Aras 3D Visualization 26

Administrator Guide

Document #: D-008026

Last Modified: *04/18/2023*

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:

You can send comments to us in the following ways:

Email:
TechDocs@aras.com
Subject: Aras Product Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Technical 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.

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` text. It may represent text you type or data you read.
`Yellow highlight`	Code with yellow highlight is used to draw attention to the code that is being indicated in the content.
`Yellow highlight with red text`	Red color text with yellow highlight is used to indicate the code parameter that needs to be changed or replaced.
Italics	Reference to other documents.
Note:	Notes contain additional useful information.
Warning	Warning contains 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 Administration Guide provides instructions for configuring Aras 3D Visualization (3DV) and all its features.

1.1 Objective

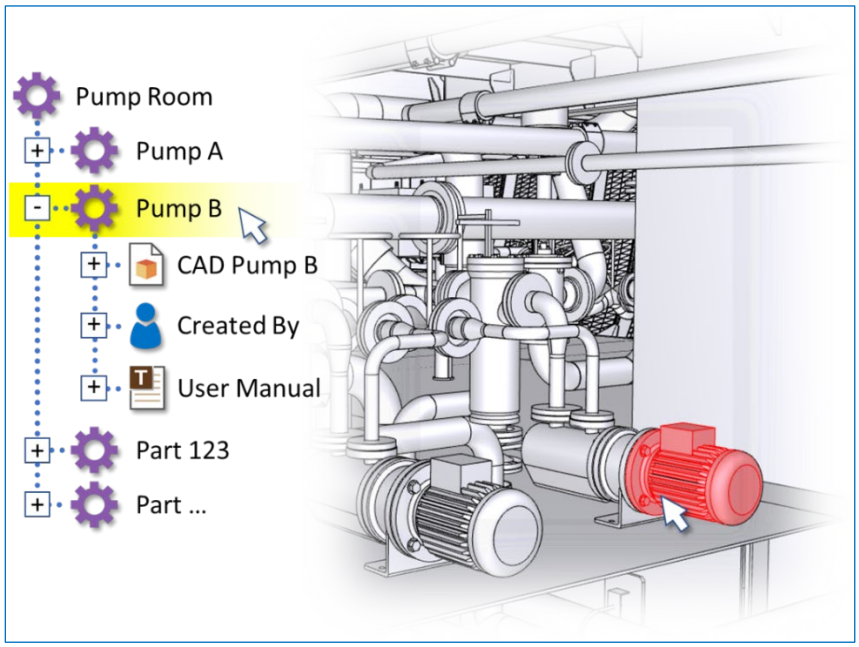


Figure 1.

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 (DPN).

1.1.1 Enabling Control of What Content is Displayed

Aras Innovator has Query Builder and Tree Grid View, which provide an end-user with a graphic editor to define an ItemType query and bind the elements of that query to a Tree Grid View. The concept is like a Model and View respectively in the 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, like Graph Navigation Views. 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 3.7. 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 or define 3D color and transparency used in a 3D View.

1.1.3 3D Visualization as a Navigation Tool

The Dynamic Product Navigation (DPN) phrase 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.

1.2 Known Limitations

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 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. The support of alternate data models exists as the ability to implement 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 26 – Installation Guide*, available in the **Documentation** folder of the 3D Visualization CD Image, which subscribers can obtain from the **Aras 3D Visualization** folder on the Aras FTP site.

2.1.2 Using Legacy View Files

Aras Innovator provides backward compatibility in both the data model and viewing features. With the current Viewer, users can convert legacy HWF view files to the new Stream Cache Single (SCS) file format using an automated asynchronous process. The **CAD** Items can either use the HWF, SCS or SCZ 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 of a given Preference Item.

