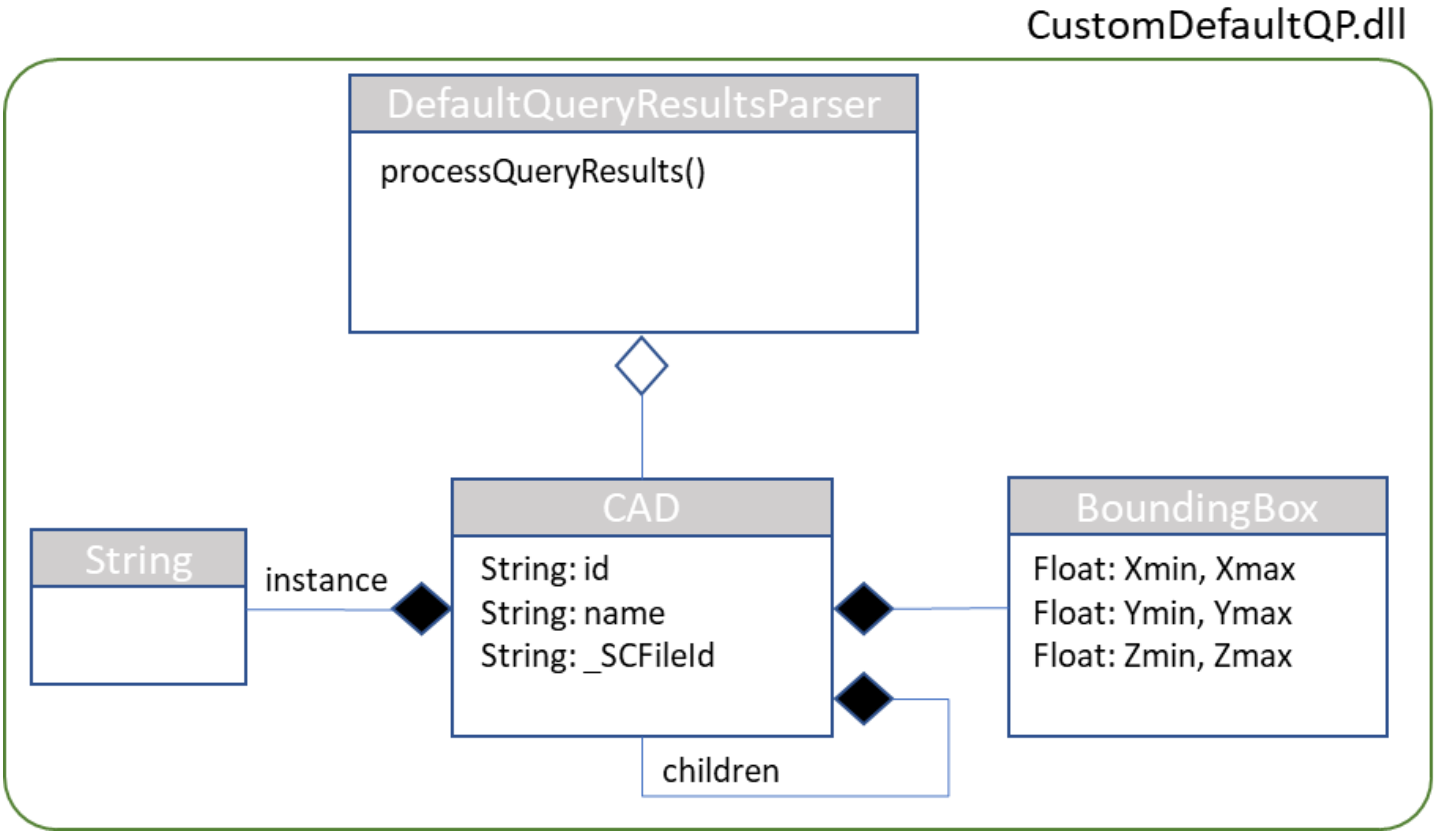
Create the Query Processor DLL

This section describes an example set of classes that were created to process the results from the Default CAD Query. It is provided to show an example of how to process query results and generate the necessary list of Product Occurrences used by the 3D Viewer. The full set of example code is included in section Sample Custom Default Query Processor.

Important

The code fragments shown in this section are for instructional purposes only. Users who create their own Query Processors should ensure the robustness, performance, and correctness of whatever is created to the environment it's meant to serve.



DefaultQueryResultsParser

The class `DefaultQueryResultsParser` performs the two main functions of a Query Processor: Parse/Process the results from the execution of a Query Definition, create a list of `ProductOccurrence` Objects representing the 3D geometry *View Data* to be displayed in the 3D Viewer.

