Aras 3D Visualization 12.0R3

Administrator Guide

Document #: 12.0R3.02021043001

Last Modified: *08/02/2021*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810

Phone: 978-806-9400
Fax: 978-794-9826

E-mail: Support@aras.com

Website: https://www.aras.com

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

Email:
Support@aras.com
Subject: Aras Innovator Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Innovator Documentation

Or,

FAX:
978-794-9826
Attn: Aras Innovator Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support/

Document Conventions

The following table highlights the document conventions used in the document:

Table 1: Document Conventions

Convention	Description
Bold	This shows the names of menu items, dialog boxes, dialog box elements, and commands. Example: Click **OK**.
`Code`	Code examples appear in `courier` font. It may represent text you type or data you read.
`Yellow highlight`	Code highlighted in yellow draws attention to the code that is being indicated in the content.
`Yellow highlight with red text`	Red text highlighted in yellow indicates the code parameter that needs to be changed or replaced.
Italics	Reference to other documents.
Note:	Notes contain additional useful information.
Warning	Warnings contain important information. Pay special attention to information highlighted this way.
Successive menu choices	Successive menu choices may appear with a greater than sign (-->) between the items that you will select consecutively. Example: Navigate to **File** --> **Save** --> **OK**.

1 Overview

This document describes the features of 3D Visualization, including CAD Conversion and the Dynamic Viewer overarching concept and vision. It also explains how to configure this capability in Aras Innovator.

1.1 Objective

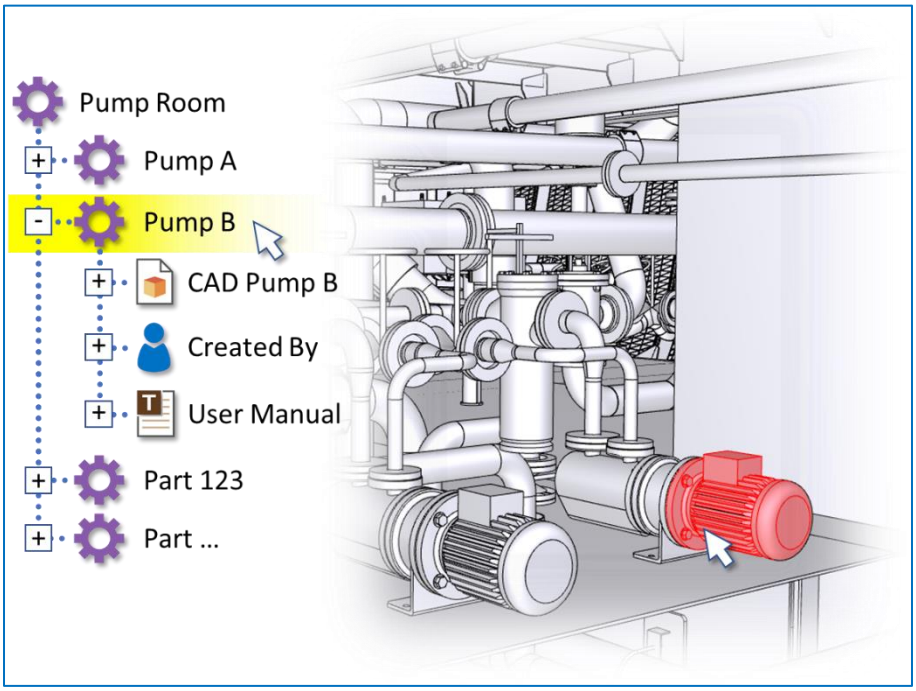

Figure 1. Dynamic Product Navigation Concept

There are several enhancements planned for the 3D Visualization capability that will greatly expand on the utility and configurability of this feature. Aras Innovator is a framework, and 3D Visualization is a tool within it. The *goal* is to be able to apply or use 3D visualization throughout the various phases of a Product's lifecycle. The *approach* is to provide a level of configurability that enables end-user customizations so that 3D Visualization can be applied based on a customer's business need. These enhancements include the ability to identify what 3D content is displayed, how it is rendered, and what related content is included. Collectively these enhancements are known as 'Dynamic Product Navigation' or DPN. This section provides an overview of the suite of enhancements that will be rolled out across subsequent releases of Aras Innovator.

1.1.1 Enabling Control of *What* Content is Displayed

In Aras Innovator 11.0, Query Builder and Tree Grid View were added. These features provided a graphic editor that allowed an end-user to define an ItemType query and bind the elements of that query to a Tree Grid View. The concept is similar to a Model and View respectively in a Model/View/Control paradigm typically used as a design guide in software engineering. The vision was to use these queries – known as Query Definitions – as a reusable data structure in such a way that other types of views could be constructed from them. Graph Navigation Views were added in a subsequent Aras Innovator release. Query Definitions and their included conditional logic, are used with DPN to identify the elements to be rendered in a 3D View, thus providing a convenient mechanism for users to control what elements are shown in the 3D Viewer.

1.1.2 Enabling Control of *How* Content is Displayed

Query Definitions define the ‘’what.” Tree Grid View and Dynamic View Definitions describe *how* to visualize the results when a Query Definition is executed. Tree Grid View Definitions map elements of a Query Definition to Tree Nodes and columns. These elements are the components that define a Tree Grid View. Thus, a specific Query Definition is *executed* to identify Query Results which are then visualized as a Tree Grid View based on the mappings in a Tree Grid View Definition (see Figure 8). DPN uses both Query Definitions and Tree Grid Views as described in Section 4. 3D Views can provide additional visual information simply based on color and transparency. That is, render the 3D geometry using a particular color to emphasize (or de-emphasize) when viewing with other parts. Likewise, the level of transparency can be used for a similar effect. Dynamic View Definitions allow users to identify a Data Processor which uses a Query Processing API to implement customization points that allow end-users to adjust/define 3D color and transparency used in a 3D View.

1.1.3 3D Visualization as a Navigation Tool

The phrase ‘Dynamic Product Navigation’ was chosen to reflect the fact that with the proposed functionality:

- 3D Views are generated dynamically as the product of a defined Query.
- Related content can be visualized together with 3D components.

3D Visualization can be used as a navigation tool whereby users select 3D components in a view and see related Business Objects (related Items), with the ability to open these Items directly from the 3D View.

1.1.4 Alternate Rendering Architectures

3D geometry for assemblies can consist of many files, and these files can be large in size. Rendering this information requires that this data be downloaded to the client web browser. Starting in 12.0 SP7, the ability to configure and manage alternative, server-side processing to streamline the rendering of 3D content and improve performance and the user experience was added (see Section 6).

1.1.5 Use 3D Visualization to Generate Custom Graphics

The ability to customize the content and display of 3D geometry provides a capability that can be extended by allowing users to reuse the resulting graphics for other purposes, for example: technical documentation. In addition, there will be features that will allow end-users to customize the position and orientation of select 3D geometry to create custom exploded views that help describe the assembly and composition of a complex part. This capability can be used in manufacturing process plans, for example.

1.1.6 Spatial Search and Analysis

Spatial search provides the ability to identify components in complex assemblies (e.g. Aircraft) based on the selection of an instance of a part/assembly or the identification of a 3D volume. Spatial analysis provides the ability to determine whether any of the instances discovered in a spatial search interfere geometrically, physically touch one another, or exist within the proximity of one another based on some provided distance threshold. This capability is useful when evaluating the juxtaposition of assemblies that are defined and that include disparate parts of an organization.

1.2 Release Approach

DPN will be developed and released using a phased approach. Functionality will be implemented/ enhanced incrementally across several releases. This approach allows for functionality to be provided sooner; allowing users to provide feedback on future enhancements and/or changes. The initial capability, released with Aras Innovator 12.0 SP2, includes the ability to define custom Query Definitions, along with associated Tree Grid View Definitions giving end-user administrators the ability to define dynamic views. In addition, conversion logic will be included to identify Part Instances and the transformations for positioning them dynamically when rendering a 3D view. Thus, the core capability will be included to allow a user the ability to define what content is displayed with some ability to define how it is displayed. This document describes this functionality.

1.3 Known Limitations

Initial versions of DPN will not have all the proposed functionality. It may not necessarily have all the functions/features of the monolithic viewer (see Section 3.2.4). This section describes the known limitations so end-users have the information they need to decide whether or not to adopt (install) this new capability now.

1.3.1 View Features Limitations

The monolithic 3D Viewer includes several ‘view functions’ accessible either using context menus in the Model Browser or by selecting 3D component geometry in the view. The initial implementation of the dynamic viewer does not include these view functions. They will be added across multiple future releases.

1.3.2 CAD / CAD Structure

Out of the box, Aras Innovator uses the CAD and CAD Structure relationship ItemTypes to capture the Bill of Materials (BOM) information for a mechanical Assembly. For Dynamic Visualization, part instance and transformation data are required and for Aras Innovator 12.0 SP2 this information is assumed to exist in `CAD Instance` Items. The software logic in the CAD Converter creates CAD Instance Items with the appropriate transformation information (see Section 5.1.1). Starting with 12.0 SP7, the ability to support alternate data models exists by implementation of a custom Query Processor (see Section 6).

2 Installation and Configuration

2.1 CAD Conversion

2.1.1 Install CAD Converter

To set up CAD Conversion, refer to the *Aras 3D Visualization 12.0R2 – Installation Guide*, available in the Documentation folder of the 3D Visualization CD Image, which can be obtained by Subscribers from the Aras FTP site in the `Aras 3D Visualization` folder.

2.1.2 Using Legacy View Files

The most recent version of the 3D Viewing and Translation software does not require users to convert their existing HWF view files. Aras PLM provides backward compatibility in both the data model and viewing features. There is a new viewer in HOOPS Communicator 2017 that gives users the ability to convert legacy HWF View files to the new Stream Cache Single (SCS) file format using an automated, asynchronous process. CAD items can either use the legacy HWF format or the SCS format.

2.1.3 Preference Settings

Users can choose to always display legacy HWF files instead of converting them to SCS by selecting the **Use Legacy 3D View Files** checkbox on the Secure Social tab.

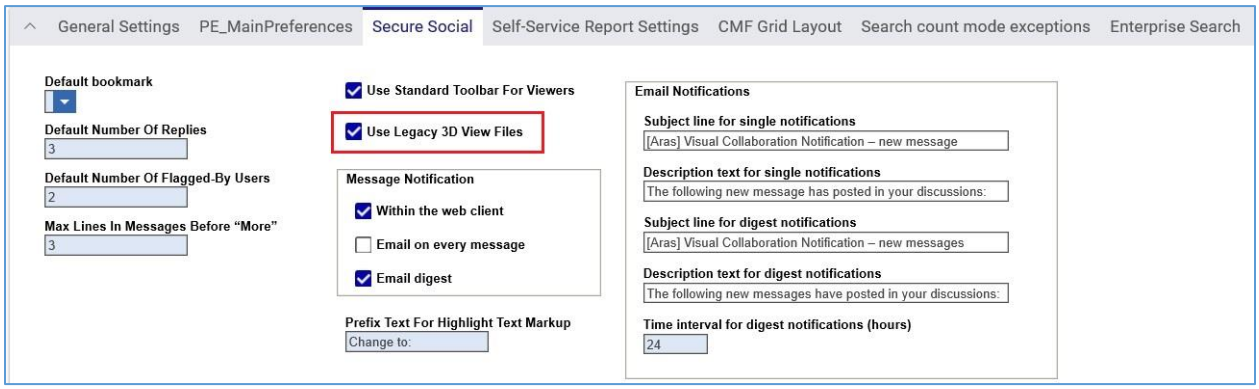

Figure 2.

The process for viewing and converting files are based on the following:

- The value of the Preference setting.
- The existence of a PRC file.
- The existence of an SCS file.

PRC files are created by default as part of the standard conversion server settings.

2.1.4 Converting Legacy View Files

The conversion process for legacy view files begins when a user asks to view a CAD document or a Part with a related CAD document that includes a legacy view file (HWF). The following figure describes the process.

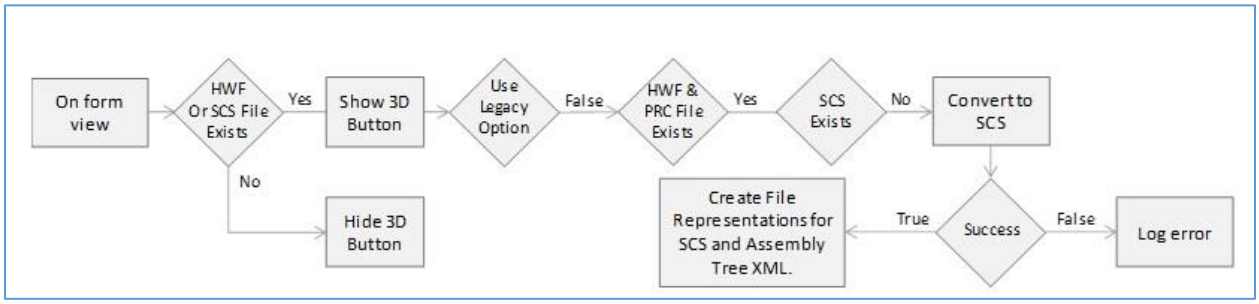

Figure 3.

The existence of an HWF or SCS file determines whether the 3D Viewer can be opened. If it can, a button appears in the sidebar enabling users to open the viewer. If either of these files do not exist, the user is unable to open the 3D Viewer. The value of the preference setting and the existence of an SCS file determines whether it is necessary to convert the file. If the following conditions exist, an ad hoc, asynchronous conversion process starts:

- The Preference setting is False.
- HWF and PRC files exist.
- An SCS file has not been created previously.

A successful conversion process results in a new file representation that points to the generated SCS and Assembly Tree files. Conversion process errors should be logged. Subsequent attempts to open the same CAD document after a successful conversion result in viewing the generated SCS file.

2.2 Configuring Dynamic Viewer Installation

The installation process creates a default configuration for Dynamic Viewer. For installation details, refer to the *Aras 3D Visualization – Installation Guide*.

The following section explains the command arguments. Any outputs not required by your implementation can be removed from the command arguments.

2.2.1 Out of the Box Setup

When Aras 3D Visualization is installed, the following OOTB setup is used in `ConversionServerConfig.xml` file.

```
<AssemblyCommand dynamicEnabled="True" arguments="--sc_compute_bounding_boxes 'All'
--input_pdf_template_file 'C:\HOOPS Converter\templates\Blank_Template_L.pdf' --
output_pdf '%filepath%\%filename%.pdf' --output_png '%filepath%\%filename%.png' --
output_png_resolution '150x150' --output_scs '%filepath%\%filename%.scs' --
output_xml_assemblytree '%filepath%\%filename%.xml' --output_prc
'%filepath%\%filename%.prc' --background_color '1.0, 1.0, 1.0' --output_logfile
'%filepath%\%filename%.log'" />
```

2.2.2 Customization Options

The following table describes the command arguments that should be used. Any outputs not required by your implementation can be removed from the command arguments.

Table 2: Conversion Command-line Arguments

Command-line Argument	Required	Description
--sc_compute_bounding_boxes	True	Used to position the camera in the viewer and prioritize the rendering order of an assembly. The bounding box information will be extracted from the native CAD file.
--input_pdf_template_file	False	Required only if generating PDF viewable files for the Item.
--output_pdf	False	Required only if generating PDF viewable files for the Item.
--output_png	True	Used to produce a thumbnail image for the Item.
--output_scs	True	Enables the monolithic and dynamic viewers.
--output_xml_assemblytree	True	Required to map 3D component geometry.
--output_prc	False	Required only if generating PRC files for the Item, which may be used for industry standard archival purposes.
--background_color	True	Used for the background color for thumbnail images (png) and the 3D PDF. Default is black. However, a background color other than white may affect the display of the image in the Item Form.
--output_logfile	True	If provided, the name of the log file where the HOOPS Converter will write error and warning messages.

Please see the included documentation for a complete description of available parameters and their functions: `\HOOPS_Converter\Documentation\online_docs.html`

3 Dynamic Viewer

This section describes the Dynamic Viewer and compares features with the Monolithic Viewer. It also provides an overview of the assumed ItemType data model, from which the data required by Dynamic Viewer is queried.

3.1 CAD Data Model

Before reviewing the Dynamic Viewer features, it's important to understand the CAD ItemType data model so it is clear what data is required to support Dynamic Visualization and where this information is stored and retrieved by default. This section describes the process of creating CAD Items, their related ItemTypes, and CAD File conversions using the following diagram.

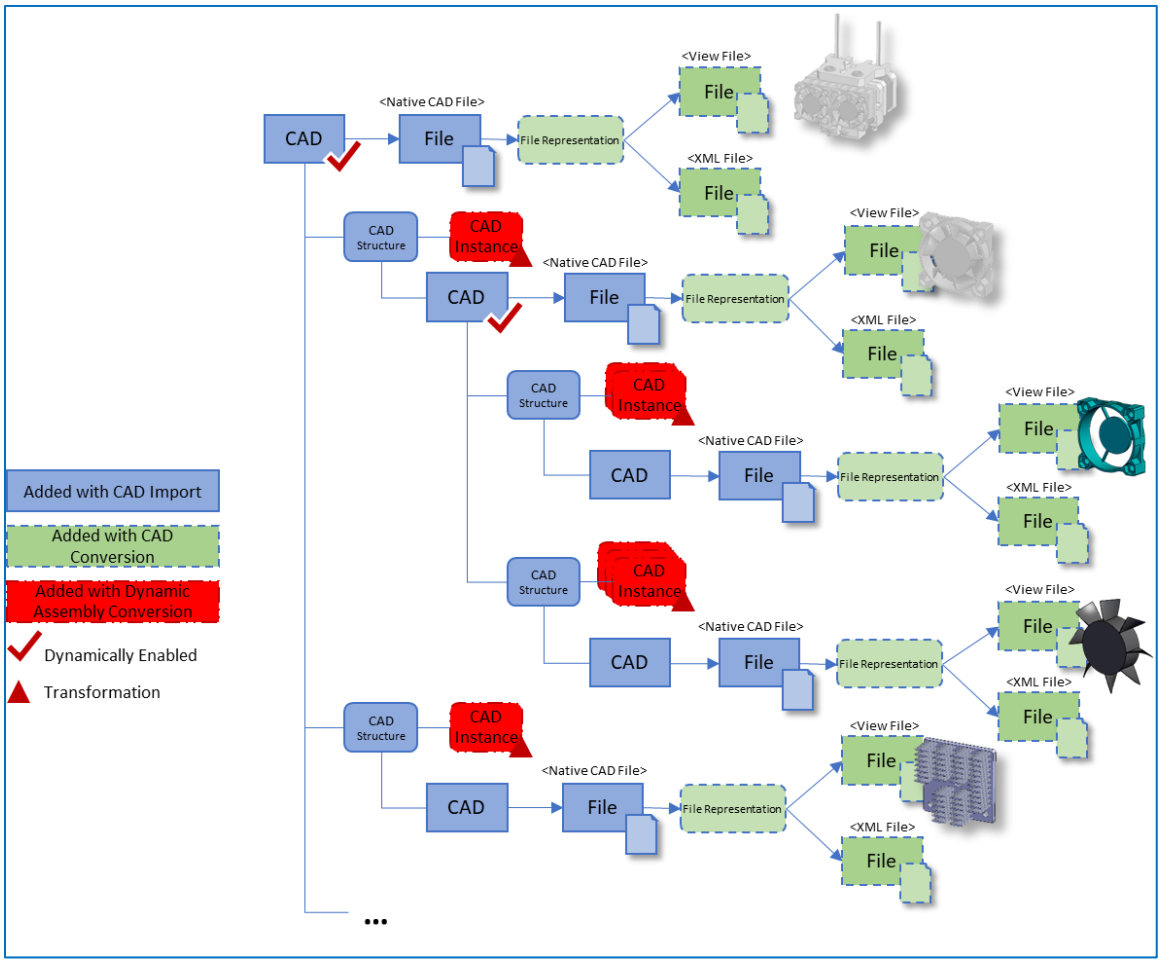

Figure 4. CAD Item Creation and Conversion

CAD Items and CAD Structure Relationships are typically created by a CAD Connector when CAD native files are 'checked in' to the PLM System. A CAD Item represents a single CAD 3D Assembly, Part, or 2D drawing. For this document, CAD Items representing 3D CAD are assumed. In this case, the native CAD file is attached (via a File ItemType property) to each CAD Item. If this CAD item represents an Assembly, all related sub-assemblies and/or parts will be included in separate CAD Items and referenced via CAD Structure Items. The CAD and CAD Structure Items represent a mechanical BOM.