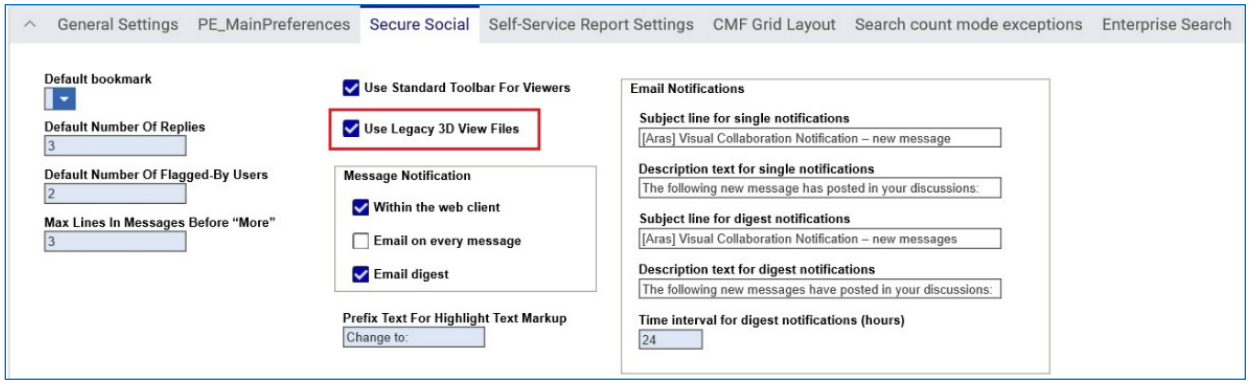


Figure 2.

The process of viewing and converting files is 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 **Part** Item with a related **CAD Document** Item that includes a legacy view file (HWF). The following figure describes the process.

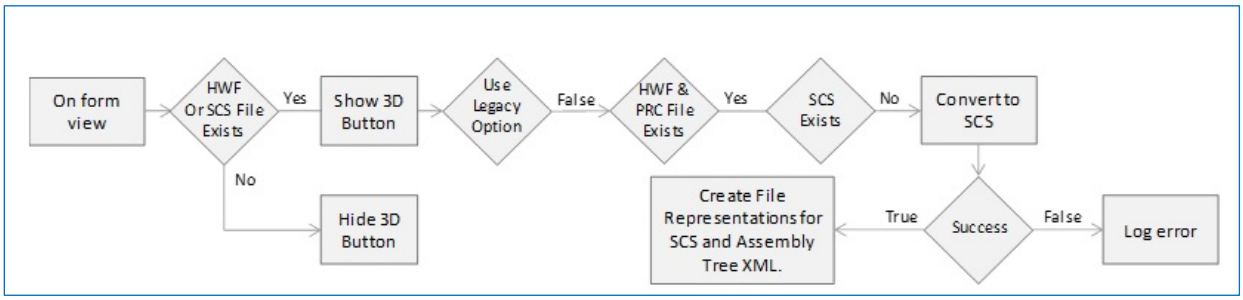


Figure 3.

The existence of an HWF, SCS file determines whether the 3D Viewer can be opened. If it can, a button appears on the sidebar enabling users to open the Viewer. If either of these files does not exist, the user is unable to open the 3D Viewer. The **Use Legacy 3D View Files** setting value of the given Preference Item 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 **Use Legacy 3D View Files** 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 the default Dynamic Viewer configuration. For installation details, refer to the *Aras 3D Visualization 26 – Installation Guide*.