Processing Query Results

Processing the query results entails extracting the information necessary to construct the Product Occurrences as well as any additional information that will be used in constructing Rendering Configurations. At a minimum, the following is needed:

- ID and name of whatever element represents the 3D Geometry in the Tree Grid View.
- ID of the Query Reference that corresponds to the row in the Tree Grid View that the geometry will map to when selected in the viewer.
- BOM hierarchy of elements.
- Transformation matrices for each Instance of 3D Geometry.
- ID of the view file used for each 3D geometry component.

> **Important**
>
> Legacy environments used the CAD ItemType to store data from the CAD Conversion process. The **CAD ConversionInfo** Relationship was added to obviate the need to update a CAD Item (and thus lock it) during the conversion process.

In this example, the element representing 3D Geometry is the CAD Item. Note that this could be a Part Item, or some other custom Item used in the data model. It represents the node that is selected in the Tree Grid View when the 3D component geometry is selected in the 3D View. The hierarchy of 3D Components is also important since transformations are applied top-down. In the default CAD Data model in Aras Innovator, Child CAD Items are transformed relative to the transformation applied to their parent CAD Item, and so on. Transformations are required to position the instance of the 3D geometry in 3D space. Finally, the ID of the view file uniquely identifies the view file in the Aras Innovator Vault.

The Default CAD Query Definition will return one or more root CAD Items, which represents the CAD Item the Query Definition was executed against along with any additional CAD Items selected using Digital Mockup (see Section). Each of these CAD Items should contain the same Properties so processing the full CAD Hierarchy can be done recursively. For each CAD Item parsed from the Query Execution results, a single CAD Item Data Object is created.

In this example, and in Figure 54, the CAD Class is used to process the data from the Query Execution results and store the data identified above. For each instance parsed, the transformation string is extracted and stored as a String within a list. The number of transformation strings in this list represent the number of instances of the current CAD Item relative to its parent CAD Item. Bounding box data, if present, is collected in a simple struct with double attributes for each X/Y/Z min and max values.

Important

Adding bounding box data to the Query Definition and the generated Product Occurrences should be included to help position the camera when the model is initially rendered.
It is recommended to make methods OS agnostic for use with Linux and Windows. The main incompatibility issues in operating systems that need to be considered when implementing the method are path case-sensitivity, path separators, OS-specific line-endings, OS-specific code. For more information about cross-platform development, see section 2.3 Cross-platform development in the *Aras Innovator 31 - Programmer's Guide.*

If a given CAD Item is an assembly, it should contain child CAD Items. In this case, each of those CAD items are processed recursively down to the component CAD Item. Component CAD Items need to parse the associated view file data. Only view files for Component (NOT Assembly) CAD Items should be included in the Product Occurrences. The following is a code snippet showing a possible implementation of processing CAD Item Data:

```
internal void processCAD(QueryBuilderNode qbItem)
{
QryRefId = qbItem.QueryItem.RefId; // Query Item Reference ID
try
{
ID = qbItem.GetProperty("id"); // required
Name = qbItem.GetProperty("name"); // required
}
catch
```

```
{
throw new ArgumentException("Query Definition is not defined properly: 'id' and 'name' Properties are required for CAD Items");
}

// Bounding Box Data. Alternate values ensure that there is a non-empty
// bounding volume defined
// NOTE: Bounding box data should first be retrieved from the related CAD ConversionInfo Relationship. If not present, retrieve from the CAD Item directly
_BBox.MinX = _getPropertyAsDouble(qbItem, "x_min", -1.0);
_BBox.MaxX = _getPropertyAsDouble(qbItem, "x_max", 1.0);
_BBox.MinY = _getPropertyAsDouble(qbItem, "y_min", -1.0);
_BBox.MaxY = _getPropertyAsDouble(qbItem, "y_max", 1.0);
_BBox.MinZ = _getPropertyAsDouble(qbItem, "z_min", -1.0);
_BBox.MaxZ = _getPropertyAsDouble(qbItem, "z_max", 1.0);

// for processing CAD children and associated view file
foreach (var child in qbItem.ChildNodes)
{
if (child.QueryItem.Alias == "CAD ConversionInfo")
```

```
_processCADConversionInfo(child);
if (child.QueryItem.Alias == "CAD Structure")
_processCADStructure(child);
if (child.QueryItem.Alias == "File")
_extractFileInfo(child);
} // foreach
} // processCAD(QueryBuilderNode qbItem)
```

In this sample, the method assumes the type of Query Node provided refers to a `CAD` ItemType. In the Default Query Definition, CAD Items include Properties for each `Property` and Child Nodes for related `Files` and `CAD Structure` . Note that the Properties for ‘id’ and ‘name’ are required and an exception is thrown if they are not found.

