

# Appendix

## Sample Custom Default Query Processor

This section includes the complete sample code discussed in section *Create the Query Processor DLL*. The following class represents the main class used for custom Query Processing. It contains the `processQueryResults` method discussed in section *Create the Data Processor Method* and the `build` method discussed in section *Creating a list of Product Occurrences*.

```
using System;

using System.Collections.Generic;

using Aras.DynamicModelViewer.DataModel; // Required for Product
Occurrences

using Aras.Server.Core.QueryBuilder; // Required to Process Query
results

using System.Drawing; // Required for Colors used in Rendering
Configurations

namespace CustomDefaultQP
{
    /// <summary>
    /// Main Class for processing the results from the execution of the
    default
    /// Query Definition used for Dynamic Visualization
    /// </summary>
```



```

public class DefaultQueryResultsParser
{
    /// <summary>
    /// Default Constructor
    /// </summary>
    public DefaultQueryResultsParser() { }

    /// <summary>
    /// Processes the given Query Result Items to build a hierarch of CAD
    Items
    /// </summary>
    /// <param name="resultItems">Input set of top-level Query
    Results</param>
    /// <returns>List of Product Occurrences</returns>
    public ICollection<ProductOccurrenceBase>
    processQueryResults(IEnumerable<QueryBuilderNode> resultItems)
    {
        // Process each root CAD Item in the query results
        List<CAD> rootCADItems = new List<CAD>();
        foreach (var resultItem in resultItems)
        {

```



```

if (resultItem.QueryItem.Alias == "CAD")
{
CAD nextCAD = new CAD();
nextCAD.processCAD(resultItem); // recurses through full hierarchy
rootCADItems.Add(nextCAD);
} // if()
} // foreach()

// Build the Product Occurrences for each root CAD Item
ICollection<ProductOccurrenceBase> poList = new
List<ProductOccurrenceBase>();
foreach (CAD c in rootCADItems)
{
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.Name = c.Name;

if (c.Children.Count == 0 && c.SCFileID != null) // Assume this is a
component part

```



```

{
    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 - no view file attached
else
{
    poInst.SetAsRoot(); // root assembly,
    poInst.SetServiceProperty(c.QryRefId, c.ID); // add service properties
    poList.Add(poInst); // add to the PO list
}

// Recurse through child hierarchy
foreach (CAD child in c.Children)
    build(poInst, child);
} // foreach (CAD c in rootCADItems)
return poList;

```



```

} // processQueryResults()

/// <summary>
/// Constructs a list of Product Occurrences (PO) from the CAD Items
parsed
/// from the Query Results. This will construct a unique Product
Occurrence
/// for each instance of a part.
/// </summary>
/// <param name="parent"> Product Occurrence to attach newly created POs
to.
/// This value can't be null</param>
/// <param name="c">CAD Item representing the child CAD data to build
from</param>
private void build(ProductOccurrenceInstance parent, CAD c)
{
// Create a Product Occurrence for each instance of the given CAD
foreach (CADInstanceInfo cadInstInfo 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));
```

```
// bounding box
```

```
poInst.ProductOccurrenceSource.BoundingBox.Max.X = c.BBox.MaxX;
```

```
poInst.ProductOccurrenceSource.BoundingBox.Max.Y = c.BBox.MaxY;
```

```
poInst.ProductOccurrenceSource.BoundingBox.Max.Z = c.BBox.MaxZ;
```

```
poInst.ProductOccurrenceSource.BoundingBox.Min.X = c.BBox.MinX;
```

```
poInst.ProductOccurrenceSource.BoundingBox.Min.Y = c.BBox.MinY;
```

```
poInst.ProductOccurrenceSource.BoundingBox.Min.Z = c.BBox.MinZ;
```

```
// *** Testing Rendering Configuration ***
```

```
ProductOccurrenceRenderingConfiguration rConfig = new  
ProductOccurrenceRenderingConfiguration();
```

```
rConfig.SetColor(Color.FromArgb(55, 40, 0));
```

```
rConfig.Opacity = 0.4;
```

```
rConfig.Name = "test";
```

```
poInst.RenderingConfigurations.Add(rConfig);
```

```
// *** Testing Rendering Configuration ***
```



```

// **** ADD SetServiceProperty METHOD CALLS ****
poInst.TransformationMatrix = cadInstInfo.XForm;
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 part
{
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 - no view file attached
else
parent.Children.Add(poInst);