The following section explains the command arguments. Any outputs not required by a given 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 the **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'" />
```

Note: Due to differences between the Windows and Linux file systems, it is required to use OS-specific path separators in paths. Because the Linux file system is case-sensitive, there is a significant difference between these file names: **./path/to/file.xml** and **./path/to/File.xml**. For more information about cross-platform development, refer to the *2.3 Cross-platform development* section in the *Aras Innovator 26 - Programmer's Guide*.

2.2.2 Customization Options

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

Command-line Argument	Required	Description
`--sc_compute_bounding_boxes`	Yes	Positions the camera in the Viewer and prioritizes the rendering order of an assembly. The bounding box information will be extracted from a native CAD file.

Command-line Argument	Required	Description
`--input_pdf_template_file`	No	Required only if generating viewable PDF files for an Item.
`--output_pdf`	No	Required only if generating viewable PDF files for an Item.
`--output_png`	Yes	Produces a thumbnail image for an Item.
`--output_scs`	Yes	Enables the Monolithic and Dynamic Viewers.
`--output_xml_assemblytree`	Yes	Maps 3D component geometry.
`--output_prc`	No	Required only if generating PRC files for an Item that may be used for industry standard archival purposes.
`--background_color`	Yes	Sets the background color for thumbnail images (png) and the 3D PDF. The default is black. However, a background color other than white may affect the display of the image in the Item Form.
`--output_logfile`	Yes	Sets a name of a log file where the HOOPS Converter will write error and warning messages.
`--drawings_mode`	No	Controls what to import from drawing files: • `0`: only 3D • `1`: only drawings • `2`: both 3D and drawings The default is `1`.

Please see the included documentation for a complete description of available parameters and their functions here: **\Aras 3D Visualization 14.0.4 CD Image\Documentation\HOOPS_Communicator_2023_U3_98Guide\Index.html**.

2.3 Configuring Streaming Viewer

The installation process creates the default Streaming Viewer configuration. For installation details, refer to the Aras 3D Visualization 26 – Installation Guide.

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

2.3.1 Out of the Box Setup

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