Creating a list of Product Occurrences

Creating a list of Product Occurrences entails processing the CAD Data Objects created when parsing the query results. Note that although the API requests a *list* of Product Occurrence objects containing only the root Product Occurrence Objects representing the CAD Items being viewed at the top level – and each Product Occurrence will store the BOM hierarchy as *children*. The following is a code snippet showing a possible implementation for creating Product Occurrences.

```
private void build(ProductOccurrenceInstance parent, CAD c)
  {
foreach(String xFormStr in c.Instances)
{
ProductOccurrenceInstance poInst = new ProductOccurrenceInstance();
```

```
//poInst.Attributes.Add(new ProductOccurrenceAttr("QUERY ITEM REF ID", c.QryRefId));
//poInst.Attributes.Add(new ProductOccurrenceAttr("ITEM ID", c.ID));
poInst.SetServiceProperty("QUERY ITEM REF ID", c.QryRefId);
poInst.SetServiceProperty("ITEM ID", c.ID);

// bounding box
poInst.ProductOccurrenceSource.BoundingBox.Max.X = c.BBox.MaxX;
…

// Testing Rendering Configuration
ProductOccurrenceRenderingConfiguration rConfig = new ProductOcc…();
rConfig.SetColor(Color.FromArgb(55, 40, 0));
rConfig.Opacity = 0.4;
rConfig.Name = "test";
poInst.RenderingConfigurations.Add(rConfig);

// Add Transformation Matrix
poInst.TransformationMatrix = cadInstInfo.XForm;
```

```
// Add SetServiceProperty Method Calls
poInst.SetServiceProperty(cadInstInfo.RefID, cadInstInfo.ID);
poInst.SetServiceProperty(c.CADStrQryRefId, c.CADStructureID);
poInst.SetServiceProperty(c.QryRefId, c.ID);
// Add SetServiceProperty Method Calls


poInst.Name = c.Name;
if (c.Children.Count == 0 && c.SCFileID != null) // Assume this is a component
{
poInst.ProductOccurrenceSource.Name = c.SCFileID; // use File Item Id for name
poInst.ProductOccurrenceSource.FileReferenceByTypeMap.Add(
SourceModelType.Scs, c.SCFileID);
}

if (c.Children.Count == 0 && c.SCFileID == null)
poInst = null; // Invalid Part
else (parent != null)
parent.Children.Add(poInst);
```

```
// Recurse through child hierarchy
foreach (CAD child in c.Children)
build(poInst, child);
} // foreach(String xFormStr in c.Instances)
} // build()
```

This logic is complex, so each key section in the code is enumerated with an explanation that follows:

1. This example uses a recursive method to construct the hierarchy of Product Occurrence Objects. Note that the specific Product Occurrence classes used will either be `ProductOccurrenceInstance` or `ProductOccurrenceSource`. The former refers to actual instances (Assembly or Component) of some 3D geometry, the latter refers to the view geometry file itself.
2. As mentioned previously in the explanation of the CAD Data Model Object class, the list of transformation strings associated with each CAD object denote the number of instances for that CAD Item. There should be at least one.
3. For each Product Occurrence Instance, it's critical that the following two Attributes be included:
    1. Query Item Reference ID. The reference ID is used for all Product Occurrence Instances and is extracted from the Query Results
    2. ID of the Item. This is used to map the instance of the geometry to the node of the CAD Item displayed in the Tree Grid View.

Important

In 3DV 14.0.4, logic for associating Product Occurrence objects with Tree Grid Views (TGV) was modified. This change now uses the SetServiceProperty() method for setting the Query Item Reference ID and Item ID values. Previous versions used Product Occurrence Attributes for this purpose. In order to maintain synchronization between the 3D View and TGV for selection, use of SetServiceProperty() is necessary as shown in the example above.

Important

For each Product Occurrence Instance at the root level, it is necessary to set the "Root Flag" property set by calling ProductOccurenceBase.SetAsRoot() method. In the case of Digital Mockup (section Digital Mockup), there may be several root Product Occurrence Instances. See the full example in Section Sample Custom Default Query Processor.

