tool id="Example_Editing_Tool1" categoryRefID="esri_editing_CommandList" ...>
  <content L_group="Intermediate Editing" gallery2d="true" gallery3d="false" />
</tool>

<updateCategory refID="esri_embeddableControls">
  <insertComponent id="BasicMapTool_BasicEmbeddableControl" className=" ...
  <content className="EmbeddableControl1View" />
  </insertComponent>
</updateCategory>
```
Advanced Customization: Categories:

**Scenario 1**: Using Categories to group items in a ArcGIS Pro Gallery UI element.

**Scenario 2**: Enhance a base add-in with extra functionality from additional add-ins.
Advanced Customization: Scenario 1 Configure Gallery

- Configure a gallery with commands
  - Gallery does not provide DAML-based customization for gallery items
  - i.e. there is no such thing as
  - We will use a category
  - We ~can~ use the category in our DAML to identify which of our commands should be added to the gallery

```xml
<updateGallery ...><insertButton ...>
```
Advanced Customization: Scenario 2 - Custom Reports

- Enhance a base add-in with extra functionality from additional add-ins.
  - Let's say we release a standard or “default” analysis report in an add-in
  - Users in our organization can get different reports (in additional add-ins) depending on their job function
  - We do not want to recompile and build “special” releases each time we need to deploy a different report…
Advanced Customization: Scenario 2 - Custom Reports

- Step 1: Define new Category in the default add-in

```xml
<categories>
  <insertCategory id="AcmeCustom_Reports"/>
</categories>
```

- Step 2: Document category contract – these are “custom” requirements
  - Require components implement `ICustomReport`

```csharp
public interface ICustomReport {
    string Label { get; }
    string Title { get; }
    string Details { get; }
    Task RunAsync();
}
```
Advanced Customization: Scenario 2 - Custom Reports

- **Step 3: Implement the report providers**
  - They can be external modules (i.e. in different add-ins).
  - Standard .NET class
  - Implements the category contract (i.e. “ICustomReport” interface)
  - Register each component within the category in the Config.daml

- For non-commands we must use:

```xml
<updateCategory...><insertComponent ...>
```
in our DAML not “categoryRefID”
Advanced Customization: Scenario 2 - config.daml

• In our config.daml, add an `<updateCategory>` element with refID="AcmeCustom_Reports"

```xml
<categories>
  <updateCategory refID="AcmeCustom_Reports">
  
  </updateCategory>
</categories>
```
Advanced Customization: Scenario 2 - config.daml

- Add an \textbf{insertComponent} child element
  - “className” points to the .NET class that implements our report (and interface)

```xml
<categories>
  <updateCategory refID="AcmeCustom_Reports">
    <!-- This element is required -->
    <insertComponent id="Acme_DefaultReport" className="DefaultReport">
    </insertComponent>
  </updateCategory>
</categories>
```
Advanced Customization: Scenario 2 - config.daml

- Add a child `<content>` element to the `insertComponent`. This is required.

```xml
<categories>
  <updateCategory refID="AcmeCustom_Reports">
    <!-- This element is required -->
    <insertComponent id="Acme_DefaultReport" className="DefaultReport">
      <!-- A content element is REQUIRED. All attributes are custom! -->
      <content/>
    </insertComponent>
  </updateCategory>
</categories>
```
Advanced Customization: Scenario 2 - config.daml

• If the category requires custom content – e.g. a name or version, then this should be added to the content element.

```xml
<categories>
  <updateCategory refID="AcmeCustom_Reports">
    <!-- This element is required -->
    <insertComponent id="Acme_DefaultReport" className="DefaultReport">
      <!-- A content element is REQUIRED. All attributes are custom! -->
      <content version="1.0" name="custom report" label="etc,etc"></content>
    </insertComponent>
  </updateCategory>
</categories>
```
Advanced Customization: Scenario 2 - Load the Reports

- Step 4: Implement the base report add-in:
  - Load the category components (report providers)
    - Use `CategoriesGetComponentElements("AcmeCustom_Reports")`
  - Test for contract requirements
    - For each `ComponentElement`
      - Custom content attributes - version and name
      - Each component implements ICustomReport
      - Instantiate a custom report instance using `Component.createComponent()`
Advanced Customization: Scenario 2 - Load the Reports

- Read the category using:

```csharp
_items = new List<IAcmeCustomReport>();

foreach (var component in Categories.GetComponentElements("AcmeCustom_Reports")) {

}
```
Advanced Customization: Scenario 2 – Load the Reports

- Test each component for the contract. Components that satisfy the contract are added to our internal collection

```csharp
1. _items = new List<IAcmeCustomReport>();
2. foreach (var component in Categories.GetComponentElements("AcmeCustom_GalleryTools")) {
3.     var content = component.GetContent(); // this is the content from Config.daml
4.     var version = content.Attribute("version").Value;
5.     // test the component – notice we instantiate it using “CreateComponent()”
6.     var report = component.createComponent() as ICustomReport;
7.     if (report != null)
8.         _items.Add(report);
9. }
```
Advanced Customization – Scenario 2:
Custom Reports

• Demo