CAD Conversion – the process of generating alternate versions of native CAD data – is triggered by the existence of a CAD file.1 The Monolithic and Dynamic Viewers render 3D component geometry stored in a Stream Cache Single (SCS) 'view file'. Note that thumbnail images, PRC, 3D PDF, and other converted formats can also be created as part of the CAD Conversion process and stored with the CAD Item or in related Items. Beginning with Aras Innovator 11.0 SP11, an XML file was also created and stored. This XML file is generated from the conversion process and used to identify all the Instances of parts/sub-assemblies that were included with a CAD Assembly. This information is used to map 3D component geometry in the Monolithic Viewer and generate CAD Instances and 3D Transformations for the Dynamic Viewer. The view file (SCS) and the XML are stored as related Items attached to the File Item for the native file. Note that Figure 4 uses a single Relationship graphic labeled 'File Representation' to simplify the diagram. However, File Representations use two `RelationshipType`s as shown in Figure 5.

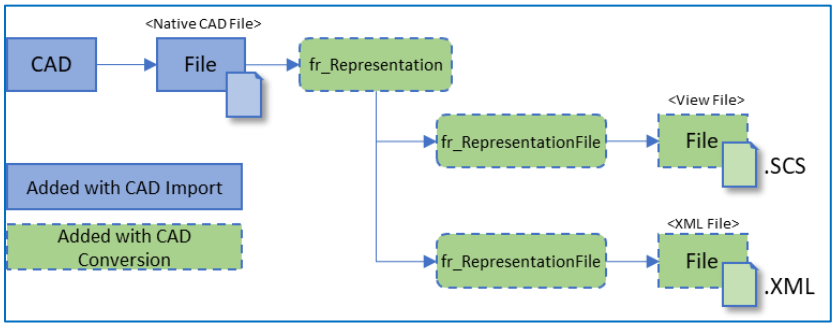

Figure 5. File Representations

For the Dynamic Viewer, the conversion process appends CAD Instance Items for all instances of an associated (related) CAD Item. CAD Instances contain a property that stores the 3D Transformation information as a 4X4 matrix. In addition, an added Boolean property – **Dynamic Enabled** – is set to true for a CAD Assembly Item when all data necessary to render that CAD assembly using the Dynamic Viewer has been created. Note that these Instances and their transformation information are created automatically by the 3D Conversion Process based on information extracted from the native CAD Assembly File.

3.2 Monolithic vs Dynamic Viewer

The best way to describe the Dynamic Viewer is to compare it to the Monolithic Viewer. This section compares the existing features of the Monolithic Viewer with the new features of the Dynamic Viewer. For reference, the following icons on the sidebar of a CAD ItemType Form open each of the viewers:

1 Aras Innovator CAD Conversion supports CAD files from multiple CAD systems. These are configured using Conversion Rules – see the CAD to PDF Converter Setup Guide.

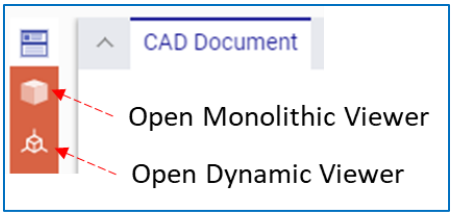

Figure 6. Monolithic / Dynamic Viewer Icons

3.2.1 Process Overview – Monolithic Viewer

The Monolithic Viewer displays a single view file (SCS); which is the view file that is attached to the corresponding CAD Item (see Figure 5). If it is an Assembly view file, then the 3D component geometry for all sub-assemblies and/or parts included in that Assembly at the time the CAD file was checked in are included in the monolithic view file. In essence, it's a static view of the assembly because it won't include any geometry changes to sub-components made after the assembly view file was created. The process to render the view file in the Monolithic viewer is as follows:

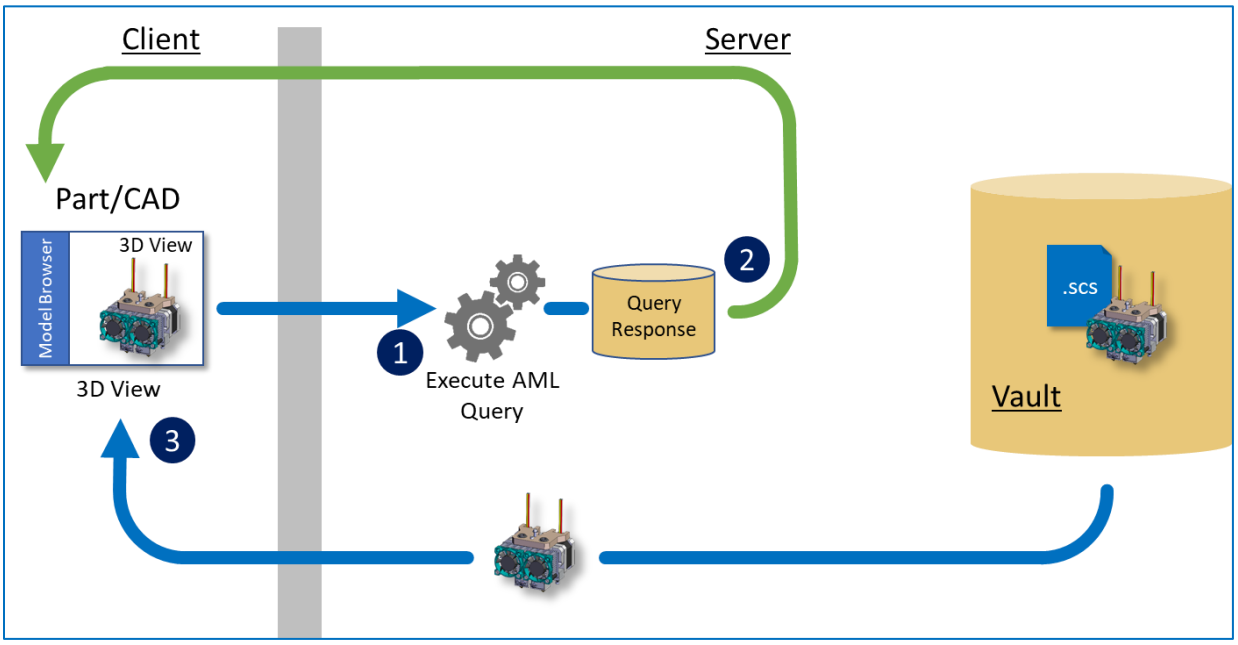

Figure 7. Monolithic Viewer Rendering Process

When the Monolithic Viewer is opened, a static AML query is executed to retrieve the CAD Structure of the opened CAD Item. The results are displayed in a Tree User Interface called the 'Model Browser' of the Viewer. A request is made to load the single SCS view file associated with the opened CAD Item, the 3D component geometry of which is displayed in the viewer. The XML data generated from the conversion process is also retrieved and the information contained within it is used to map the Items displayed in the Tree View with the associated 3D components in the view. This provides selection synchronization between the Model Browser and the 3D View.

3.2.2 Process Overview – Dynamic Viewer

The Dynamic Viewer is used for assemblies only and displays view files (SCS) as determined by the results returned from the execution of a Query Definition. Unlike a monolithic view file for an Assembly, the query results for a dynamic query target the view files for component parts and the instance and transformation data required to properly position each file given the returned assembly hierarchy. The process is described as follows:

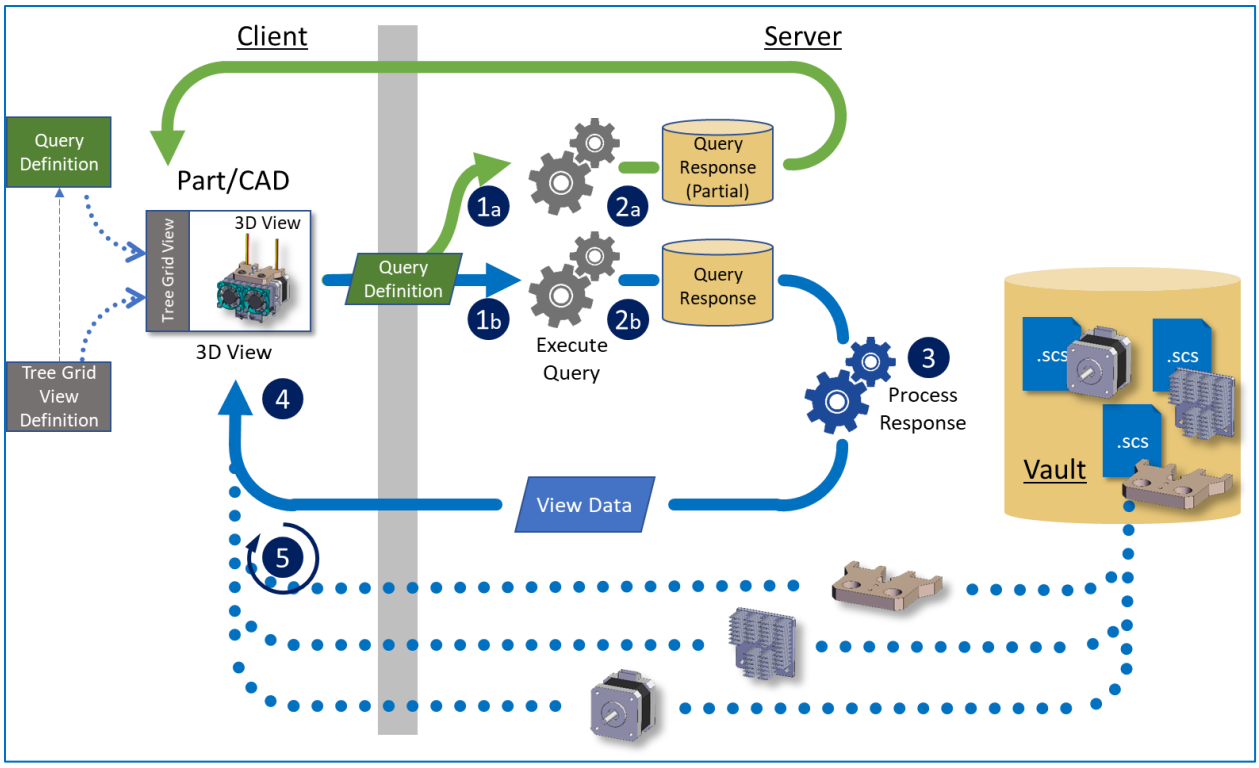

Figure 8. Dynamic Viewer Rendering Process

The Dynamic Viewer consists of two main components: Tree Grid View and the 3D View. The Tree Grid View displays the results of the associated Query Definition as defined by the chosen Tree Grid View Definition (see section 5.2.2). The Tree Grid View is a composite of a Tree View and a Table (or Grid). The left-most column contains a hierarchical Tree View showing all the related contents starting from (rooted by) the CAD or Part Item from which the Dynamic View was opened.

Upon refreshing the view, the system executes the associated Query Definition in two simultaneous operations. The first executes the query to populate the Tree Grid View. The second executes the query and processes the complete response.

The partial response is used by the Tree Grid View based on the configuration of the associated Tree Grid View Definition.2 . Note that by default, the Tree Grid View is 'lazy loaded'. That is, only a portion of the response is returned to the client and displayed. To generate the view data however, the full response is processed so that all component parts are identified.

2 Tree Grid View Definitions provide settings that are used to determine the maximum number of peer elements and depth of related content. These settings can be overridden using the Tree Grid View toolbar.

While processing the query results for the 3D View, XML data is constructed that identifies the assemblies, parts, part instances and their transformations for each. Parts will have links to the respective View File in the Vault.

The View Data (XML) is processed by the 3D Viewer and subsequent requests are made to the Aras Innovator Vault to retrieve each of the corresponding View files to render.

View Files are processed individually and rendered in sequence in the 3D Viewer.

3.2.3 Tree Grid View vs Model Browser

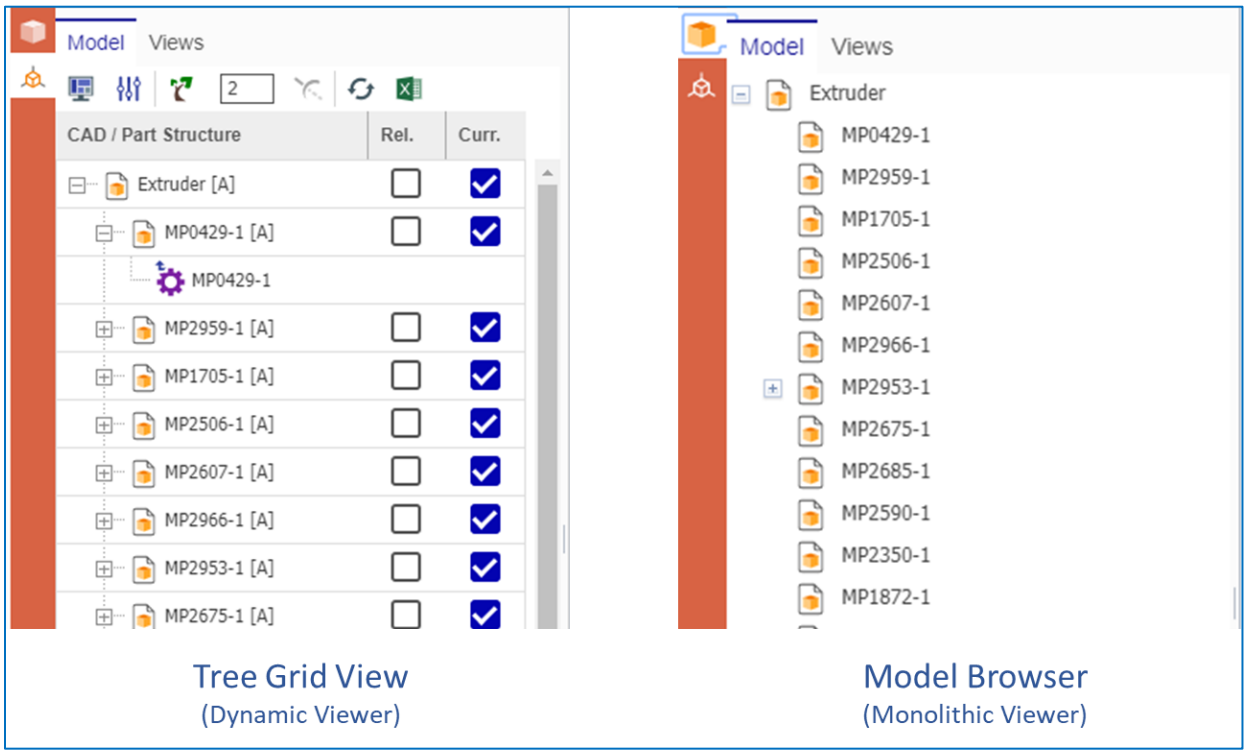

Figure 9. Tree Grid View vs Model Browser

The concept and vision for Dynamic Product Navigation (see Section 1) requires an alternate mechanism for displaying the business objects (ItemTypes) associated with the 3D components displayed in the 3D View. The 'dynamic' element of DPN specifies that the contents of a 3D view are determined by some conditional (and customizable) logic. The Query Definition provides that ability. The 'navigation' element of DPN specifies that the 3D View can be used as a means to access *related* business Objects and the Tree Grid View Definition provides that ability. Thus, the Tree Grid View (the user interface component that displays Items and associated properties in a grid) was the optimal choice for the Dynamic Viewer. This section compares the features of the Tree Grid View with the Model Browser so that it's clear how each functions and what information they each provide.

3.2.3.1 Query Definition vs AML

The Tree Grid View is populated with the response generated from the execution of an associated Query Definition. In addition to the CAD Items that form the basis of the query, administrators can add other related ItemTypes the properties of which are displayed in the user interface. The Tree Grid View Definitions give administrators the ability to choose which Items to display, what Properties to include, how to format the Property values, static text, what columns to include, and alternative icons. For example, Figure 9 shows the Tree column (left-most) with nodes for the CAD Item which displays the associated CAD Item Name and its revision in square brackets as well as associated Part Items which display the Part Item Name. In addition, two columns were added: one for the '`is_released`' Property and one for the '`is_current`' Property; both are displayed as Boolean values (checkboxes).

In contrast, the Model Browser content is determined by the execution of a static (hard-coded) AML query on the CAD Structure of the opened CAD or Part Item. Each node in the Tree Grid View uses the '`keyed_name`' of the CAD Item with the icon for the CAD ItemType. Other than the configuration of the keyed_name Property, there are no other ways to configure the display.

3.2.3.2 Partial vs Full CAD Structure display

By Default, the Tree Grid View is populated using only a portion of the results of the Query Definition response. When there are multiple levels of related content, the user determines which content to query based on selected nodes in the Tree column. Expanding the node results in the re-execution of the query starting at that node and the Tree Grid View is updated with the added rows. The Model Browser uses AML to execute an exhaustive (fully-recursive) search for the entire CAD BOM structure. The Tree Grid View is then populated with the full list of CAD Items from the query.

There is a trade-off with each approach. For the Tree Grid View, the execution of the query is generally faster (much faster if the results are large). However, after the Model Browser is populated, all the data exists on the client and can be readily viewed. This affects synchronization between the data displayed in the Model Browser/Tree Grid View and the 3D View. For the Dynamic Viewer, when a 3D component is selected in the 3D View, a directed set of 'sub-queries' is performed to populate the Tree Grid View with all levels of the hierarchy up to the selected part component (see Figure 35). Once the data is populated, the query does not need to be re-executed, but initial population can take several seconds.

In general, and especially for large/deep assembly hierarchies, the lazy-loading approach used by the Tree Grid View will yield a better user experience and require less network data exchange and less memory for the client browser. Also, the number of peer nodes and depth of query responses is configurable in the Tree Grid View Definition so administrators have control over the granularity of query response data.

3.2.4 View Function Comparison

The Monolithic and Dynamic 3D Viewers use the same technology – HOOPS Communicator. The main difference is how the view is populated. However, in 12.0 SP2 there are also limitations associated with the available view functions. These limitations are explained in this section.

3.2.4.1 Fit All and Isolate Functions

The **Fit All** and **Isolate** functions are not currently available in the Dynamic Viewer. For now, refreshing the view performs a similar function.

3.2.4.2 Reset View, Display All, Hide, Hide All Functions

The **Reset View** and **Display All** functions exist because of the ability to Hide 3D components in the Monolithic Viewer. Future functionality in the Dynamic Viewer (and Tree Grid View) will emulate the Monolithic Viewer in its ability to Hide/Show selected 3D Components either from context menus in the 3D View or by icon/buttons in the Tree Grid View. The latter will require logic and User Interface changes in the Tree Grid View as to how associated 3D Components are *marked* as being either Hidden or Shown given that only a portion of the associated Tree Grid View nodes may be displayed (see Section 3.2.3.2).

3.2.4.3 Product Manufacturing Information (PMI)

PMI information is typically stored at the assembly level within CAD native files. When dynamically generating assemblies – such as the logic used for the Dynamic Viewer – only view files for the component parts are included. In doing so, only PMI information that has been stored with the native file and included with the converted data in the view file[3] can be viewed in the 3D View. Note that there is no guarantee that graphic PMI information will be positioned such that it doesn't interfere with other rendered 3D component geometry.

3.2.4.4 Configurations

The Dynamic Viewer is only accessible for CAD Assemblies. Similar to the display of PMI information, Configuration data stored with an Assembly view file is not accessible and likely not applicable when rendering an Assembly based on some arbitrary query. Therefore, the Configuration interface is not included in the Dynamic Viewer.

3.3 Visual Collaboration – 3D Markup Views

3D Visualization was introduced with Visual Collaboration as a tool used to help visualize complex product information using a simple user interface. Part of this capability allowed users to 'freeze' views of a 3D View, add 2D Markup (graphics and text) and save the resulting image with a comment as part of a discussion thread. Users can then reconstruct a 3D View by selecting the graphic in the associated message. Reconstructing a 3D View requires information about the state of the view, for example: camera zoom and rotation and the hidden/shown 3D Components. This information, known as 'view state' is contained with the graphic in the message.

Visual Collaboration gives users the ability to define snapshots of 3D Views with associated comments to define discussion threads related to 3D Design, for example. This capability also provides the ability to reset a 3D View to restore the original view when the snapshot was created. For DPN, restoring a 3D View necessitates the re-execution of the query used to define the view (see Section 3.5), any parameters used, the view mode, and the camera location. The ability to add a 3D Markup to a snapshot on an SSVC comment was added in 12.0 SP9.

3.4 View Modes

View Modes were added in 12.0 SP7 as a mechanism to visualize 3D geometry data using alternate color and/or Opacities. View Modes are enabled exclusively using the Query Processor API with the addition of Rendering Configurations. See Sections 6 and 6.2.6 respectively.

3 PMI information is included by default in the CAD conversion process

3.5 Saved Views

Saved Views were added in 12.0 SP8 as a mechanism to restore Dynamic Views by applying the input information used to create the original view. Restoring a Saved View automatically re-applies the functions that resulted in the view at the time it was saved. These include the selected Dynamic View Definition, the Parameter Values used, the View Mode selected, and added Parts / Assemblies (see Section 3.6), and the current camera position.