```
<AssemblyCommand arguments="--sc_compute_bounding_boxes 'All' --
input_pdf_template_file 'C:\Aras\14SP10\HOOPS
Converter\Templates\Blank_Template_L.pdf' --output_pdf
'%filepath%\%filename%.pdf' --output_png '%filepath%\%filename%.png' --
output_png_resolution '150x150' --output_xml_assemblytree
'%filepath%\%filename%.xml' --output_prc '%filepath%\%filename%.prc' --
background_color '1.0, 1.0, 1.0' --output_log file '%filepath%\%filename%'"
streamingEnabled="True"/>
```

Note: Due to differences between the Windows and Linux file systems, it is required to use OS-specific path separators in paths. Because the Linux file system is case-sensitive, there is a significant difference between these file names: **./path/to/file.xml** and **./path/to/File.xml**. For more information about cross-platform development, refer to the *2.3 Cross-platform development* section in the *Aras Innovator 26 - Programmer's Guide*.

Warning The Streaming Viewer currently cannot be deployed in a cloud environment.

The HOOPS Server must be deployed with networked file access to a single vault containing view files for rendering.

Note: Only one Streaming Viewer can be installed on one machine at a time.

2.3.2 Customization Options

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

Command-line Argument	Required	Description
`--sc_compute_bounding_boxes`	Yes	Positions the camera in the Viewer and prioritizes the rendering order of an assembly. The bounding box information will be extracted from a native CAD file.
`--input_pdf_template_file`	No	Required only if generating viewable PDF files for an Item.
`--output_pdf`	No	Required only if generating viewable PDF files for an Item.
`--output_sc`	Yes	Full path of SC Model to generate. Warning: This option will recursively delete all existing files in the target directory.
`--sc_create_scz`	Yes	If set, then generated SC models will be .scz files. Default: False
`--sc_compress_scz`	No	If set, then generated .scz files will be compressed. Default: True
`--output_png`	Yes	Produces a thumbnail image for an Item.
`--output_xml_assemblytree`	Yes	Maps 3D component geometry.
`--output_prc`	No	Required only if generating PRC files for an Item that may be used for industry standard archival purposes.
`--background_color`	Yes	Sets the background color for thumbnail images (png) and the 3D PDF. The default is black. However, a background color other than white may affect the display of the image in the Item Form.
`--output_logfile`	Yes	Sets a name of a 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 here: **\Aras 3D Visualization 14.0.4 CD Image\Documentation\HOOPS_Communicator_2023_U3_98Guide\Index.html**.

3 Aras 3D Viewers

3.1 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.