```



```

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

} // build()

} // class DefaultQueryResultsParser
} // namespace CustomDefaultQP

```

The following `CAD` class is used to collect information from the Query Results. It contains the `processCAD` method discussed in section *Processing Query Results*.

```

using System;
using System.Collections.Generic;
using Aras.Server.Core.QueryBuilder;

namespace CustomDefaultQP
{
struct BoundingBox
{
public double MinX; // X Coord for minimum point

```



```
public double MaxX; // X Coord for maximum point
public double MinY; // Y Coord for minimum point
public double MaxY; // Y Coord for maximum point
public double MinZ; // Z Coord for minimum point
public double MaxZ; // Z Coord for maximum point
} // class BoundingBox
```

```
/// <summary>
```

```
/// Stores information for each CAD Instance Query Item
```

```
/// </summary>
```

```
class CADInstanceInfo
```

```
{
```

```
public CADInstanceInfo()
```

```
{
```

```
XForm = "1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1"; // Default Identity Matrix
```

```
ID = "0";
```

```
}
```

```
/// <summary>
```

```
/// CAD Instance transformation matrix
```



```
/// </summary>
```

```
internal String XForm { get; set; }
```

```
/// <summary>
```

```
/// CAD Instance Id
```

```
/// </summary>
```

```
internal String ID { get; set; }
```

```
/// <summary>
```

```
/// CAD Query Reference Id
```

```
/// </summary>
```

```
internal String RefID { get; set; }
```

```
} // class CADInstanceInfo
```

```
/// <summary>
```

```
/// Stores all data necessary for creating 3D View data assuming
```

```
/// it is contained within 'CAD' Query Items
```

```
/// </summary>
```



```

class CAD
{
private BoundingBox _BBox; // Bounding volume

/// <summary>
/// Default Constructor
/// </summary>
internal CAD()
{
// Ensure that at least one instance is added
// This will be replaced in _processCADStructure() if instances
// are included in the query results
Instances.Add(new CADInstanceInfo());
} // CAD()

/// <summary>
/// Processes a given CAD Item contained within the given
<code>QueryBuilderNode</code>
/// </summary>
/// <param name="qbItem">Item containing properties for CAD
Items</param>

```



```

/// <exception cref="ArgumentException">Required Properties not included
in results</exception>
internal void processCAD(QueryBuilderNode qbItem)
{
    QryRefId = qbItem.QueryItem.RefId;
    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(QueryBuilderInterface qbItem)

/// <summary>

```



```

/// Processes the bounding box data in the given CAD ConversionInfo node
/// </summary>
/// <param name="qbItem"></param>
private void _processCADConversionInfo(QueryBuilderNode qbItem)
{
    _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);
} // _processCADConversionInfo(QueryBuilderNode qbItem)

/// <summary>
/// Returns the value of the given Property as a double, use given
alternate
/// if the value is not set or is not defined.
/// </summary>
/// <param name="qbItem">Query Builder Item containing the
Property</param>
/// <param name="propName">Property Name</param>

```



```

/// <param name="alt">Used when given property not set</param>
/// <returns>Value if Property exists and is set, 'alt' value
otherwise</returns>

private double  getPropertyAsDouble(QueryBuilderNode qbItem, String
propName, double alt)
{
// check for existence of Property (in future)

if (qbItem.TryGetProperty(propName, out string tmpPropVal) && tmpPropVal
!= null)
return Double.Parse(tmpPropVal);
else
return alt;
} // getPropertyAsDouble()

/// <summary>
/// Processes the child CAD Instances and Files associated with the
given
/// <code>QueryBuilderItem</code>
/// </summary>
/// <param name="qbItem"></param>
/// <exception cref="ArgumentException">Required Properties not included
in results</exception>

```



```

private void _processCADStructure(QueryBuilderNode qbItem)
{
    // Save the instances and store with child CAD Item
    List<CADInstanceInfo > curInstances = new List<CADInstanceInfo>(); CAD
nextCAD = new CAD();
    nextCAD.CADStructureID = qbItem.ItemId;
    nextCAD.CADStrQryRefId = qbItem.QueryItem.RefId;
    // Loop through all CAD Instances and CAD Items
    foreach (var child in qbItem.ChildNodes)
    {
        try
        {
            if (child.QueryItem.Alias == "CAD Instance" &&
child.TryGetProperty("transformation_matrix", out string tmpXForm) &&
tmpXForm != null) {
                CADInstanceInfo cadInfo = new CADInstanceInfo();
                cadInfo.XForm = tmpXForm;
                cadInfo.ID = child.ItemId;
            }
        }
    }
}