The Saved View includes only *input* information as opposed to the IDs, for example, of all the View Files displayed. As a result, it is important to state that a restored View is not guaranteed to match the original View. This is because the results returned by the execution of a Query Definition may include a different result set. Saved Views will remain associated with all versions of the Target Item.

It is also important to note that not all view parameters are restored. For example, the Display Style, Exploded Increment, Measurements, and Cutting Planes are not included.

Note: Display Style, Exploded Increment, Measurements, and/or Cutting Planes may be included in future releases.

Saved Views are an Item Type with source types derived from the Dynamic View Definition context item types, collected under the poly Item Type ‘SVTargetItem’. On creation of a Dynamic View Definition, the context Item Type will be added to the "Poly Sources" list of the ‘SVTargetItem’ item type, if it was not already added. The ‘SVTargetItem’ item type will be created during the import of the DynamicModelConstrator AML package.

Warning If a Dynamic View Definition was created for an item type other than CAD, prior to 12.0 SP8, the Saved View button may not appear in the Dynamic Viewer tab (due to the source type not existing). In this case, the context item for this Dynamic View Definition must be added to the ‘Poly Sources’ list of the ‘SVTargeItem’ item type.

Note: If the Query Definition or Query Definition Parameters are modified or removed after a Saved View is created, there may be errors restoring the view state.

3.5.1 Creating a Saved View

To create a Saved View, select the icon to the right of the Saved Views Row in the Views Tab of the Dynamic Viewer.

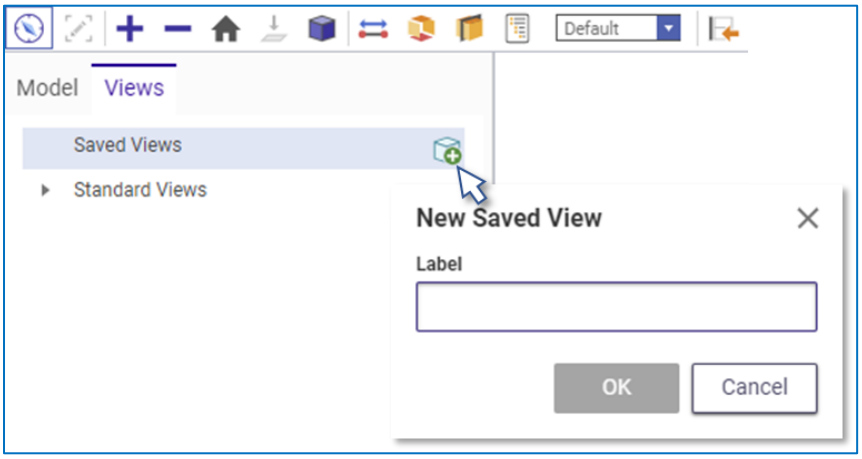

Figure 10. Create a Saved View

The system will open a dialog in which the user will enter a name/label for the Saved view. Saved View Names are restricted to alphanumeric characters. A Name value must be supplied, only valid characters will be permitted, and the maximum length for a Name is 32 characters. The **OK** button will only be enabled when a valid Name is provided. Names are not required to be unique.

Selecting the **OK** button will close the dialog and the Saved View will be stored with the Item that is opened and associated with the selected Dynamic View Definition. A row will be added to the Saved View Section with the name chosen. When a Saved View is created, it can only be used for the Item the Dynamic View was opened on and only on the selected Dynamic View Definition (5.2.4) used. Selecting the **Cancel** button will cancel the operation and a Saved View will not be created.

3.5.2 Restore a Saved View

To restore a Saved View, open the Item the Saved View was created on, select the Dynamic View Definition (5.2.4) used, and open the **Views** Tab in the Dynamic Viewer. Select the Saved View to restore. Note that restoring a Saved View results in a refresh of the View and the Parameters and View Mode saved within the Saved View will be applied. Any previous settings for Parameters or selected View Mode are overridden. Once selected, the System will re-render the 3D View as if the user chose the same settings when the Saved View was created and refreshed the view.

Figure 11. Restore a Saved View

3.5.3 Delete a Saved View

To permanently remove a Saved View, open the Item the Saved View was created on, select the Dynamic View Definition (5.2.4) used, open the **Views** Tab in the Dynamic Viewer, and select the '**X**' to the right of the Saved View row. The user will be prompted to confirm the deletion of the Saved View. Selecting the **Delete** button will result in the Saved View being removed from the system. Selecting the **Cancel** button will cancel the operation and the Saved View will not be removed.

3.6 Digital Mockup

For 3D Visualization, Digital Mockup refers to an ad-hoc arrangement of 3D Component geometry for purposes of analysis or review for example. Such functionality allows users to visualize collections of Parts / Assemblies in a manner that may be different from how they were defined within the CAD System. Starting with the release of 12.0 SP10, the Dynamic Viewer includes functions to enable digital mockup by providing the ability to place additional Parts and/or Assemblies within a single 3D View. Future releases will add the ability to manipulate the position, orientation (by 3D rotation), and display of selected 3D geometry to extend the Digital Mockup capability.

3.6.1 Adding Parts / Assemblies to a 3D View

Creating the 3D View for a digital mockup starts with an Assembly – referred to as the context Assembly, the Dynamic Viewer, and a selected Dynamic View Definition. Additional Parts and/or Assemblies can be added to the context Assembly by selecting the **Add Part / Assembly** Button.

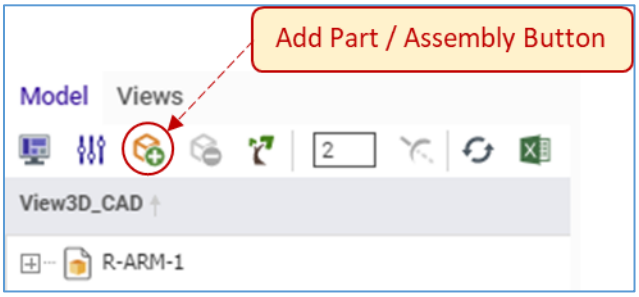

Figure 12. Add Part / Assembly

The system will display a Search Grid dialog for the Context ItemType associated with the selected Dynamic View Definition (see Section 5.1). Users can use the search grid to filter the set of Items, select Part or Assembly Item(s) and select the **OK** button. Selecting Cancel closes the dialog and no additions will be made. The search dialog is closed, and the selected Part/Assembly is added to the view.

Whenever a Part or Assembly is added to the Dynamic Viewer, the associated Dynamic View Definition is re-executed (using selected Parameters) and the view (Tree Grid View and 3D View) is refreshed. All defined View Modes and Parameters will apply to the context and added Parts/Assemblies. The selected Part / Assembly is added as a separate root Item in the Tree Grid View.

Note: The root nodes displayed in the Tree Grid View will be ordered based on the sorting logic of the associated Query Definition. It is possible that the node for the context Assembly will not be displayed at the top.

It is important to note that there is no transformation applied to the root of the added Part or Assembly.

3.6.2 Removing Parts / Assemblies from a 3D View

To remove an added Part / Assembly, select the **Remove Part / Assembly** Button.

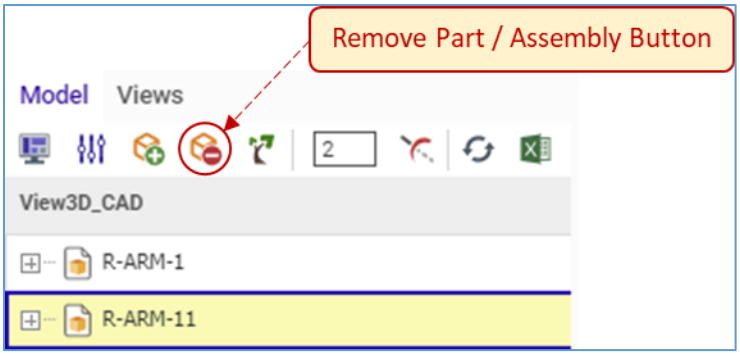

Figure 13. Remove Part / Assembly

This button will only be enabled when the root node of an added Part / Assembly is selected in the Tree Grid View. It is not possible to remove the context Assembly. When the button is selected, the view (Tree Grid View and 3D View) is refreshed and the selected Part / Assembly is removed.

Note: There is no prompt to confirm the deletion of a selected Part / Assembly

3.6.3 Restoring a Digital Mockup

The ad-hoc view state created using Digital Mockup may be restored for future use using Saved Views. To capture a Digital Mockup in a Saved View, make sure all the applied parameters and view modes are set to the desired values, and follow the instructions in section 3.5.1 to create a Saved View.

Alternatively, a Digital Mockup can be restored from a snapshot in the discussion panel, as described in section 3.3 Visual Collaboration - 3D Markups.

3.7 3D Viewer Preferences

3D Viewer Preferences were introduced in 12.0 SP12 to enable users to configure their own 3D viewer preference settings. 3D Viewer Preferences are persistent for the user and apply to the Monolithic and Dynamic Viewers separately. 3D Viewer Preferences can be accessed from the command toolbar in the Monolithic or Dynamic Viewers. The available 3D Viewer Preferences will continue to expand in the future.

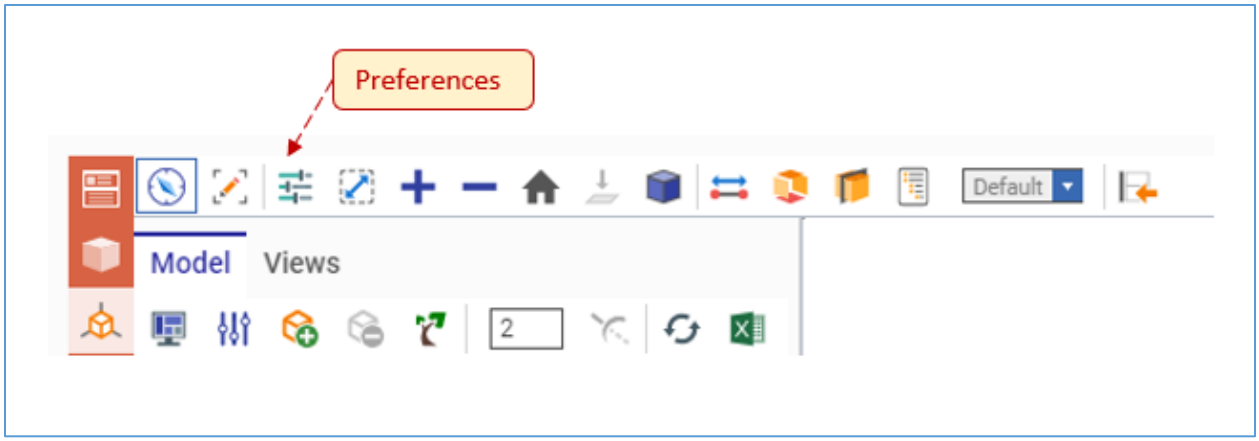

Figure 14. Preferences in Dynamic Viewer Command Toolbar

3.7.1 Zoom Preferences

Zoom Preferences were introduced in 12.0 SP12 for both the Monolithic and Dynamic Viewers. The default setting is ‘Zoom to Viewport Center’, which reflects the legacy behavior of zooming to the center point of the viewport when scrolling the mouse wheel, regardless of the mouse position within the viewer.

A new setting is now possible for users by selecting ‘Zoom to Cursor Position’. This setting will center the zoom to the current mouse position within the viewer, when scrolling the mouse wheel.

These settings are persistent for the user and unique to the Monolithic and Dynamic Viewer; meaning unique settings may be set for each.

Note: The 3D Viewer Zoom Preference does not change the functionality of the Zoom Up ‘+’ and Zoom Down ‘-‘ buttons in the command toolbar.

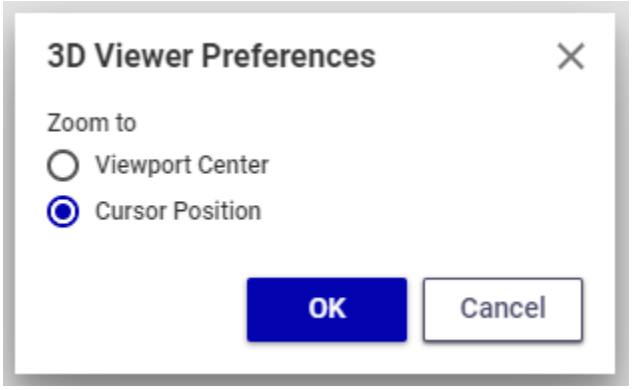

Figure 15. Zoom Preferences

4 Dynamic Enabling

In Aras Innovator 12.0 SP4 an Action was added that, when used on a CAD Item Assembly that is currently not Dynamically Enabled, will process the complete Assembly hierarchy to generate the necessary data to enable the Assembly for viewing using the Dynamic Viewer. As noted in Figure 4, in order for the Dynamic Viewer to be used, CAD Instance data needs to exist in the CAD Structure that identify instances and their transformations. In addition, a 'Dynamic Enabled' flag is set on each CAD Assembly. The combination of this information will enable the Dynamic Viewer. The Action – **Dynamically Enable Legacy CAD Items** - was added so that any CAD Structure that was generated in Aras Innovator versions from 11.0 SP11 and higher, upgraded to Aras Innovator 12.0 SP4 and higher, could be processed to add the CAD Instance Items so that those Assemblies could be visualized using the Dynamic Viewer.

Note: Aras Innovator versions prior to 11.0 SP11 do not have the necessary data that the Dynamic Enable Action requires. For these environments, users will have to re-import CAD data in order to use the Dynamic Viewer

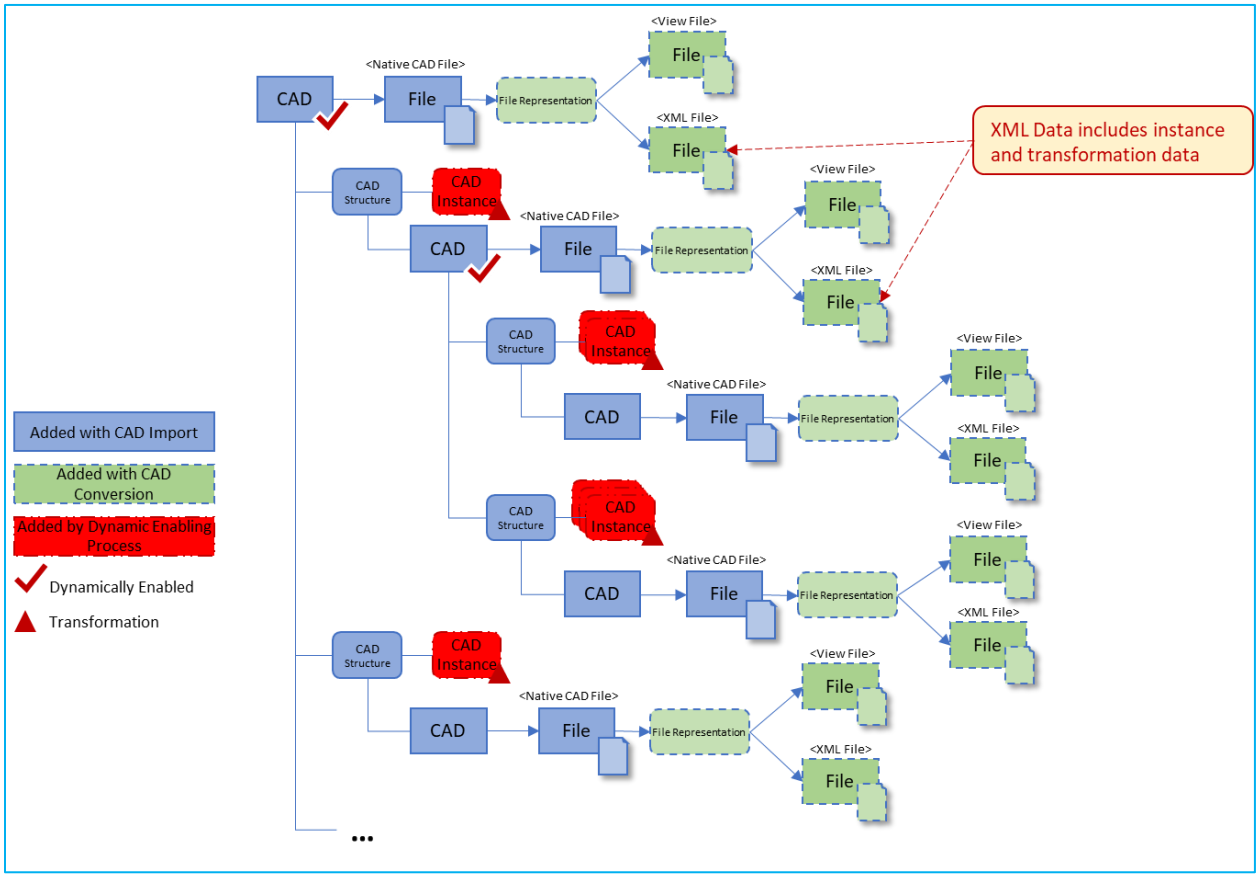

Figure 16. Dynamic Enable Process

4.1 Dynamic Enable Process

The Dynamic Enable process uses the XML data created from the Conversion Process (see Section 3.1 and Figure 16) to identify each Component Part and Sub-Assembly instance and its transformation matrix to generate CAD Instance Items for each level in the CAD / CAD Structure hierarchy. This XML data exists for all CAD data imported into Aras Innovator versions 11.0 SP11 and higher. The Dynamic Viewer uses the same view file data (SCS) to render the 3D geometry. Because the necessary information exists, the Dynamic Enabling process executes quickly as compared to the import and conversion process for new CAD data.

The Dynamic Enable process uses the Conversion Server to process the selected CAD Assembly asynchronously on the Server. As a result, the only status provided to the user is the Conversion Task Item (**Administration**->**File Handling**->**Conversion Tasks** in the TOC) that is subsequently created when the Action is executed. When complete, the selected Assembly and all sub-Assemblies will be dynamically enabled. Users can check the **Dynamic Enabled** Boolean property for each CAD Assembly Item.

Note: The Dynamic Enable process will modify CAD Assembly Items by setting the Dynamic Enable Boolean Property. It requires that CAD ItemTypes are manually versioned. However, this process will not update the generation or revision of CAD Items.

4.1.1 Dynamic Enable Query

When processing Assemblies for *Dynamic Enabling*, a Query Definition is used to identify all sub-assemblies and sub-sub-assemblies and so on. The default Query Definition - **View3D_DynamicallyEnablingCAD** – is used for this purpose. The query uses the '`Exists`' Aggregate Function in the Where Condition applied to the CAD Structure, along with added Query Items that will identify the existence of related CAD Structures and CAD Items. The result will only process CAD Items in the CAD Structure hierarchy that are Assemblies – it ignores Component CAD Items. This is because the Dynamic Viewer only applies to Assemblies.

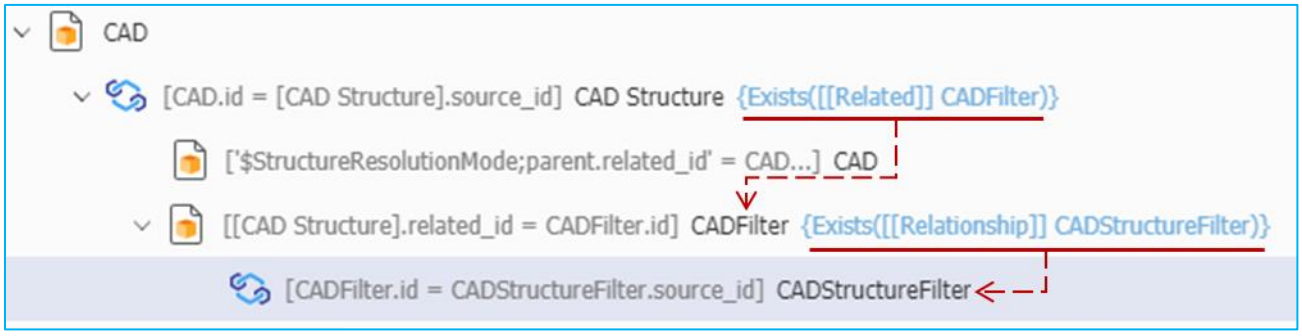

Figure 17. Dynamic Enable Query

4.2 How to Enable Legacy CAD Items to Work with Dynamic Visualization

1. Enable the DynamicEnabler User. The DynamicEnabler User is required to update legacy CAD items only – it is not required to dynamically enable newly converted CAD items.

a. Log into Aras Innovator as an administrator.

b. Find and edit the dynamic_enabler User item.

c. Set **Logon Enabled** to true and choose an appropriate password.