4. Set the bounding box values for each of the min and max values for X, Y, and Z
5. Add a Rendering Configuration. This is optional, but necessary to add one or more View Modes to the Dynamic or Streaming Viewer (section *Rendering Configurations*). Rendering Configurations add alternate rendering parameters for the geometry associated with the Product Occurrence. This includes color and opacity (transparency). Together they can be used to visually

isolate/highlight certain 3D geometry to show some information about the model or about related Items. There can be multiple rendering configurations for each Product Occurrence, although each must have a unique Name.

6. If a Transformation string is included in the CAD Instance Item, then it should be applied to the TransformationMatrix Property of the Product Occurrence. If there is only one instance, and the geometry can be rendered as it was positioned/stored in the CAD software, then a transformation matrix is not required. In this case, an Identity matrix will be assumed. Note that if there are multiple instances of a CAD Item, then there should be a transformation matrix for each of them. Without a transformation matrix for each, the geometry will be rendered at a location based on how the geometry was stored in the CAD system.
7. Set the Instance properties for instance id, CAD id, and CAD Structure id to support combined rows and the synchronization of instances. This is required to synchronize combined rows in the TGV to extract the path of the combined row and map it to the view.

Important

For each Product Occurrence Instance, it is necessary to set the service properties SetServiceProperty() method. See the full example in Section Sample Custom Default Query Processor.

8. The name used for the Product Occurrence. It's helpful to either use the document number or name of the CAD Item.
9. For the CAD Data Model Object used, if there are no children, then it is assumed that the CAD Item is a component; in which case it should have an associated view file. In this case, create a ProductOccurrenceSource object, setting the properties as shown. Note that the 'Name' needs to store the File ID of the view file.
10. If there are no children (CAD is assumed to be a Component) and there is no associated view file, then there is no sense (and it is invalid) in creating a Product Occurrence since there is no 3D geometry to load.

Important

Errors returned from the processing of Product Occurrence lists regarding the value 'null' for 'key' Parameters are typically related to the inclusion of Product Occurrence Instances for assemblies where no child in that assembly contains associated view data. To avoid these errors, ensure that at least one descendant Product Occurrence has a valid view file attached to it.

11. If a valid parent was passed into the method, then add the newly created Occurrence to it.
12. Recurse through the CAD hierarchy.

Product Occurrence List – View Data

The list of Product Occurrence objects generated by the Query Processor result in an XML set that is like the following example:

```
<Root assyId="…">
  <ModelFile>
    <ProductOccurence Id="0" Name="R-ARM-43" Children="2 10 12">
      <Attributes>
        <Attr Name="QUERY ITEM REF ID" Type="String" Value="…" />
        <Attr Name="ITEM ID" Type="String" Value="…" />
        …
      </Attributes>
    <Transformation RelativeTransfo="…" />
    </ProductOccurence>
    <ProductOccurence Id="2" Name="R-ARM-44" Children="4 6 8">
      <Attributes>
        …
      </Attributes>
      <Transformation RelativeTransfo="…" />
    </ProductOccurence>
    <ProductOccurence Id="4" Name="R-ARM-45" InstanceRef="3">
      <Transformation RelativeTransfo="…" />
    </ProductOccurence>
    <ProductOccurence Id="3" Name="R-ARM-45">
      <ExternalModel Name="107FC69D133C4CBF8B1B90C5AF2E2E30">
        <BoundingBox Max="…" Min="…" />
      </ExternalModel>
    </ProductOccurence>
    <ProductOccurence Id="6" Name="R-ARM-46" InstanceRef="5">
      <Transformation RelativeTransfo="…" />
    </ProductOccurence>
    …
  </ModelFile>
</Root>
```

Assembly Instance

Component Instance

File Source Instance

Note the following about the XML View Data:

- The format of the XML is specific to the HOOPS Viewer; the XML schema is defined by HOOPS. It is generated automatically from the generated Product Occurrence objects.
- < `ProductOccurence` > (one r) XML Elements define both Instance and File source information. This is consistent with the two types of Product Occurrence objects defined above when generating the Product Occurrence list.
- The data is flat, in that the hierarchy is defined by the `Children` attribute.
- All IDs are generated automatically and must be >=0. Each ID uniquely identifies the Product Occurrence.
- Attribute Elements are used for mapping instances of the 3D geometry to the nodes in the Tree Grid View. Note the Query Item Reference and ID are set as defined in section Creating a list of Product Occurrences.
- Component Instances reference their geometry using the `InstanceRef` attribute.
- The system will ensure that there are no duplicate File Source Instances. Thus, reused geometry will ‘point to’ the same Product Occurrence for the File Source.