```



```

cadInfo.RefID = child.QueryItem.RefId;
curInstances.Add(cadInfo);
}
if (child.QueryItem.Alias == "CAD")
{
nextCAD.processCAD(child);
Children.Add(nextCAD);
}
} // try
catch { }
} // foreach()
if (curInstances.Count > 0)
nextCAD.Instances = curInstances;
} // _processCADStructure(QueryBuilderNode qbItem)

/// <summary>
/// Returns the first found <code>QueryBuilderNode</code> with the
/// given alias that is a descendant of the given Item. Note that if
/// the given Item has the alias, it is returned
/// </summary>

```



```

/// <param name="qbItem">Item containing descendants to search</param>
/// <param name="alias">Alias Name of Item to search for</param>
/// <returns>NULL, if not found; first found node with given alias
otherwise</returns>

private QueryBuilderNode _findChildWithAlias(QueryBuilderNode qbItem,
String alias)
{
if (qbItem.QueryItem.Alias == alias)
return (qbItem);
else
{
foreach (var child in qbItem.ChildNodes)
{
QueryBuilderNode descItem = _findChildWithAlias(child, alias);
if (descItem != null)
return (descItem);
} // for()
} // else
return (null);
} // findChildWithAlias(QueryBuilderNode qbItem, String alias)

```



```

/// <summary>
/// Extracts the SCS file data from the given
<code>QueryBuilderNode</code>. This
/// method assumes that the Properties for 'id' and 'filename' exist
/// in the given results node with alias 'File_1'
/// </summary>
/// <param name="qbItem"></param>
/// <exception cref="ArgumentException">Required File Properties not
included in results</exception>
private void _extractFileInfo(QueryBuilderNode qbItem)
{
QueryBuilderNode child = _findChildWithAlias(qbItem, "File_1");
if (child != null)
{
// filename and id are required for Product Occurrences
if (child.TryGetProperty("filename", out string tmpFileName) &&
tmpFileName != null &&
child.TryGetProperty("id", out string tmpId) &&
tmpId != null)
{
SCFile = tmpFileName;
}
}
}

```



```

SCFileID = tmpId;
}
else
throw new ArgumentException("Query Definition is not defined properly:
'filename' and 'id' Properties are required for view files");
} // if (child != null)
} // _extractFileInfo(QueryBuilderItem qbItem)

/// <summary>
/// Returns the name of the CAD Query Response Item
/// </summary>
internal String Name { get; private set; }

/// <summary>
/// Returns the Query Reference ID from the CAD Query Response Item
/// used to create this CAD Object
/// </summary>
internal String QryRefId { get; private set; }

```



```
/// <summary>
/// Returns the Query Reference ID from the CAD Structure Query
/// Response Item used to create this CAD Object
/// </summary>
internal String CADStrQryRefId { get; private set; }
```

```
/// <summary>
/// Returns the Id of the CAD Query Response Item
/// </summary>
internal String ID { get; private set; }
```

```
/// <summary>
/// Returns the Id of the CAD Query Response Item
/// </summary>
internal String CADStructureID { get; private set; }
```

```
/// <summary>
```



```
/// Returns the SCS/SCZ File name associated with the CAD Query Response
Item
/// </summary>
internal String SCFile { get; private set; }

/// <summary>
/// Returns the SCS/SCZ File ID associated with the CAD Query Response
Item
/// </summary>
internal String SCFileID { get; private set; }

/// <summary>
/// Returns the list of data for CAD Instance Query Response Items
/// </summary>
internal List<CADInstanceInfo> Instances { get; private set;} = new
List<CADInstanceInfo>();

/// <summary>
/// List of child CAD Items as determined by the CAD Structure
/// </summary>
internal List<CAD> Children { get; } = new List<CAD>();
```



```
/// <summary>
```

```
/// Bounding Volume
```

```
/// </summary>
```

```
internal BoundingBox BBox { get { return _BBox; } }
```

```
} // class CAD
```

```
} // namespace CustomDefaultQP
```