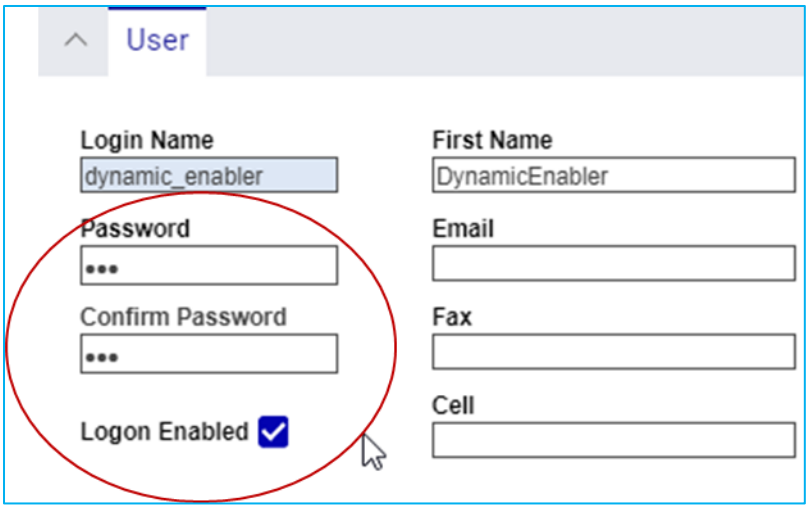

Figure 18.

2. On the Aras Innovator Server, open the InnovatorServerConfig.xml file and add the following Operating Parameter:

```
<operating_parameter key="dynamic_enabler" value="dynamic_enabler|<Password>"
/>
```

3. In Aras Innovator, add the **DynamicEnabler User** Identity as a member of the Aras PLM Identity.

> **Note:** This is done because DynamicEnabler needs Update access to all CAD items that need to work with Dynamic Visualization.

4. Select CAD Items from the database and run the **Dynamically Enable Legacy CAD Items** Action as needed.

5 Creating Query and Tree Grid View Definitions

Dynamic visualization renders a 3D View based on the product of a query. The data model on which the query is executed needs to include the elements required for a 3D view:

- 3D Component geometry
- Instances
- The transformations necessary to place and orient each instance

This information is contained in the out-of-the-box data model for CAD, CAD Structure, and CAD Instance ItemTypes. This section describes the default Query Definition, how you can copy and customize it, how to use it in a Tree Grid View Definition, and how to 'register' this Tree Grid View Definition for use by the Dynamic Viewer.

5.1 Query Definitions

Query Definitions are used by the Dynamic Viewer to determine what should be included in a 3D View when the View is processed (see Section 3.2.2). There are restrictions on the Query Items included and the specific relationships in order for the Dynamic Viewer to work properly. This section identifies the information that is required to render a dynamic view, where this information is located by default, the Query Definition that properly identifies this data, and areas that can be customized.

5.1.1 CAD / CAD Structure Data Model

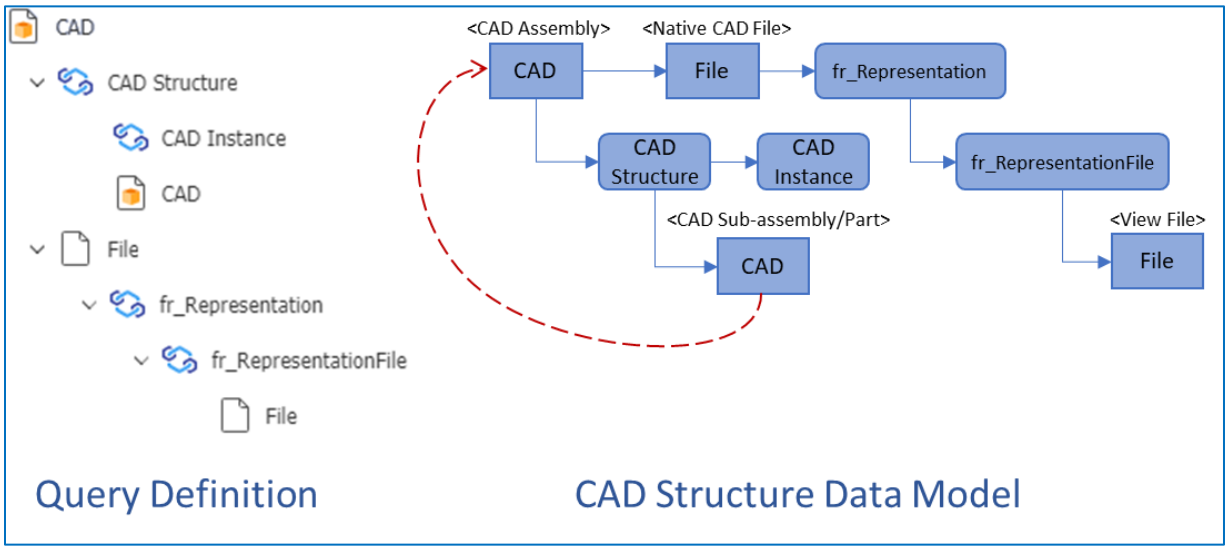

Figure 19. CAD Structure Data Model

In order to render a 3D View, the following data is required:

- *View files*, for 3D component geometry
- *Transformations*, to determine where to place the 3D component geometry in 3D space

- *Instances*, to determine the number of each of the 3D components
- *Assembly hierarchy*, to determine the lineage of transformations to apply

The view file is associated with each `CAD` Item in the `CAD Structure`. It is accessed as *related content* on the File Item used for storing the native CAD `File`. This 'related content' is referred to as a 'File Representation' because the generated View File is based on the associated native CAD file. Note also that the XML data generated from the conversion process (see Section 3.1) is also stored as a File Representation. The Transformation is stored as a String but parsed as a 4X4 matrix of floating point values.4 This String Property is contained in the `CAD Instance` Relationship Item; which is directly associated with the `CAD Structure` Relationship. The number of `CAD Instance` Items determine the number of Instances of the related/child `CAD` Item within the parent/source CAD Assembly. The following diagram further illustrates the data model. Note that the File Representations are removed for clarity.

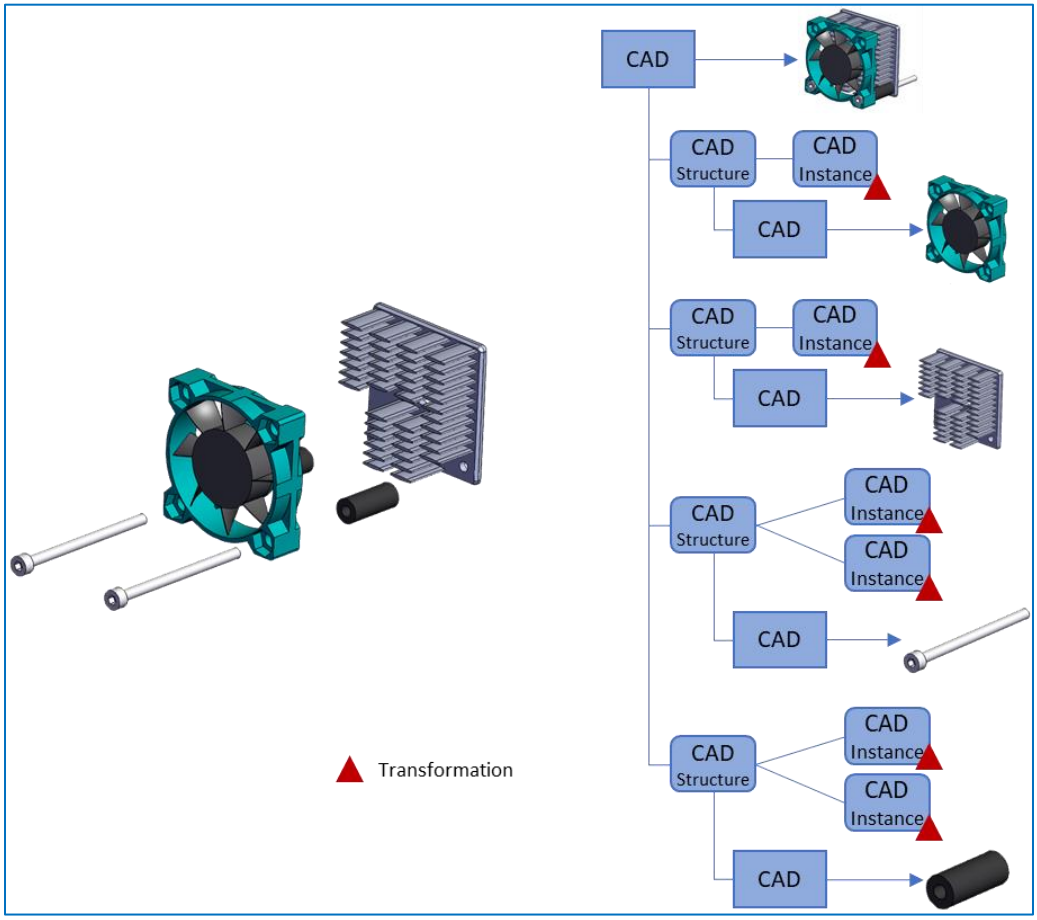

Figure 20. CAD Assembly / Parts and Instances

CAD Structure queries are processed top-down in a recursive manner. That is, CAD Items 'higher' in the CAD Structure hierarchy are processed before their children and so on. The order of processing determines the order of applying the transformations.

4 The interpretation of the matrix values is determined by the 'CAD Transformation Matrix Format' Variable (see Variables in the Administration folder in the main Table of Contents).

5.1.2 Base Query Definition

Figure 19 shows the Query Items for the Base Query Definition. These elements correspond to the diagram. For Aras Innovator 12.0 SP2, the Query Definition named 'View3D_CAD', accessible in the Table of Contents folder hierarchy **Administration->Configuration->Query Definitions**, is configured by default for the Dynamic Viewer (see section 5.2.1 for the default Tree Grid View Definition). This Query Definition can be used as a guide when creating alternative Queries for use with the Dynamic Viewer.

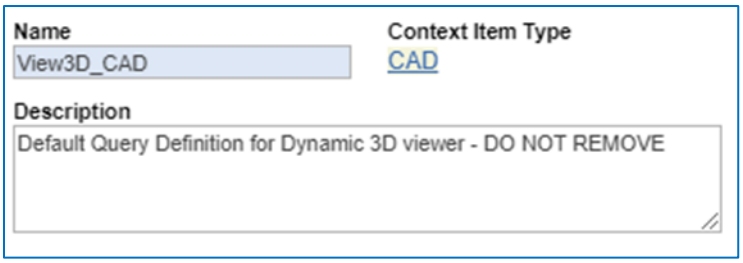

Figure 21. Base Query Definition - View3D_CAD

Note: The Base Query Definition should not be modified. It has default Permissions designed to prevent inadvertent removal or change. See section 5.1.3 for a description of how to customize the Query Definition used for the Dynamic Viewer.

The base query identifies what data is necessary for the required information described in section 5.1.1. It is important to understand that this Query Definition, when executed, will return the 'Latest' in Structure Resolution – see Section 5.1.3.5) CAD Structure.

Note: The CAD Structure Relationship is Hard Fixed by default

The following Properties are required to be included in a Query Definition for the Dynamic Viewer:

Table 1: Required Base Query Properties

ItemType	Alias	Property
CAD	CAD	`id, keyed_name, x_min, x_max, y_min, y_max, z_min, z_max, dynamic_enabled`
CAD Instance	CAD Instance	`id, transformation_matrix`
File	File_1	`id, filename`

The CAD Query Item, defined as the child of the root (context Query Item), 'reuses' the relationships and Properties of the root Query Item. This also defines a recursive Query whereby all Items of the child will be re-queried with the same Properties and related content as the root, and so on. The `dynamic_enabled` Property is new for use with Dynamic Visualization. This Property only applies to CAD Items that represent Assemblies and will only be `true` when that associated CAD Item has been converted to produce the necessary data for the Dynamic Viewer (see Section 3.1). By default, the `keyed_name` Property value is used for the text in the Tree Grid View. The `xyz_min/max` Properties are used to define the bounding volume for the associated geometry in the view file. It is used by the 3D View to help optimize the display priority of 3D component geometry when rendering and focus the camera when the view data is refreshed.

The CAD Instance Query Item should include the `id` and `transformation_matrix` Properties. As noted, the transformation matrix is used to position the view file geometry of the related child CAD Item.

The File Query Item that refers to the view file (note the Alias name - *File_1* - in the table) should include the `id` and the `filename` Properties. These values are used to form the proper URL to the vault so the 3D Viewer can retrieve the view files during processing (see Section 3.2.2).

5.1.3 Customizing the Query Definition

The Base Query Definition should not be modified. Doing so risks disabling Dynamic Visualization entirely. Instead, the best approach to creating alternative/custom queries is to copy this Item and modify the copy to adjust/add whatever changes are necessary. This section identifies the scope of changes that are permissible and how to make those changes.5

5.1.3.1 Adding Properties

Any Property can be added to the base Query Definition as long as the Core Properties (see Table 1) are not removed. In addition, there are no restrictions on the set of Properties added to additional related Query Items. Properties are mapped to Tree nodes or columns in the Tree Grid View Definition (see Section 5.2). Doing so exposes the values of these Properties in the Dynamic Viewer.

Note: You must add Properties to the Query Definition in order for them to be mapped in a Tree Grid View Definition.

5.1.3.2 Adding related content

Related ItemTypes – that is, the ItemTypes that are related to any ItemType via an Item Property – can be included in a Query Definition for the Dynamic Viewer. As noted in Section 1.1.3, this mechanism enables users to see content related to parts displayed in the 3D View. A related ItemType is added as a separate Query Item in the Query Definition.

Note: Query Items must be mapped in the Tree Grid View Definition in order for related Items to be included in the Tree Grid View.

Any related ItemTypes can be included but it's important to note that the Base Query Definition must not be altered and the context ItemType must be associated with the CAD ItemType. Related ItemTypes use a Join Condition to determine how rows from the two associated ItemType tables should relate. By default, these Join Conditions use the `id` Property of one of the Query Items in the Join. This Join Condition can be altered, but users should understand the ItemType data model they are referring to when doing so.

Since the Part ItemType represents a core Business Object in PLM, it is likely that users will want to see the Part Items associated with the CAD Items returned in the Base query. Parts are associated with CAD Items by default using the Part CAD relationship. Part Items, using this relationship, are the source and CAD Items are the related. Thus, if users want to include the Part CAD relationship as a 'referencing Item' in the Query Definition, they need to include the Part Item using the following steps as described by Figure 22 and Figure 23. Similar steps can be used when adding content that references the selected Query Item – that is, has an Item Property that 'points to' the ItemType of the selected Query Item.

5 See the 'Aras Innovator 12.0 - Query Builder Guide' for more information on using Query Builder to create and modify Query Definitions.

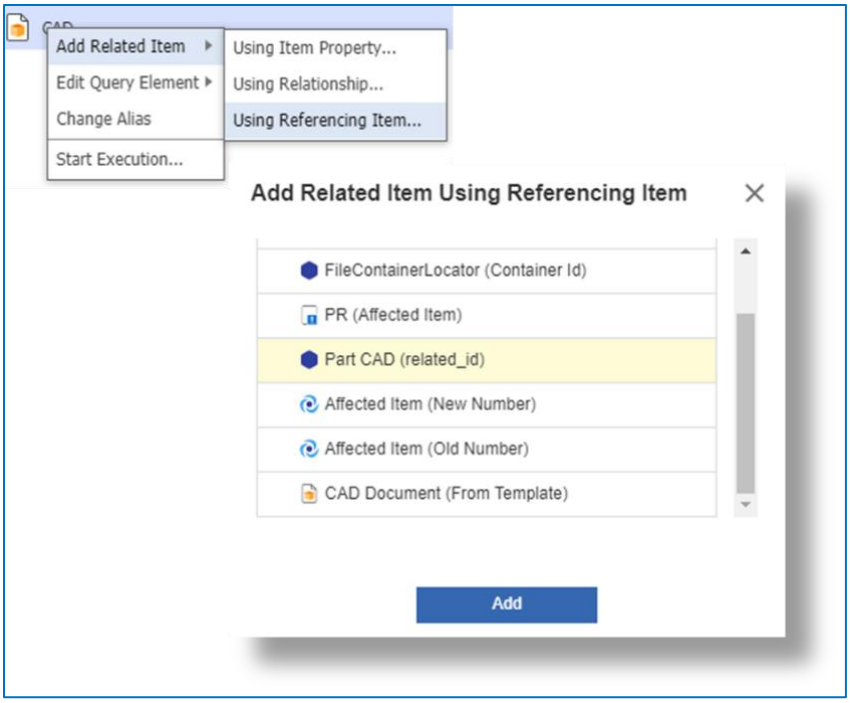

Figure 22. Adding the Part ItemType via Part CAD Step 1

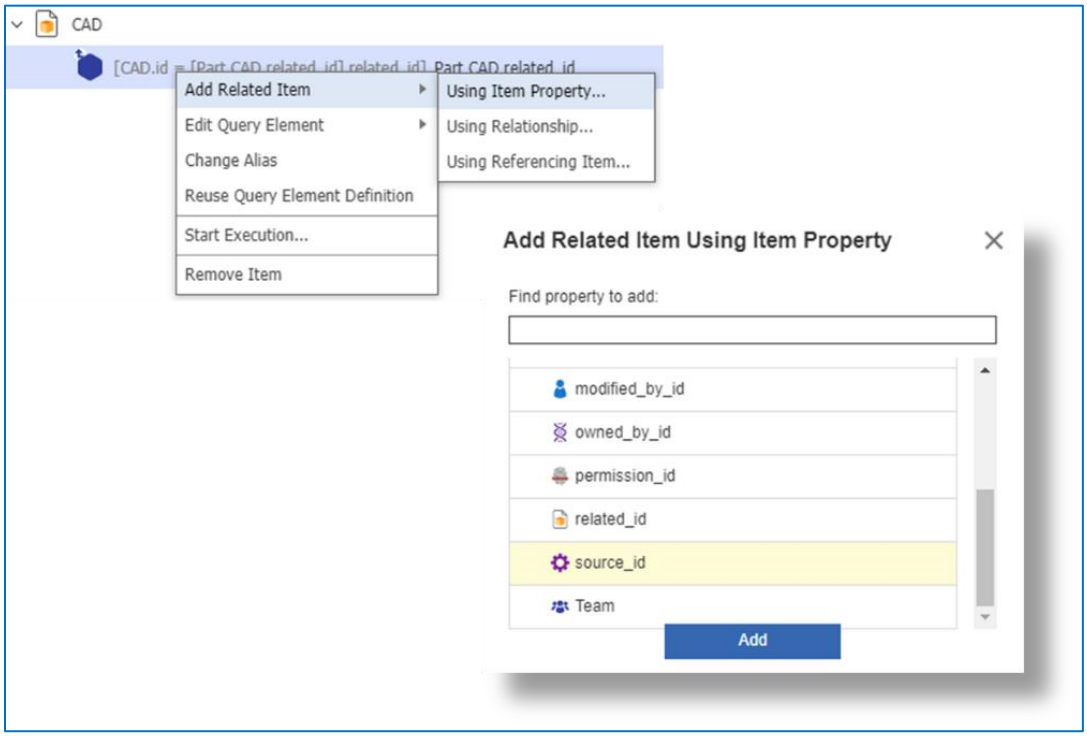

Figure 23. Adding the Part ItemType via Part CAD Step 2

5.1.3.3 Conditions and Filters

Conditional logic can be used to filter the content that is returned from the execution of a Query Definition. *Where* Conditions on the Query Items are used to add conditional logic. For example, the following Where Condition, applied to a related Part Item, results in only those Parts with the String 'valve' included somewhere in the Part Name being included in the query results:

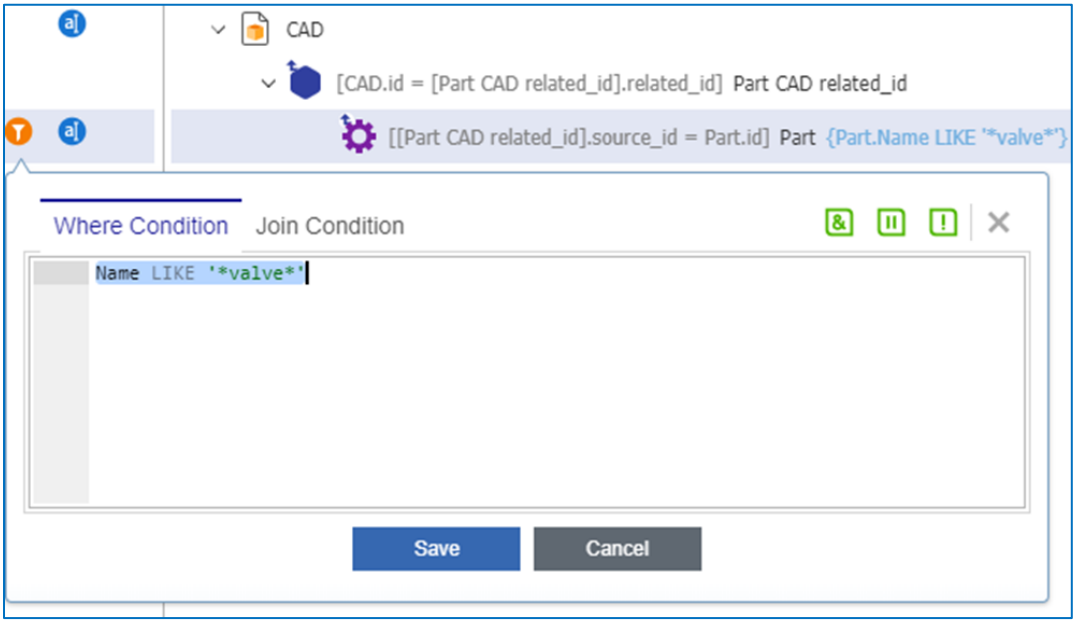

Figure 24. Example Part Where Condition

Note: Properties used in Condition statements should be included with the Query Item. For example, the `Name` Property would need to be included with the Part Query Item for it to apply in this condition. Note also that the example in Figure 24 will only filter Part Items, the Part CAD relationship Items will still be included in the query results.

Conditional logic applied to any ItemType that is *added* to the Base Query Items can be done without affecting the Dynamic Viewer. However, Administrators must use caution when applying conditions to the Query Items that *are* part of the Base Query Definition. For this purpose, the following information should be referenced:

1. Conditions that filter an assembly will result in all descendant CAD Items being filtered.
2. For dynamically-enabled CAD Items, only CAD Items that refer to assemblies will have the `dynamic_enabled` Property set to true. Checking this Property is one way of determining whether the particular CAD Item has child CAD Items using the CAD Structure relationship. For example, to target component (non-assembly) CAD Items the following can be added to the condition statement: [Dynamic Enabled] = 0 AND some other condition.
3. It is possible to filter CAD Items based on conditions applied to related content. For example, to filter component CAD Items based on conditions applied to related Parts:

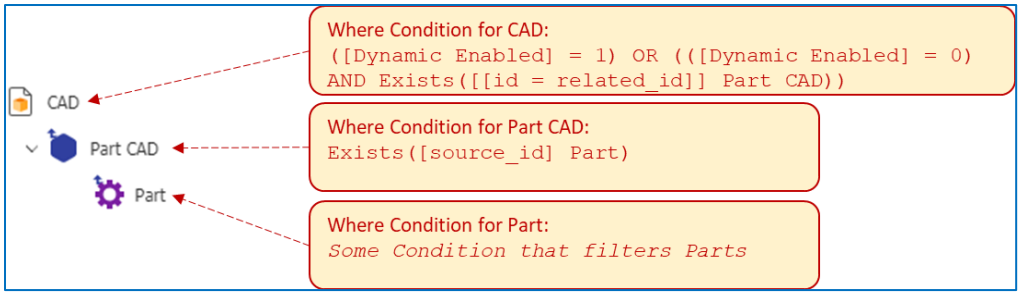

Figure 25. Filtering Component CAD Items

- The Condition applied to the Part Query Item can be whatever logic is necessary to isolate specific Part Items.
- The Condition applied to the Part CAD Query Item uses the '`Exists()`' function. This function effectively applies a *SQL Inner Join* between the Part CAD and Part Tables. The result is the rows for the Part CAD Items are reduced to only those with related Part rows in this case. That is, only Part CAD Items are returned if there are related Part Items returned which are filtered by some logic.
- The Condition applied to the CAD Query Item needs to only associate the related part filter to CAD Items that represent components (non-Assembly Items). To do this, the condition checks the `dynamic_enabled` Property. If it's true, then any related CAD should be included in the results. If it's false ('0'), then include the condition on the *existence* of the Part CAD Items. If a row is included, then include the CAD Item as well.

4. Use caution when applying filters to CAD Query Items. They are reused and exist to execute recursive sub-queries. In this case, the CAD Query Item represents both Assemblies and Components. Applied Conditional logic needs to consider that when Parts in a Dynamic View are placed, they require the application of the full lineage of Transformations from the root CAD Item to each leaf Component CAD Item.
5. If Conditions are applied to CAD Instance Query Items, instances of the related CAD Item are also filtered.
6. The Base Query Definition has a Condition that filters the File Item (view file) related by a File Representation based on the Representation 'kind'; which is SCS. If this Condition is modified it may result in the corresponding 3D geometry being removed.

5.1.3.4 Query Parameters

Query Parameters used in Query Definitions provide the ability to add 'placeholders' for actual values used in Conditional statements. This is useful when there is a need to provide some level of control for additional filtering on the part of the end-user. Using Figure 24 as an example, it is possible to use a Parameter in place of the hard-coded value '*`valve`*'. In this case, Users can provide any value they would like to use in place of the default to filter based on the *name* of the related Part. Figure 26 shows the added Parameter '`partNameFilter`' which can then be used in place of the value 'valve' in Figure 24. This example uses the default Parameter value of '`*`' which is a wildcard character that will result in all names chosen in this case. Also, for this example the Parameter is placed in between two '`%`' characters which will cause the name value given to be valid within the actual Part Name for the Part Items.5 For example, a Parameter value of '`intake`' would match Part Names: '*Intake Valve*', '*Left Intake Spring*', '*Right Intake*' and '*Intake*'. Parameters are added using the Query Parameter Dialog and referenced within a Condition Statement by preceding the Parameter Name with a '`$`'. Query Parameters must be added to the Tree Grid View Definition to be exposed to the user.

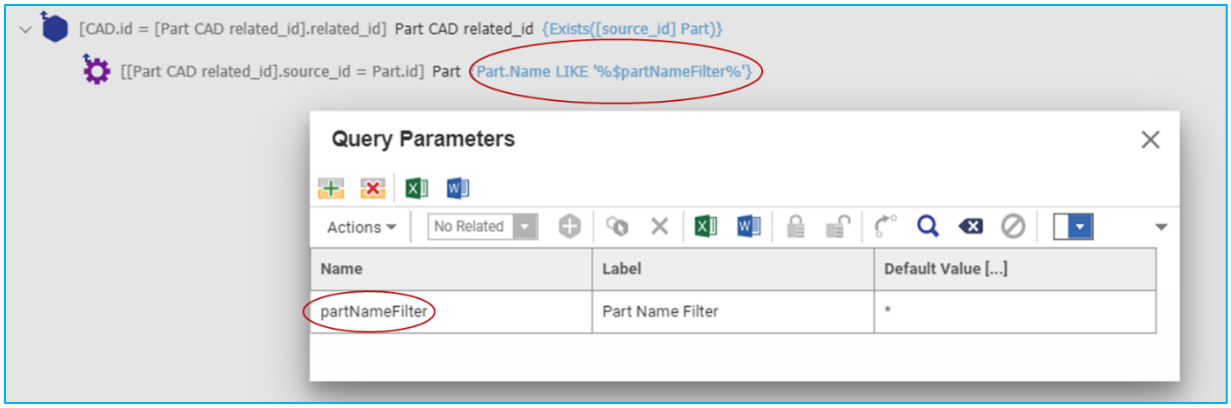

Figure 26. Creating\Using Query Parameters

5.1.3.5 Structure Resolution

Query Parameters enable another query mechanism to control the display of content – *Structure Resolution*. Structure Resolution is used to identify a specific generation of an Item. This is especially useful for CAD structures since the CAD Structure Relationship is fixed by default. This means that a CAD Structure hierarchy will be based on the CAD Items and specific relationships created from the CAD Connector (see Section 3.1). Updates to individual CAD Items do not necessarily involve an update to the CAD Structure Relationship. As a result, Assembly CAD Items will 'point to' a generation of a CAD component that isn't the latest.

Note: In Aras Innovator 12.0 SP2 and SP3, there was an error in processing Structure Resolution Queries for the Dynamic Viewer in that it would assign the Structure Resolution filter to the root CAD Item. Thus, if a Structure Resolution query was meant to include all Latest Released CAD Items (and the root CAD Item was not Released), the query would return nothing. This has been fixed in 12.0 SP4 and later releases.

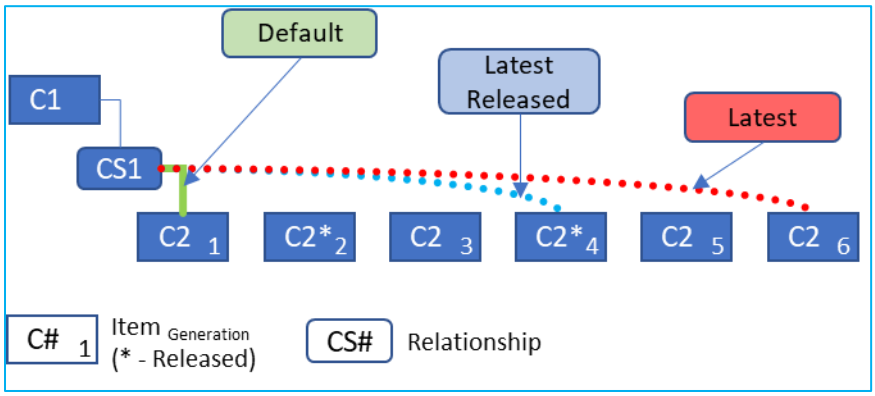

Figure 27. Structure Resolution Logic

Structure Resolution focuses on the Released state of generations of Items. It can be applied based on the following conditions/Modes:

- **Default**: Uses the generation of the Item that the Relationship Item includes. That is, it will be the Item with an ID matching the `related_id` Property of the Relationship.
- **Latest**: Uses the generation of the Item with the highest value.

- **Latest Released**: Uses the generation of the Item with the highest value which is also Released.
- **Latest Released or Latest**: Uses the generation of the Item with the highest value which is also Released or the Latest if there are no Released generations.

5.1.3.6 Enabling Structure Resolution

Structure Resolution is enabled for the Base Query Definition (Section 5.1.2). However, to incorporate it a new Query Definition must be created – as defined earlier in this section – and Structure Resolution enabled for it. The following steps describe how to add Structure Resolution to a Query Definition that derives from the Base Query Definition.

Step 1: Create Structure Resolution Mode Parameter

Applying a specific Structure Resolution Mode ('Default', 'Latest', etc.) is done using a Query Parameter. Using the Query Parameters Dialog, add a Parameter to be used for Structure Resolution Mode. For example '`StructureResolutionMode`' and assign the default value '`Aras.Resolution.EntryPoint;Default`'

Note: Be sure to check the syntax of the Default Value for the Parameter. It is used to select a specific List Item as set in the Tree Grid View Definition and described in Step 4.

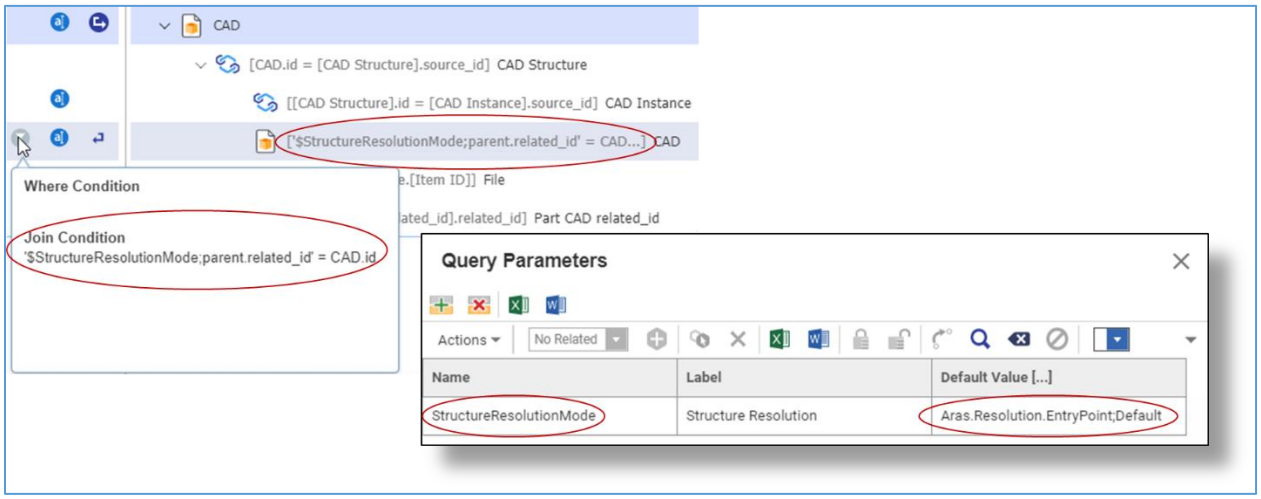

Figure 28. Creating Structure Resolution Parameter

Step 2: Incorporate the Query Parameter

The added Query Parameter must be incorporated into the Query Definition in order for it to be used/exposed to the end-user. Add it as part of the Join Condition of the Child CAD Query Item as shown in Figure 28. Replace the default Join Condition – '`[CAD Structure].related_id = CAD.id`' with the following – '`''$StructureResolutionMode;parent.related_id' = CAD.id`'

Note: Be sure to check the syntax of the Join Condition. It must be as specified above (without the outer quotes). This assumes the value `StructureResolutionMode` was used for the Query Parameter Name. The Structure Resolution Mode Query Parameter can only be inserted once in a Query Definition. It cannot be used/applied to more than one Query Item.

Step 3: Add Resolve Query Entry Point Method

Show the Relationship Tabs for the Query Definition in the Form View if they are not shown. This can be done by selecting *Tabs On* for the **Default Structure View** in the `qry_QueryDefinition` ItemType form. Once shown, select the `qry_QueryDefinitionEvent` Relationship Tab. A Method needs to be added to the `OnBeforeExecute` event. To do this, select the **Select Items Icon** to create a Relationship and select the `qry_ResolveStructureEntryPoint` Method and assign it to the `OnBeforeExecute` event as shown in Figure 29.

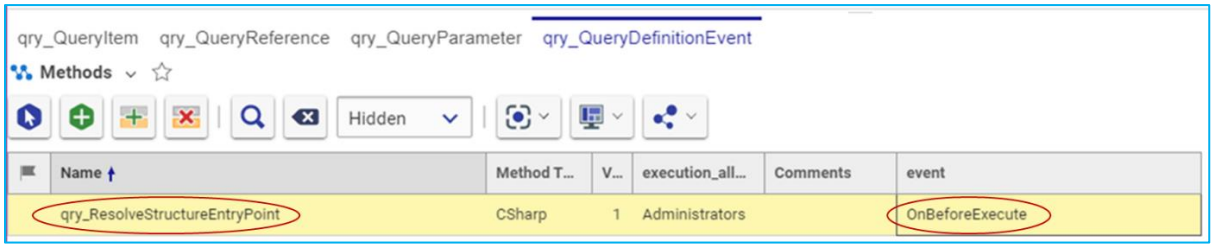

Figure 29. Assigning Structure Resolution OnBeforeExecute Method

Step 4: Add the Query Parameter to the Tree Grid View Definition

In order for the Query Parameter to be used, it must be enabled (made visible) in the Tree Grid View Definition used for the 3D View. Open the Map Parameters Dialog in the Tree Grid View Definition within the Editor View. Select the **Visible** check box for the `StructureResolutionMode` row. The **Data Type** is a `List` and the **Data Source** is the List `qry_StructureResolution`.

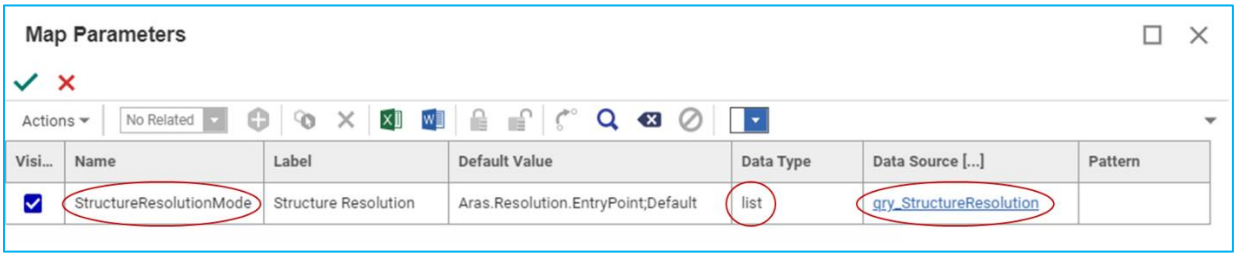

Figure 30. Mapping Tree Grid View Definition Query Parameters

With this Configuration, users will be able to select specific Resolution Modes by opening the Query Parameters Dialog in the Tree Grid View used within the 3D Viewer.

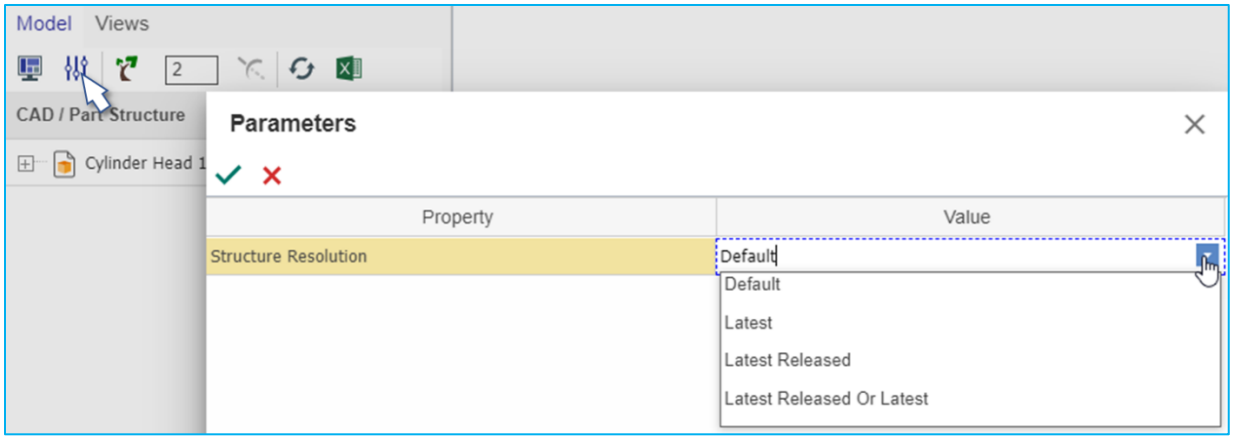

Figure 31. Tree Grid View Parameter Dialog

5.1.3.7 Applying a Where Condition Against Resolved Items

In Aras Innovator 12.0 SP4 and later releases it is possible to add a Where Condition to a Query Item that is used for Structure Resolution (e.g., CAD). Doing so allows the Administrator to apply an additional condition (filter) when resolving the Structure of Items with fixed Relationships (such as CAD / CAD Structure). For example, assuming the CAD Structure in the following diagram:

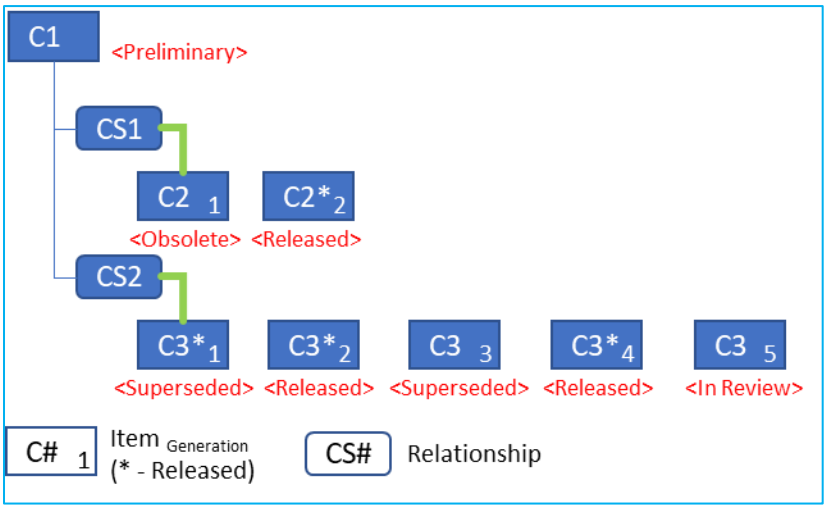

Figure 32. Structure Resolution Logic with Condition

If the Administrator would like to use a Structure Resolution of 'Latest' and apply an additional condition to only include CAD Items that were *In Review* (that is, with a CAD.State = 'In Review') then Where and Join conditions like the following could be added to the Query Item:

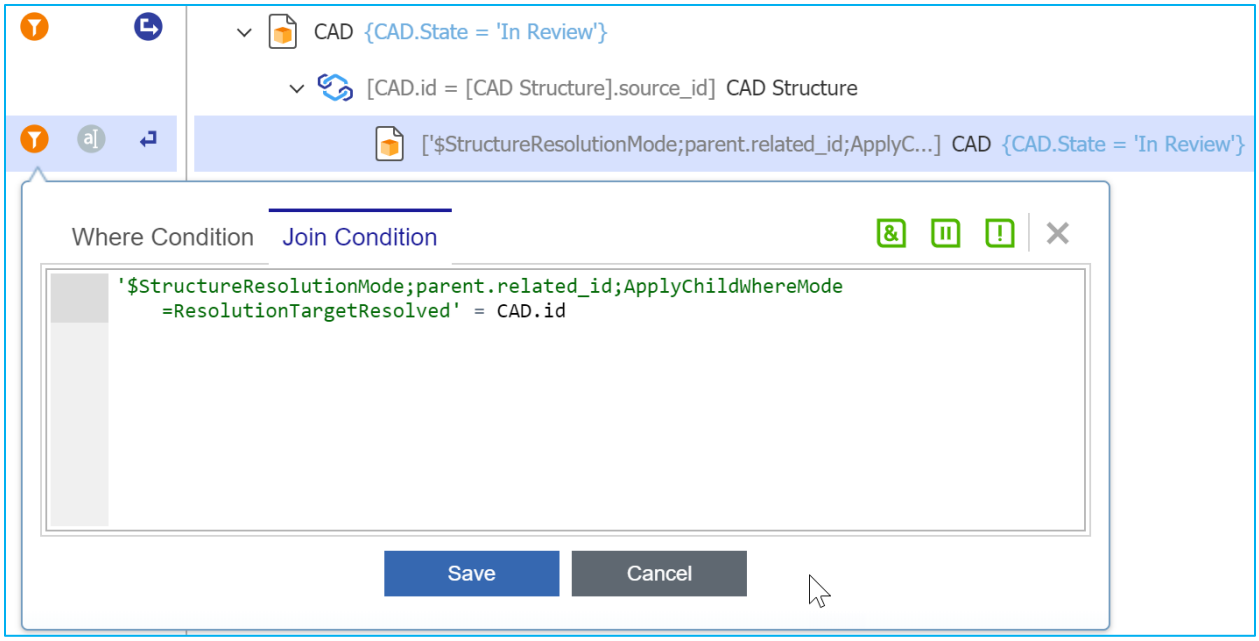

Figure 33. Where/Join Condition Example for Structure Resolution

Note the addition of the left-side of the Join Condition:

`'$StructureResolutionMode;parent.related_id;ApplyChildWhereMode=ResolutionTargetResolved' = CAD.id]`

Executing this Query Definition, with Structure Resolution of '*Latest Released*' with return:

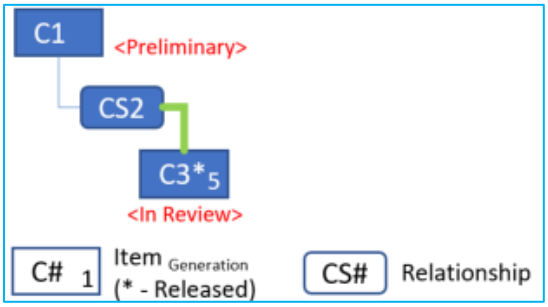

Figure 34. Structure Resolution Logic with Condition Result

In this example, only the CAD Item – CS3, version 5 is returned because of the added Where Condition of `'CAD.State = 'In Review'`.

5.2 Tree Grid View Definitions

Tree Grid View Definitions are used to specify how information returned from the execution of a Query Definition will be displayed. Tree Grid Views are a combination of a Tree View – with the ability to display collapsible hierarchies of data in parent/child relationships – and tables – with columns of individual cells showing various static text and Property values. The Tree Grid View Definition configures the elements of the Tree Grid View; namely nodes for the Tree View (text and icon), columns with column labels, and mappings between Properties and column cells. Tree Grid View Definitions are associated with one, and only one Query Definition. This section describes the default Tree Grid View Definition, how to create custom Tree Grid View Definitions, how the Tree Grid View is used in the Dynamic Viewer, and how to configure for automatic query execution and refresh when the Dynamic View is opened.

5.2.1 Default Tree Grid View Definition

The default Tree Grid View Definition uses the Base Query Definition (see Section 5.1.2) to display the hierarchy of CAD Items only. It is a similar configuration (in view only) to the CAD Item Display used by the Monolithic Viewer (see Section 3.2). For Aras Innovator, the Tree Grid View Definition named 'View3D_CAD', accessible in the Table of Contents folder hierarchy **Administration->Configuration->Tree Grid Views**, is configured by default for the Dynamic Viewer. This Tree Grid View Definition can be used as a guide when creating alternative Views for use with the Dynamic Viewer.

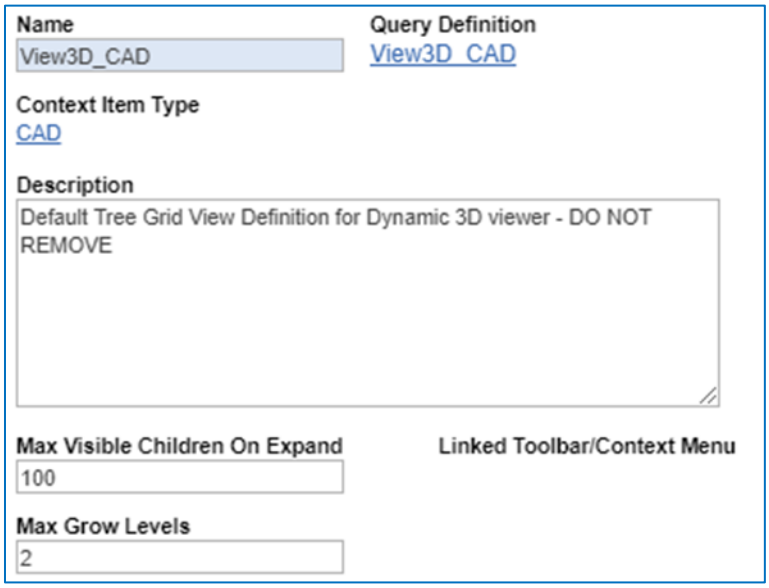

Figure 35. Default Tree Grid View Definition - View3D_CAD

Note: The Default Tree Grid View Definition should not be modified. It has default Permissions designed to prevent inadvertent removal or change. See section 5.2.2 for a description of how to customize the Tree Grid View Definition used for the Dynamic Viewer.

The **Max Visible Children On Expand** field is used to specify the maximum number of peer Items to query and display in the Tree Grid View when the view is refreshed or a related Item is expanded. Peer Items refer to the direct child Items for any parent Item. If there are more Peer Items that exist for any parent Item, a show more link is included in the Tree View portion of the Tree Grid View. Selecting this link displays the next set of peer Items and so on.

The value used for the **Max Visible Children On Expand** setting affects the performance of **3D View -> Tree Grid View** selection synchronization. The reason has to do with the algorithm used to query down to the CAD Item that is associated with the selected 3D component geometry in the view. The larger the breadth of the assembly (that is, the larger the number of CAD Items at any given level in the CAD Structure hierarchy) the larger the potential for multiple simultaneous queries to be executed as the associated 'leaf' node in the Tree View is uncovered.

Figure 36 is used to illustrate the automated process of the system making subsequent queries to display the CAD Item rows in the Tree Grid View associated with the selected part. Assume the context CAD Item ('A') is shown in the Tree Grid View. The shaded circles in the diagram represent CAD Items and the numbers in each represent the query sequence number associated with displaying that CAD Item in the Tree Grid View. When the **Max Visible Children On Expand** is set to '3' the system makes 7 queries until it reaches the depth and breadth of the selected Path. Likewise, when the **Max Visible Children On Expand** is set to '10' the system makes 4 queries until it reaches the depth and breadth of the selected Path.

The **Max Grow Levels** field is used to set the depth of queries for each related Item. The default is 2. The larger the value, the *deeper* the query into each related Item. For a CAD Structure that is large (both in depth and breadth) these queries can take a long time before the Tree Grid View is updated.

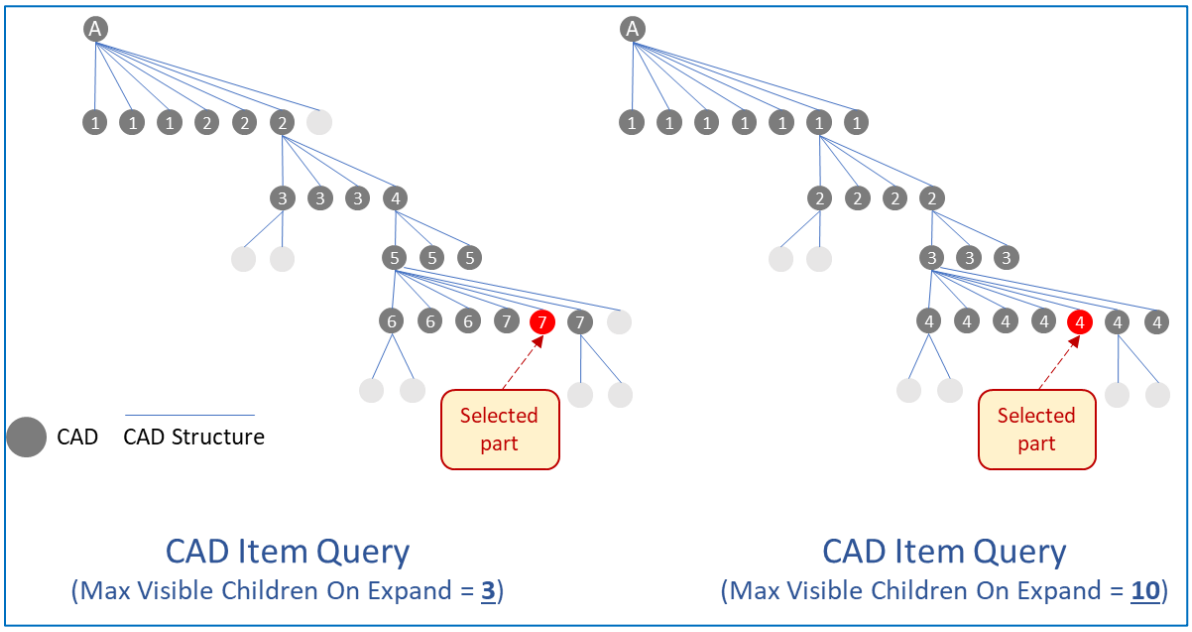

Figure 36. Automated Tree Grid View CAD Item Selection Process

The default Tree Grid View simply maps the CAD Query Items from the Base Query and uses the `keyed_name` Property for the node text.

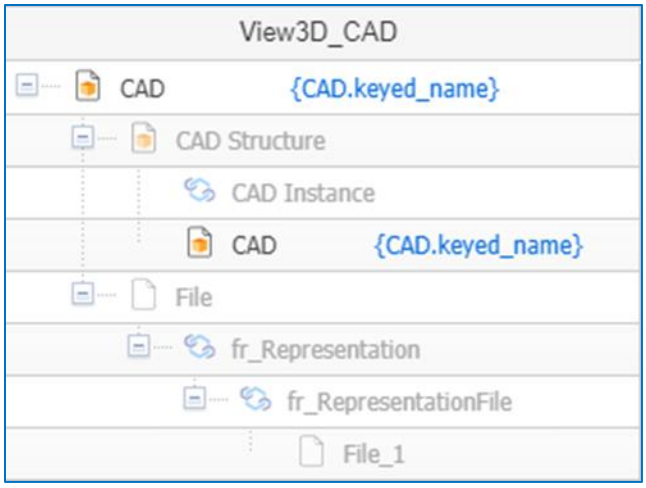

Figure 37. Default Tree Grid View Definition

5.2.2 Customizing the Tree Grid View Definition

Use the following procedure to create a custom Tree Grid View Definition for use with the Dynamic Viewer:

1 Create a new Tree Grid View Definition. It is necessary to create a new Tree Grid View Definition rather than copy the default Tree Grid View when a separate Query definition is being used.

2 Assign a **Name** and **Description**. They help identify the purpose of the Tree Grid View for future reference.

3 Choose a Query Definition based on the Base Query (see Section 5.1.2). Query Definitions must have the CAD ItemType as the context ItemType.

Note: Once a Query Definition is selected and the Tree Grid View Definition saved, the selection of the Query Definition cannot be changed.

After saving the Tree Grid View Definition Item, the Editor View becomes available

4 Assign values for **Max Visible Children On Expand** and **Max Grow Levels**. Refer to Section 5.2.1 for a description of these values and their effect on the Dynamic Viewer.

5 Assign the Query Definition Mappings. Refer to the Tree Grid View Administrator's Guide for a description of how to assign mappings to Query Items. Note that the name of the Tree Grid View Definition appears in the View Selection context menu.

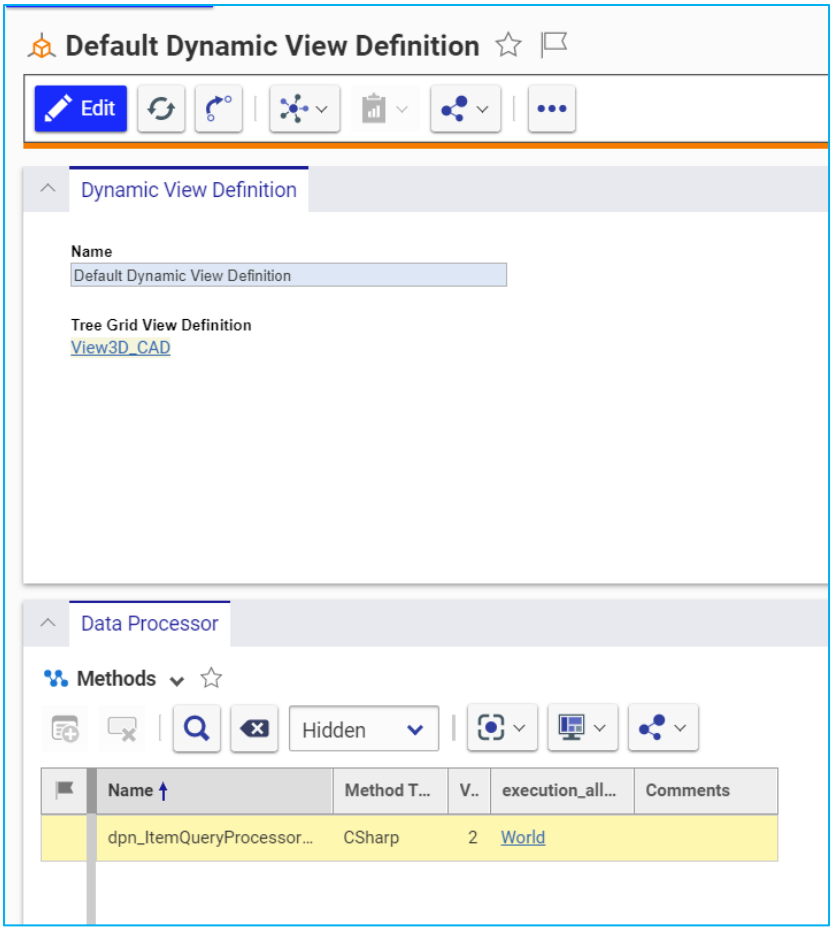

Figure 38.

6 Create a Dynamic View Definition Item. The set of Dynamic View Definitions define the Tree Grid View Definitions assigned to the Dynamic Viewer. The **Name** chosen for the Dynamic View Definition Item is used in the context menu for the Dynamic Viewer as shown below. Dynamic View Definition Items are located in the **Administration->Configuration Table of Contents** (TOC).

   a. Assign a Tree Grid View Definition and Data Processor method. "dpn_ItemQueryProcessorMethod" is available by default to process CAD. Save the Item when complete.

Note: The Dynamic View Definition ItemType was added in Aras Innovator 12.0 SP5 as a replacement for the '*Set Usage for 3D View Definition*' Action used in previous versions. In future versions, this ItemType will be used for additional configuration options.

Warning When migrating/upgrading to Aras Innovator 12.0 SP5 from either 12.0 SP2, 3, or 4 it is necessary to create Dynamic View Definition Items for each Tree Grid View Definition previously configured.

When migrating/upgrading to Aras Innovator 12.0 SP9 from 12.0 SP5, 6, 7 or 8 it is necessary to add a Data Processor method to any Dynamic View Definition without a Data Processor method assigned. This only affects users who have custom Dynamic View definitions with no data processor assigned.

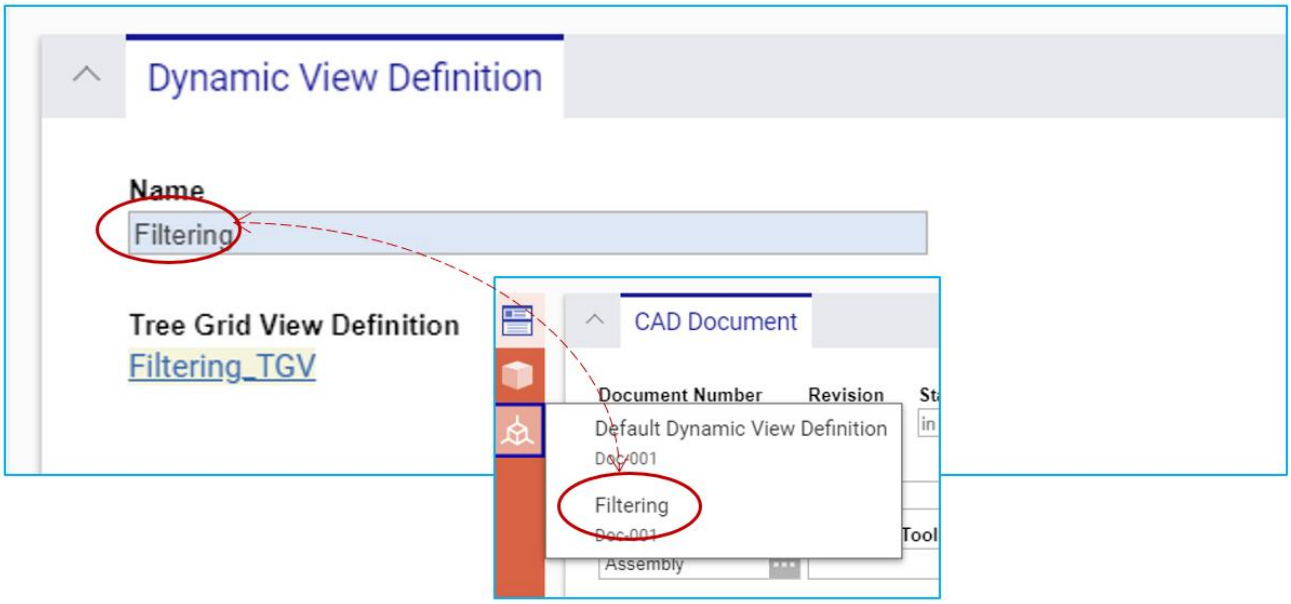

Figure 39. Dynamic View Definition Name

Note: The CAD Query Item must be mapped for the Tree Grid View to function properly in the Dynamic Viewer.

Once set, the Tree Grid View Definition can be removed by removing the corresponding Dynamic View Definition Item.

5.2.3 Tree Grid View / 3D View Synchronization

Synchronization between the Tree Grid View and 3D View refers to the simultaneous selection of 3D component geometry and its corresponding Tree Grid View CAD Node. When selecting a CAD node in the Tree Grid View the corresponding 3D geometry is selected in the 3D View. Likewise, when a part is selected in the 3D View, the corresponding CAD node is selected in the Tree Grid View. Refer to Section 5.2.1 for a description of the process of applying multiple queries until the selected CAD Item is 'uncovered' in the Tree Grid View.

When a CAD Item Node is selected in the Tree Grid View, the 3D component geometry *for all instances of the associated part* is highlighted in the 3D View. There is currently no ability to select a single

instance in the Tree Grid View. However, when selecting a part only the 3D component geometry of the selected body of the part is highlighted (see Figure 40 and Figure 41).

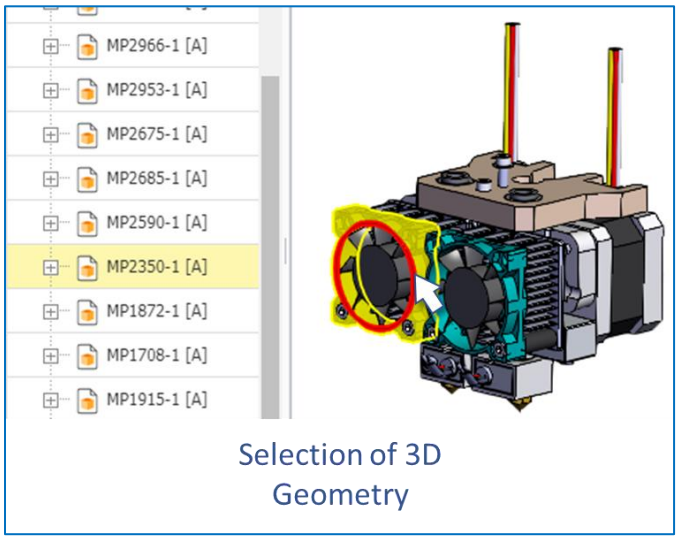

Figure 40. Synchronization via Selected 3D Geometry

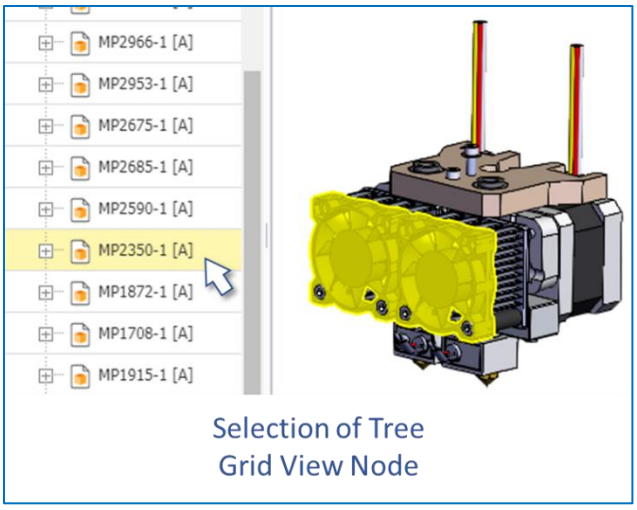

Figure 41. Synchronization via Selected Tree Grid View Node

5.2.4 Selecting a Dynamic View Definition

Note: The default DVD will be displayed as an option in the Dynamic Viewer, along with any other configured DVDs for the context item type. If alternate DVDs are created, and the user no longer wants the default displayed, the default DVD must be removed by the root user.

To use a configured Dynamic View Definition (see Section 5.2.2), select from the available choices when selecting (left-click) the Dynamic View icon in the sidebar.

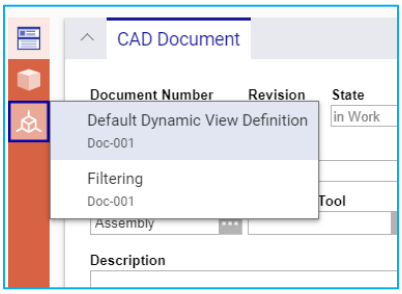

Figure 42. Selecting Configured Tree Grid View Definitions

If there is a Dynamic View Definition that is currently in use, the radio button next to it will be selected.

5.2.5 Auto-Execution

By default, users have to refresh the view (by selecting the refresh button in the Tree Grid View toolbar) to load the 3D View and execute the initial query for the Tree Grid View (see Section 3.2.2). However, it is possible to execute the initial query when the 3D View is opened by selecting the **Execute Query On Opening Dynamic Viewer** option in the Secure Social Preference settings (see Figure 43).

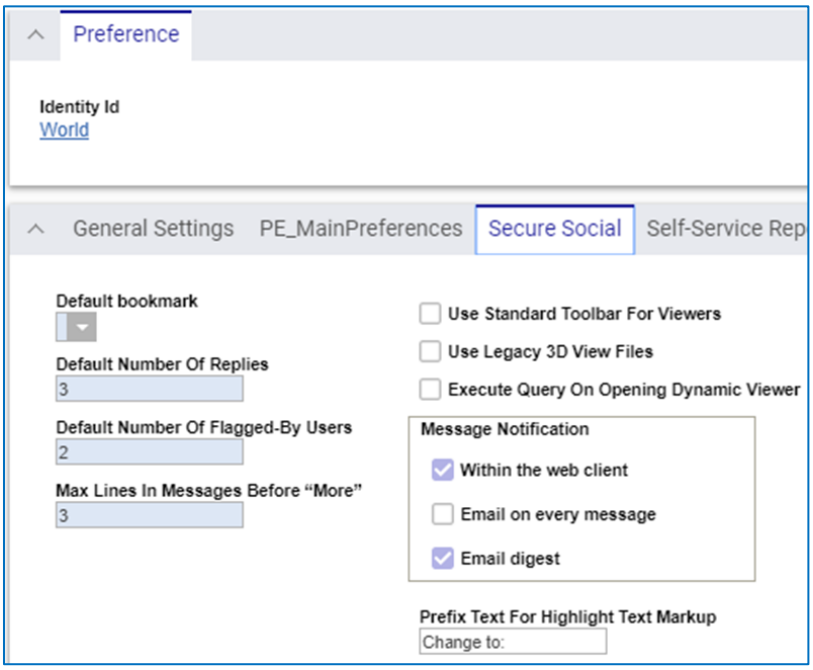

Figure 43. Auto-Execute Setting

5.2.6 Enabling the 'View' Context Menu

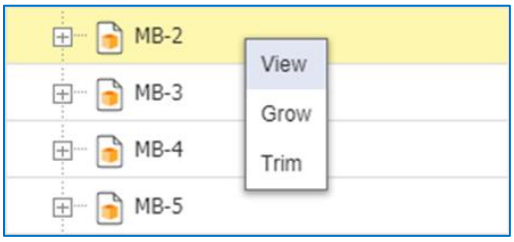

Figure 44. View Context Menu

Tree nodes (rows) in the Tree Grid View have a default context menu, which includes a menu item for opening the associated Item. This menu Item is named **View**. The **View** context menu item is not enabled by default for the Tree Grid View Definition. This is because the associated rows in the Tree Grid View Definition could be combined and thus reference more than one Item (see the Tree Grid View Administrator Guide for more information). To enable the View menu so that it can be used to open the associated Item in the Tree Grid View:

1. Make sure that the `id` Property of the ItemType is included in the Query Definition (see Section 5.1.3.1). The `id` is required to identify the specific Item to open when the action associated with the View menu is executed.
2. Make sure the **Data Template** for the row in the Tree Grid View Definition includes the reference to the Item ID (see Figure 45). The software that executes the 'View' function requires the ItemType and the ID of the specific Item in order to open the associated Form. For this, a data template is used by the Tree Grid View. This JSON template is defined within the Data Template for each row in the Tree Grid View Definition. The format of this JSON string is as follows:

   a. `id`: ID of the Item to open

   b. `type`: ItemType name of the Item to open

   An example Data Template is:

   ```
   {"id": "{CAD.id}", "type": "CAD"}
   ```

   Note the use of brackets - '{' and '}' - in the string. The brackets around the string `CAD.id` is used to extract the id of the Item associated with the row in the Tree Grid View when it is populated in the User Interface. All other strings are static.

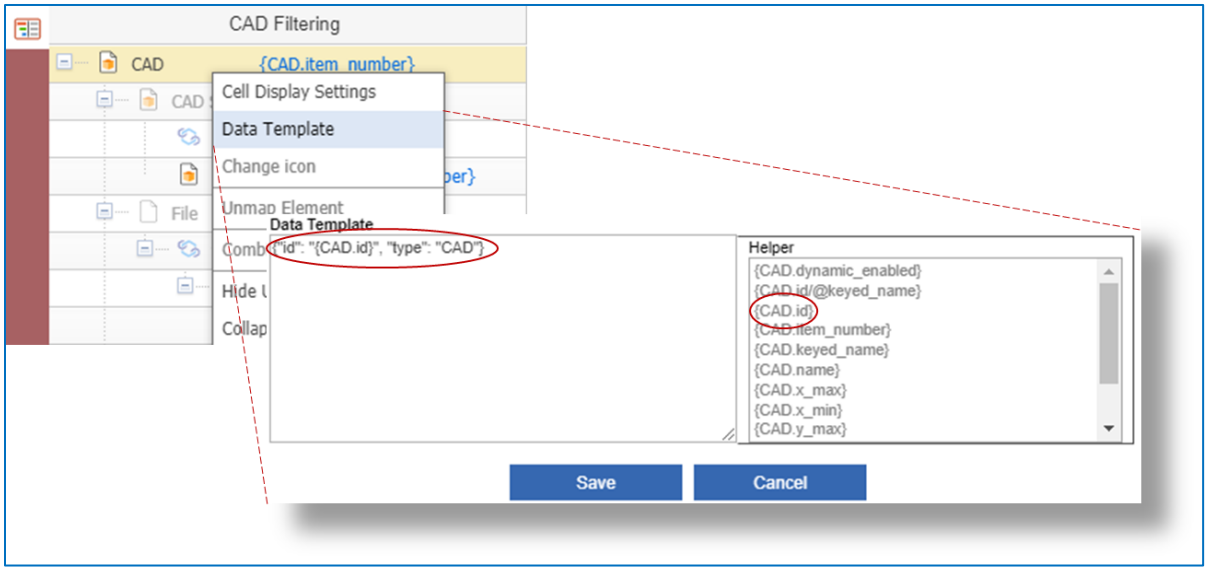

Figure 45. Data Template

6 Alternate Query Processing

This section provides details for implementing and deploying custom Query Processors.

6.1 Overview

For Dynamic Visualization, Query Processing refers to the process of executing a given Query Definition, parsing the results, and constructing a set of 'Product Occurrences' that represent the instances of the view geometry to display in the Dynamic Viewer. Dynamic Visualization includes a Default Query Processor and, as explained in Section 2, uses the CAD and CAD Structure data model as the source for the data required to construct a 3D View. As explained in Section 4, this required data includes:

- 3D Component geometry
- Instances
- The transformations necessary to place and orient each instance

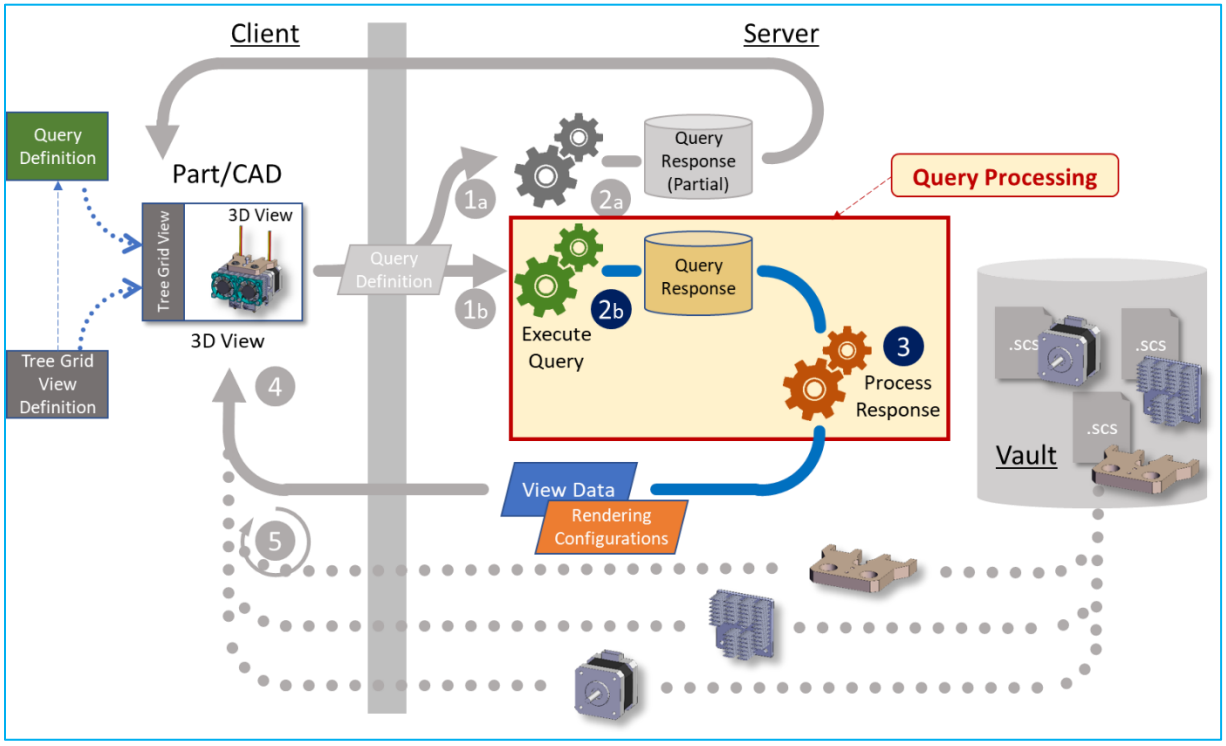

Figure 46. Query Processing

In v12.0 SP7, a Query Processing API was provided for Dynamic Visualization to allow implementations to utilize their own custom Query Processor when default processing will not fulfill the 3D Visualization requirements. A custom Query Processor provides the ability to:

1. Retrieve required data from alternate sources in a customer's data model
2. Apply custom logic when processing the Query Definition execution results
3. Create and apply alternate *Rendering Configurations* used to render the 3D View with different color and opacity for the 3D Component Geometry

4. Map 3D Component Geometry to ItemType nodes displayed in the Tree Grid View other than CAD ItemTypes

This capability greatly expands the applicability of 3D Visualization and enables use cases that are not possible using the Default Query Processor.

6.2 Implementing a Query Processor

Note: Implementing a Query Processor requires knowledge of software development in .Net – specifically C#, IOM, XML and XML Processing, Query Definitions, and the data model (ItemTypes) used by the targeted implementation. As such, creating a custom Query Processor is a highly technical undertaking and should only be performed by competent users. This section is written assuming this level of knowledge.

6.2.1 Create the Query Definition

Developing a custom Query Processor starts with the Query Definition. Users can use the default Query Definition as is, use a modified copy, or use the default simply as a reference when creating a new Query Definition. The specific ItemTypes chosen do not matter so long as the required data is either included in the Query results or is provided in some other manner within the logic of the Query Processor implementation. For this section the discussion will be referring to the Default Query Definition discussed in Section 5. It's important to note the following about this query:

- CAD as the context ItemType

  The Query is rooted (top level node) by a Query Item referring to the CAD ItemType.

- Recursion

  There is a recursive relationship based on CAD and the CAD Structure Relationship. CAD Structure defines the Mechanical Bill of Materials. Thus, there is a hierarchical relationship between upper Assemblies and lower sub-Assemblies and Components.

- Data required for custom processing logic
  - Instances – determined by the related CAD Instance Items with the Transformation strings included. CAD Instances identify both assembly and component instances of child CAD Items.
  - Transformations – attached to each CAD Instance Item. An Identity matrix is assumed when a CAD Instance does not exist. Also, the matrix ordering is assumed to be column-centric. See Section 5.1.2. The transformation value is set during conversion and represents the format of the data expected by the HOOPS 3D Viewer. As such, it is used as it is stored when creating the Product Occurrence data as part of the output of a Query Processor.
  - View Files – attached to each CAD Item through the native file Property. See Section 3.1.

6.2.2 Create the Tree Grid View

When creating a custom query processor, the Tree Grid View Definition can be defined as it is described in Section 5.2. No additional provisions need to be made other than ensuring that the Query Items used for mapping to 3D Component geometry are mapped and displayed in the Tree Grid View.

6.2.3 Create the Dynamic View Definition with Data Processor Method

Dynamic View Definition Items are used to add the Dynamic Viewer icon to the sidebar so the Dynamic Viewer can be opened for an Item of a type associated with the configured Tree Grid View Definition Context ItemType (see Figure 6). They can also be used to assign an alternate Query Processor for use with the Dynamic View Definition.

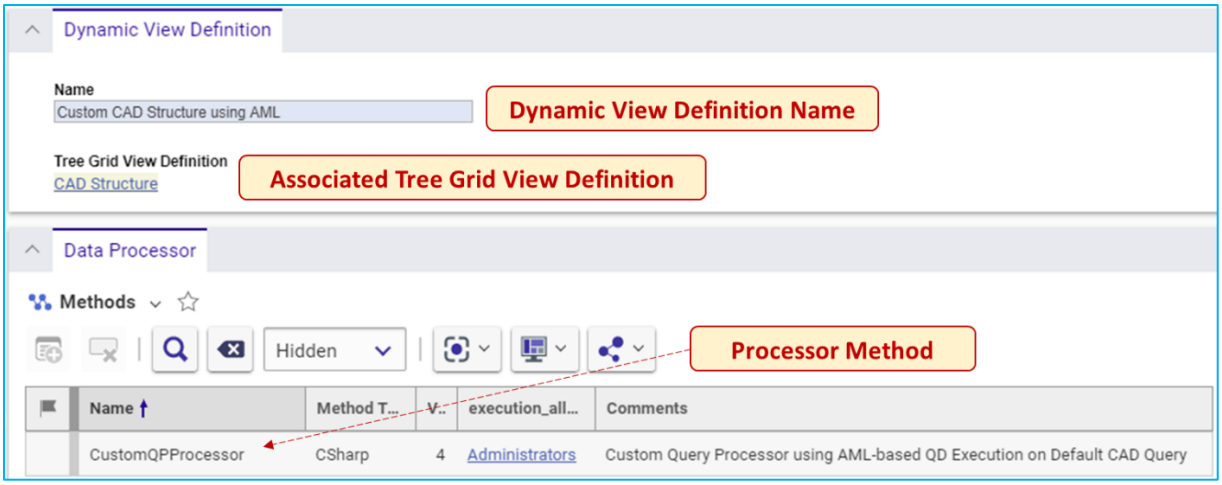

Figure 47. Dynamic View Definition

The execution of an alternate Query Processor starts with the assigned, server-side Method. This is referred to as the 'Data Processor Method'. Each Dynamic View Definition requires a 'Data Processor Method' to be assigned. The default Dynamic View Definition uses a default 'Data Processor Method' called by the method "dpn_ItemQueryProcessorMethod".

Note: The Default Query Processor will only work with the Default Query Definition or with Query Definitions that use the same Base Query for context ItemType: CAD

The existence of a Dynamic View Definition Item triggers the application of the configured Tree Grid View Definition and its associated Query Definition as explained in prior sections. The following sections describe the execution of the Query Processor and how to create a custom implementation.

6.2.4 Create the Data Processor Method

The example described in this section relegates the bulk of the Query Processing logic to a separate DLL which is linked with the Aras Innovator Server. This separation of code is not necessary, but because of the amount of software that may apply to custom query *processing* this method of implementation is easier to manage. The actual Data Processor Method therefore is much smaller and is used mostly to validate input and make the necessary calls into the custom DLL in this case.

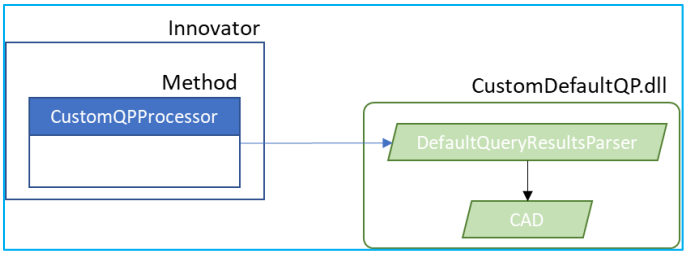

Figure 48. Example Software Architecture

The following is an example of the `CustomQPProcessor` Method which is used to execute the Default Query Definition, pass the results to a custom DLL for processing, and generate the view data used for the 3D Viewer – Product Occurrences. The example is described here.

```
1 //MethodTemplateName=CSharp:Aras.Public.Events.EventHandler<Aras.DynamicModelViewer.DataModel.DataProcessorArgs>;

2 if (eventArgs == null || !eventArgs.IsValid())
    return;

3 // Get query definition item
  Item qdItem = Aras.DynamicModelViewer.DataModel.Helpers.QueryDefinitionHelper.Instance.GetQueryDefinitionItem(eventArgs.QueryDefinitionId);

  // Execute query definition and get result
4 var qdResult = Aras.DynamicModelViewer.DataModel.Helpers.QueryDefinitionHelper.Instance.GetQueryDefinitionResultModel(qdItem, eventArgs.ItemId,
  eventArgs.QueryDefinitionParams);

5 // Custom Query Processing Class
  CustomDefaultQP.DefaultQueryResultsParser defQryResParser = new CustomDefaultQP.DefaultQueryResultsParser();

6 // Process the query response
  var result = new Aras.DynamicModelViewer.DataModel.QueryProcesingResult();
  result.SetProductOccurrenceList(defQryResParser.processQueryResults(qdResult));

7 // Return the results
  eventArgs.SetQueryProcessingResult(result);
```

Figure 49. Example Data Processor Method

1. This initial comment must be the first line in the Method. It is used to identify the Method template to be used.
2. The `EventArgs` object is passed into, and available, to this Method. This object contains The Query Definition ID, ID of the Item being viewed, the Query Definition Parameters map, and will contain the results created by the Query Processor.
3. Retrieves the Query Definition used for this Query Processor. It is used to execute and return the results in #4
4. Execute the Query Definition using the given Item ID and the query parameters used. These results will be processed in step 6.
5. This example includes a custom DLL used for the query processing logic, with the main class – `CustomDefaultQP.DefaultQueryResultsParser`. This software, explained below, is used to parse the query results and construct the response used to populate the 3D Dynamic Viewer.
6. The results of the processing – Product Occurrence List – is constructed by the custom DLL and stored in the `QueryProcessingResults` Object…
7. …which is then returned to the `EventArgs` Object passed into the Method (see Help files for API documentation)

6.2.4.1 Processing Results from Query Execution

The custom Query Processor example processes the results returned from the execution of the Query Definition. To illustrate this process this section will use the example query results shown in Figure 50 and discuss each relevant XML Element when processing.

```
<Result>
  <Item alias="CAD">
    <id keyed_name="R-ARM-43">…</id>
    <Additional Item Properties…>
    <Relationships>
      <Item alias="CAD Structure">
        <Relationships>
          <Item alias="CAD Instance">
            <Additional Item Properties…>
            <transformation_matrix>…</transformation_matrix>
          </Item>
          <Item alias="CAD">
            <id keyed_name="R-ARM-44">…</id>
            <Additional Item Properties…>
            <Relationships>
              <Item alias="CAD Structure">
                <Relationships>
                  <Item alias="CAD Instance"/></Item>
                  <Item alias="CAD">
                    <id keyed_name="R-ARM-45">…</id>
                    <Relationships>
                      <Item alias="File">
                        <Relationships>
                          <Item alias="fr_Representation">
                            <Relationships>
                              <Item alias="fr_RepresentationFile">
                                <Relationships>
                                  <Item alias="File_1">
                                    <id keyed_name="J3 Spindle.scs">…</id>
                                    <filename>J3 Spindle.scs</filename>
                                  </Item>
                                </Relationships>
                              </Item>
                            </Relationships>
                          </Item>
                        </Relationships>
                      </Item>
                    </Relationships>
                  </Item>
                </Relationships>
              </Item>
            </Relationships>
          …
```

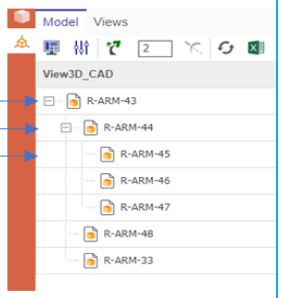

Figure 50. Query Results Example

Figure 50 shows the partial XML result data when executing the Default (Base) Query Definition against a sample Assembly shown at the right. In this example, the root Assembly – `R-ARM-43` – has one sub-Assembly – `R-ARM-44` – and two child Components – `R-ARM-48`, `R-ARM-33`. Assembly `R-ARM-44` has three Components – `R-ARM-45`, `R-ARM-46`, and `R-ARM-47`. The diagram highlights key XML Elements and Attributes in the XML results that are necessary when processing this data. Note also the XML Element - `<Additional Item Properties>` is notional – it represents additional Properties included for the associated XML Element that were left out of the example for simplicity.

When implementing a Query Processor, it is necessary to process the result set such as displayed above and extract the required information. The following sections identify some points to consider for this purpose.

6.2.4.2 Aliases

Each Item in the results is represented by an `alias` attribute matching the alias value provided in the Query Definition. This is important, because the alias identifies the specific Query Item that matches the associated Item (see Table 1). There can be multiple Query Items in a Query Definition that refer to the same ItemType. The alias is the only way to distinguish the resulting query information.

6.2.4.3 Data Model Object

It is useful to collect the information extracted from the Query Results in a separate object; which can then be used to construct the Product Occurrence list. In the example shown in Section 6.2.5, the Class `CAD` was included to store key information extracted from parsing the Query Results.

6.2.5 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 7.1.

Note: 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.

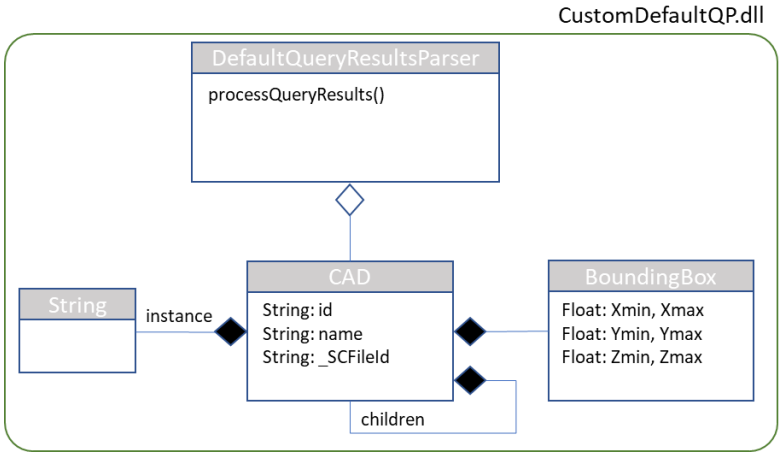

Figure 51. Example Query Processor DLL Class UML Diagram

6.2.5.1 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.

6.2.5.1.1 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

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 3.6). 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 51, 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.

Note: 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.

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
      _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 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.

6.2.5.1.2 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.

```
1.  private void build(ProductOccurrenceInstance parent, CAD c)
    {
2.    foreach(String xFormStr in c.Instances)
      {
        ProductOccurrenceInstance poInst = new ProductOccurrenceInstance();
3.      poInst.Attributes.Add(new ProductOccurrenceAttr("QUERY ITEM REF ID", c.QryRefId));
        poInst.Attributes.Add(new ProductOccurrenceAttr("ITEM ID", c.ID));

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

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

6.      if (xFormStr.Length > 0)
          poInst.TransformationMatrix = xFormStr;

7.      poInst.Name = c.Name;
8.      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);
        }

 9.     if (c.Children.Count == 0 && c.SCFileID == null)
```

```
            poInst = null;  // Invalid Part
10.       else (parent != null)
            parent.Children.Add(poInst);

          // Recurse through child hierarchy
11.       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. See Figure 52.
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:
   a. Query Item Reference ID. The reference ID is used for all Product Occurrence Instances and is extracted from the Query Results
   b. 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.

Note: 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 3.6), there may be several root Product Occurrence Instances. See the full example in Section 7.1.

Warning When upgrading from 12.0 SP7, 8 or 9 to 12.0 SP10+ it will be necessary to add the call to the `ProductOccurenceBase.SetAsRoot()` method.

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 Viewer (Section 6.2.6). 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. The name used for the Product Occurrence. It's helpful to either use the document number or name of the CAD Item.

8. 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**.
9. 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.

Note:	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.

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

6.2.5.2 Product Occurrence List – View Data

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

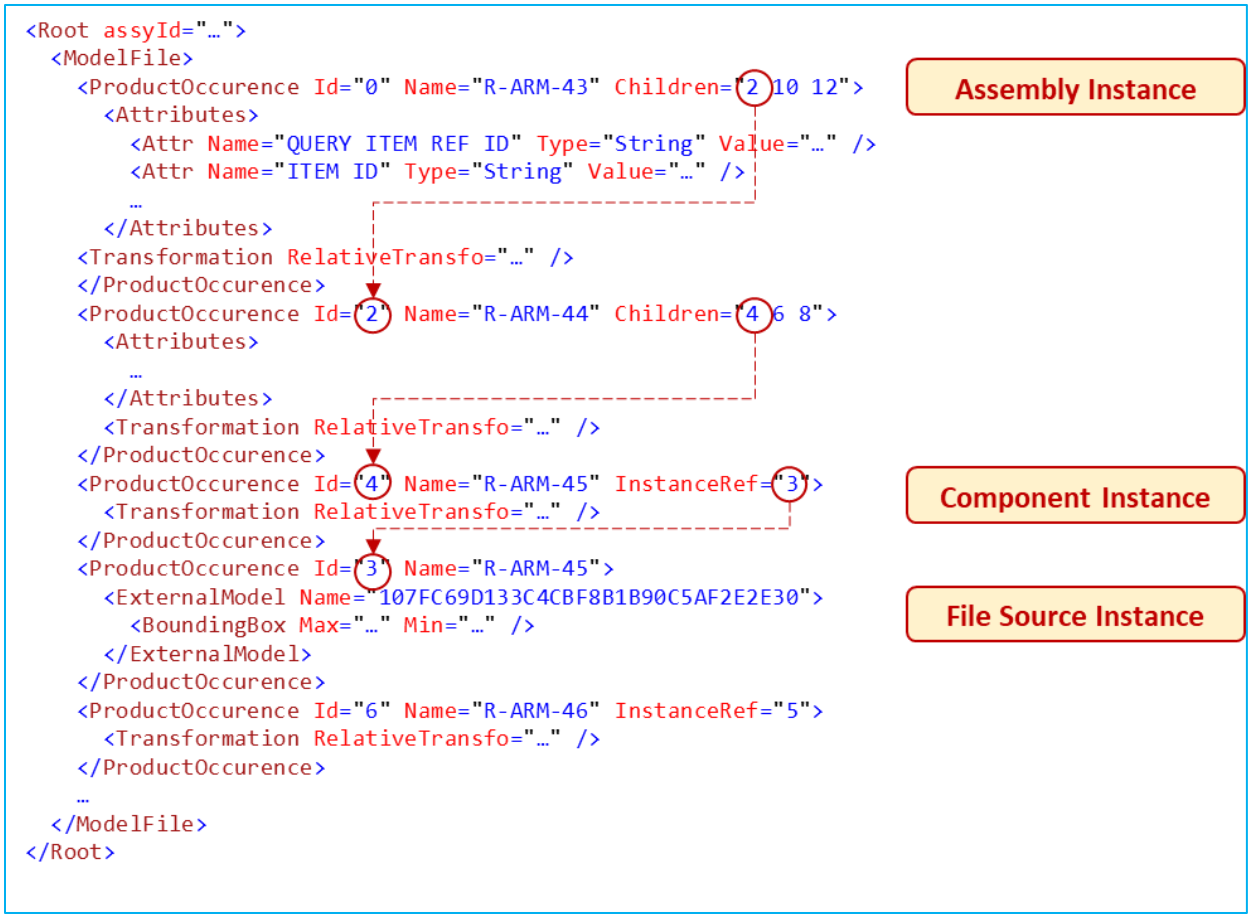

Figure 52. View Data XML Example

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 6.2.5.1.2.
- 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.

6.2.6 Rendering Configurations

Rendering Configurations are used to define alternate rendering properties (color, transparency) to apply to 3D geometry in the Dynamic Viewer. The intent is to isolate/highlight certain 3D components or represent some aspect of related Items using color to differentiate from other 3D components. Each of these Rendering Configurations are accessible in the Dynamic Viewer via the View Modes dropdown on the viewer toolbar.

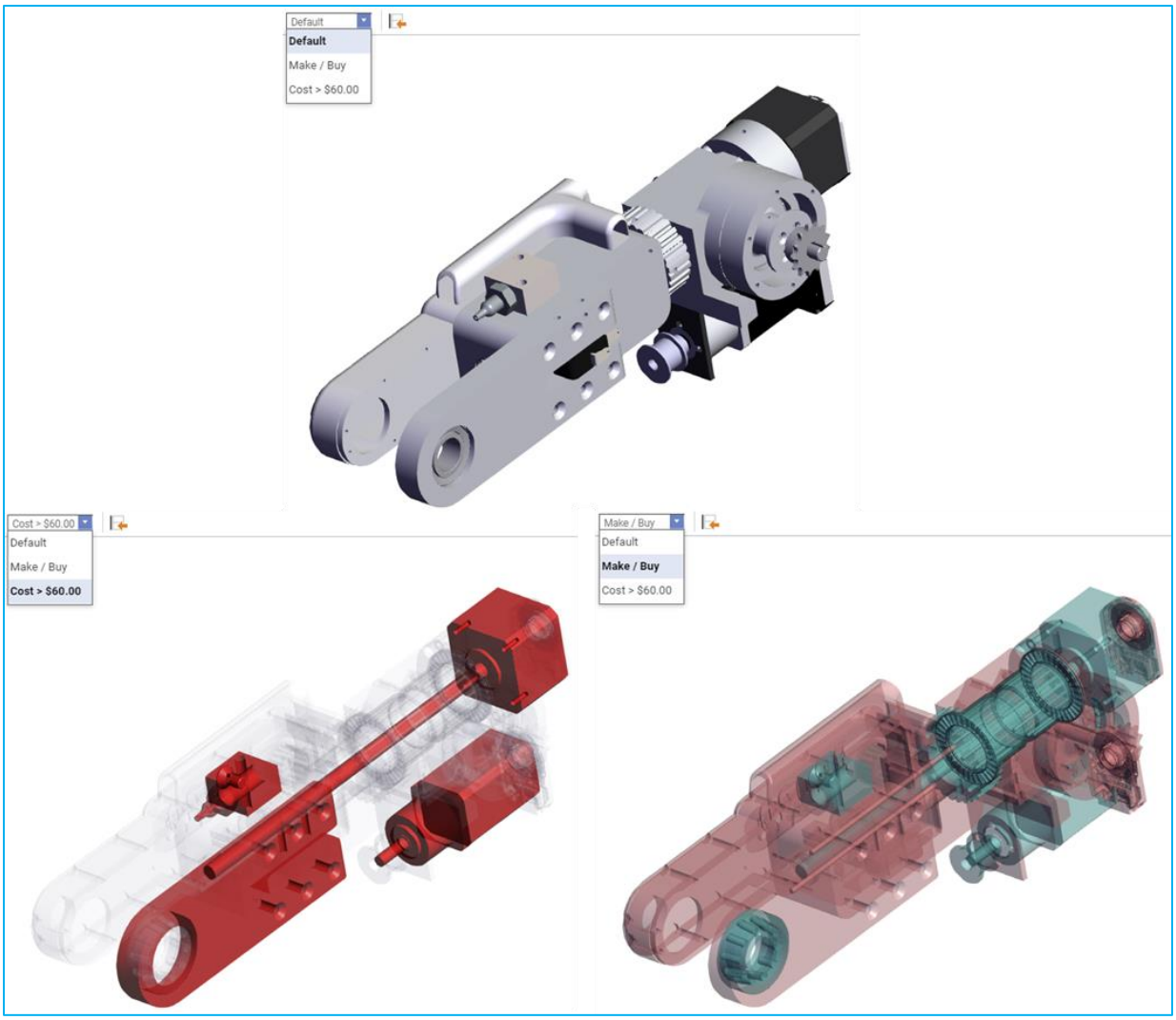

Figure 53. View Modes

Figure 53 shows sample View Modes created by the inclusion of Rendering Configurations (see code related to #5 in Section 6.2.5.1.1). The top diagram shows the 3D geometry rendered using the default colors as defined when the CAD data was imported. The diagram on the lower left shows a View Mode that applies alternate colors to all 3D Components associated with Parts that have a cost greater than $60. The diagram on the lower right shows a View Mode that renders parts in Coral that are manufactured and light Green that are purchased. Note the use of transparency (Opacity) in each example.

Each of the bottom two examples were created using two Rendering Configurations with a 'Name' that is displayed as a View Mode in the 3D Dynamic Viewer. When creating Rendering Configurations, the name is what distinguishes them. There can be multiple Rendering Configurations added to the same set of Product Occurrences; each must have a unique Name.

The following code snippet shows an example of creating the Rendering Configuration shown in the bottom right above:

```
1. ProductOccurrenceRenderingConfiguration rConfigMakeBuy = new
     ProductOccurrenceRenderingConfiguration();
2. rConfigMakeBuy.SetColor(c.isManufactured() ? Color.LightCoral :
                           Color.LightSeaGreen);
   rConfigMakeBuy.Opacity = .5;

3. rConfigMakeBuy.Name = "Make / Buy";


4. poInst.RenderingConfigurations.Add(rConfigMakeBuy);
```

Rendering Configurations are defined using the `ProductOccurrenceRenderingConfiguration` class and are added to each Product Occurrence Instance.

Note: It is required that all Product Occurrence Instance objects have a Rendering Configuration object added if View Modes are going to be used within a Query Processor.

Each unique View Mode must use the same string name value for the Name Property. Thus, a View Mode represents the collection of all Rendering Configuration Objects with the associated Name.

1. A Rendering Configuration Object is created.
2. Logic included in the Data Model Object determines whether the associated CAD Item is related to a manufactured Part. Note that when constructing Query Definitions for a Query Processor, additional Properties may be needed to be used solely for Rendering Configurations (View Modes).

   The default color is black, and the default opacity is 0 (completely hidden). As mentioned, it is important to add a Rendering Configuration to each Instance to prevent default settings from applying.

Note: Use of 0 transparency does not prevent parts from being selected in the viewer. If an Opacity property is set to 0, the geometry will still be loaded and it will be selectable even though the geometry isn't shown.

3. A unique Name is applied. It will be same for all Product Occurrences returned for the View Data.
4. The Rendering Configuration object is added to the Product Occurrence Instance object.

6.3 Deploying a Query Processor

If the implementation of Query Processing uses the approach in this section – use of a separate DLL – the following steps can be used to compile the DLL and integrate it with the Aras Innovator Server.

Note: The full implementation of the Custom Default Query Processor referenced in this Section is available in Section 7.1.

At a high level, the steps include:

1. Implement the DLL as defined in previous sections.
2. Follow the Steps in Section 2 to install the Dynamic Viewer.
3. Build and deploy the DLL.
4. Define the Data Processor Method.
5. Build Query Definition(s), Tree Grid View Definition(s), and Dynamic View Definition(s) and link with the Data Processor Method created.

6.3.1 Build and deploy the DLL

The example described in this section was implemented using Microsoft Visual Studio with an output type of *Class Library* and a target Framework using *.Net Standard 2.0*.

Note: DLLs integrated with Aras Innovator need to be signed.

The DLL must also be compiled by linking with the `Aras.DynamicModelViewer.DataModel` and `Aras.Server.Core` libraries which are included with the Dynamic Model View Install and with the standard Innovator installation respectively. Note also that the Rendering Configuration classes use the `Color` class as defined in `System.Drawing`. The resulting DLL needs to be stored within the `<Aras Innovator Install Dir>/server/bin` directory. Once the DLL is stored, update the `method-config.xml` file in the `<Aras Innovator Install Dir>/server` directory to include a reference to the DLL. For example:

```
<MethodConfig>
    <ReferencedAssemblies>
        ...
        <name>$(binpath)/Aras.DynamicModelViewer.DataModel.dll</name>
        <name>$(binpath)/Aras.DynamicModelViewer.QueryProcessor.dll</name>
        <name>$(binpath)/CustomDefaultQP.dll</name>
        <name>$(binpath)/CustomPartQP.dll</name>
    </ReferencedAssemblies>
```

Figure 54. Method Config XML Update

6.3.2 Define the Data Processor Method

Create the Data Processor Method as described in the beginning of Section 6.2.5. Make sure that the appropriate level of validation is included. Note also that Query Parameters can be used to *parameterize* the custom Query Processor. Query Parameters are defined within the Query Definition and enabled in the Tree Grid View Definition. Parameters used within a Query Definition can be applied to the Where Conditions within the Query Definition, used for a Query Processor, or both. See Section 5.1.3.4 for a description of creating a Query Parameter. Query Parameters are accessible using the `EventArgs` variable in the Data Processor Method.

6.3.3 Create the Query/Tree Grid View/Dynamic View Definitions

Define the Query and View Definitions as described in this section. Note that related Item data can be added to the Query Definition and displayed in the Tree Grid View without being processed by the Query Processor. Doing so uses the 3D View as a navigation aid to access related Item data. When adding ItemTypes to a Query Definition, make sure you add the `ID` Property so the View context menu will be enabled by default (see Section 5.2.6). Likewise, related Item data can be added to a Query Definition to be processed solely by the Query Processor. Doing so can provide additional information when/if Rendering Configurations are included.

6.3.4 Help files

The Installation files for the Dynamic Viewer contain a directory `/QueryProcessorAPI` with API help files describing the classes/methods of the API.

7 Appendix

7.1 Sample Custom Default Query Processor

This section includes the complete sample code discussed in Section 6.2.5. The following Class represents the main class used for custom Query Processing. It contains the `processQueryResults` method discussed in Section 6.2.4 and the build method discussed in Section 6.2.5.1.2.

```
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,
          poList.Add(poInst); // so 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 buld from</param>
    private void build(ProductOccurrenceInstance parent, CAD c)
    {
      // Creat a Product Occurrence for each instance of the given CAD
      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));

        // 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 ***

        if (xFormStr.Length > 0)
          poInst.TransformationMatrix = xFormStr;

        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 6.2.5.1.1.

```
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 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("1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1");
    } // 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
      _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 Structure")
          _processCADStructure(child);
        if (child.QueryItem.Alias == "File")
          _extractFileInfo(child);
      } // foreach
    } // processCAD(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<String> curInstances = new List<string>();
      CAD nextCAD = new CAD();

      // 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)
            curInstances.Add(tmpXForm);
          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 Id of the CAD Query Response Item
    /// </summary>
    internal String ID { get; private set; }

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

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

    /// <summary>
    /// Returns the list of Transformation String values with the
    /// associated CAD Instance Query Response Items
    /// </summary>
    internal List<String> Instances { get; private set; } = new List<string>();

    /// <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
```