3.1.1 CAD Data Model

Before reviewing the Dynamic Viewer features, it is 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 **CADFile** conversions using the following diagram.

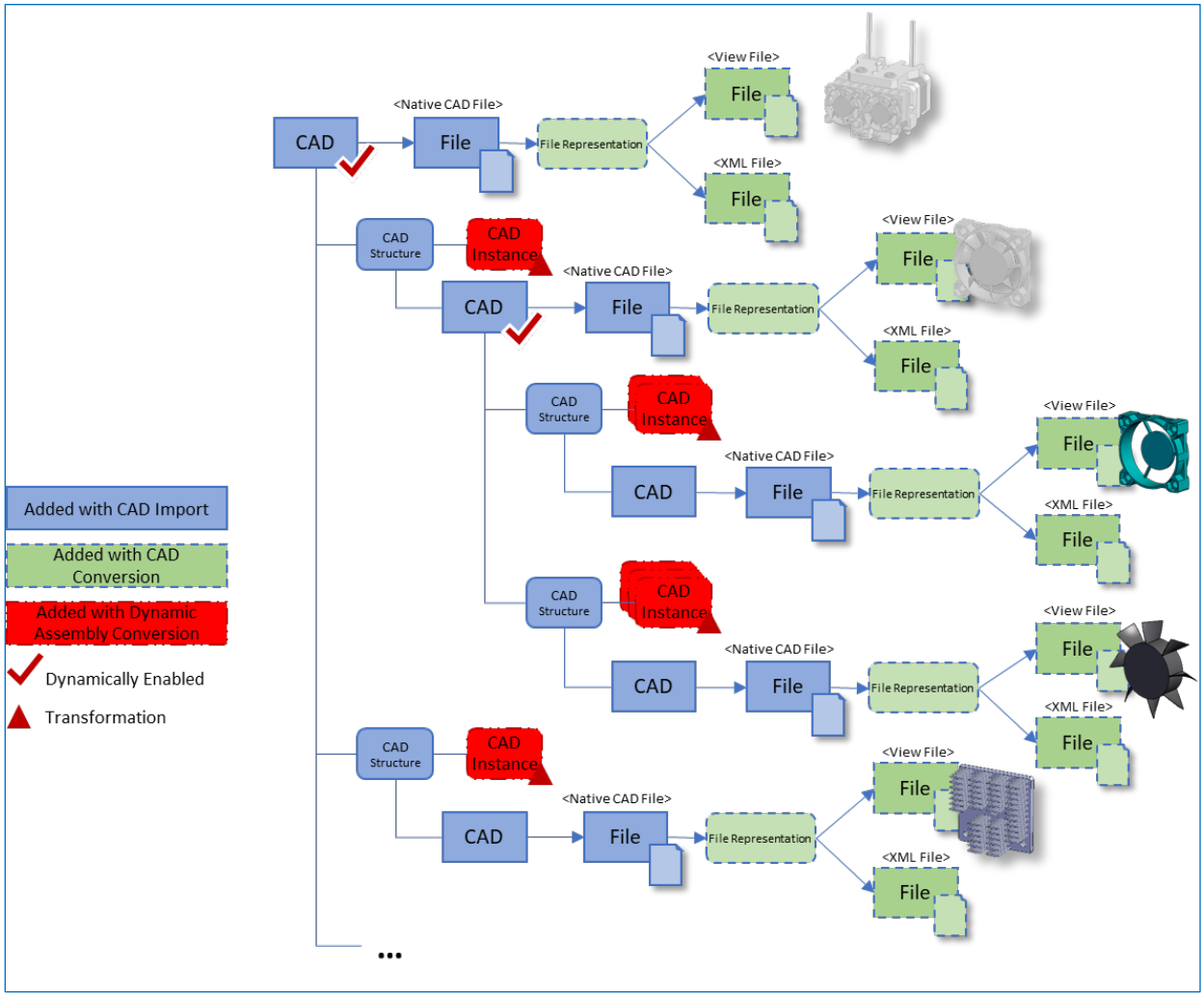


Figure 4.

The **CAD** Items and **CAD Structure** Relationship Items are typically created by a CAD Connector when CAD native files are checked into the Aras Innovator. A **CAD** Item represents a single 3D CAD assembly, part, or 2D drawing. For this document, **CAD** Items representing 3D CAD assemblies and parts are assumed. In this case, the native CAD file is attached (via the **File** ItemType property) to each **CAD** Item. If this **CAD** Item represents an assembly, all related subassemblies and 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 is the process of generating alternate versions of native CAD data. It 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 in a **CAD** Item or its related Items. An XML file is also created and stored. This XML file is generated from the conversion process and used to identify all the instances of parts and subassemblies included in 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 SCS view and XML files are stored as related Items attached to a **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 RelationshipTypes, as shown in Figure 5.

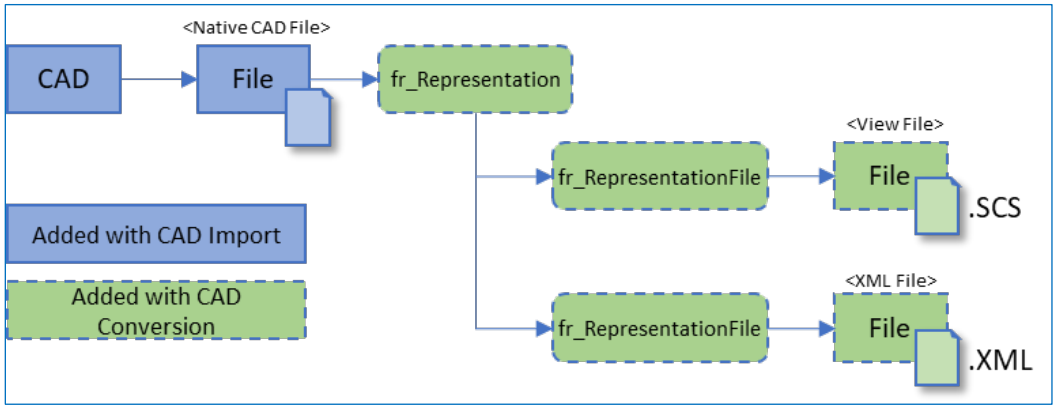


Figure 5.

For the Dynamic Viewer, the conversion process appends the **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, the added **Dynamic Enabled** Boolean property is set to **true** for a **CAD** Item with an assembly 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 a native CAD assembly file.

3.1.2 Monolithic vs Dynamic Viewer

The best way to describe the Dynamic Viewer is to compare it against the Monolithic Viewer. This section compares the existing features of these Viewers. Figure 6 shows the buttons on the sidebar of a **CAD** Item form that open the Viewers.

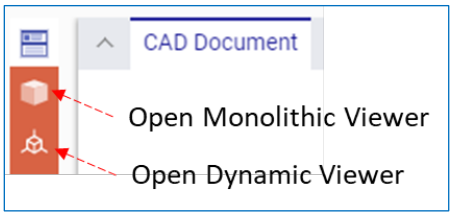


Figure 6.

1 Aras Innovator CAD Conversion supports CAD files from multiple CAD systems. These are configured using Conversion Rules; see section 2.2.

3.1.3 Monolithic Viewer Process Overview

The Monolithic Viewer displays a single SCS view file attached to a corresponding **CAD** Item; see Figure 5. If it is an assembly view file, the monolithic view file includes the 3D component geometry for all subassemblies and parts included in this assembly at the time the CAD file was checked in. In essence, it is a static view of the assembly because it will not include any geometry changes to subcomponents made after the assembly view file was created. Figure 7 shows the process of rendering the view file in the Monolithic Viewer.

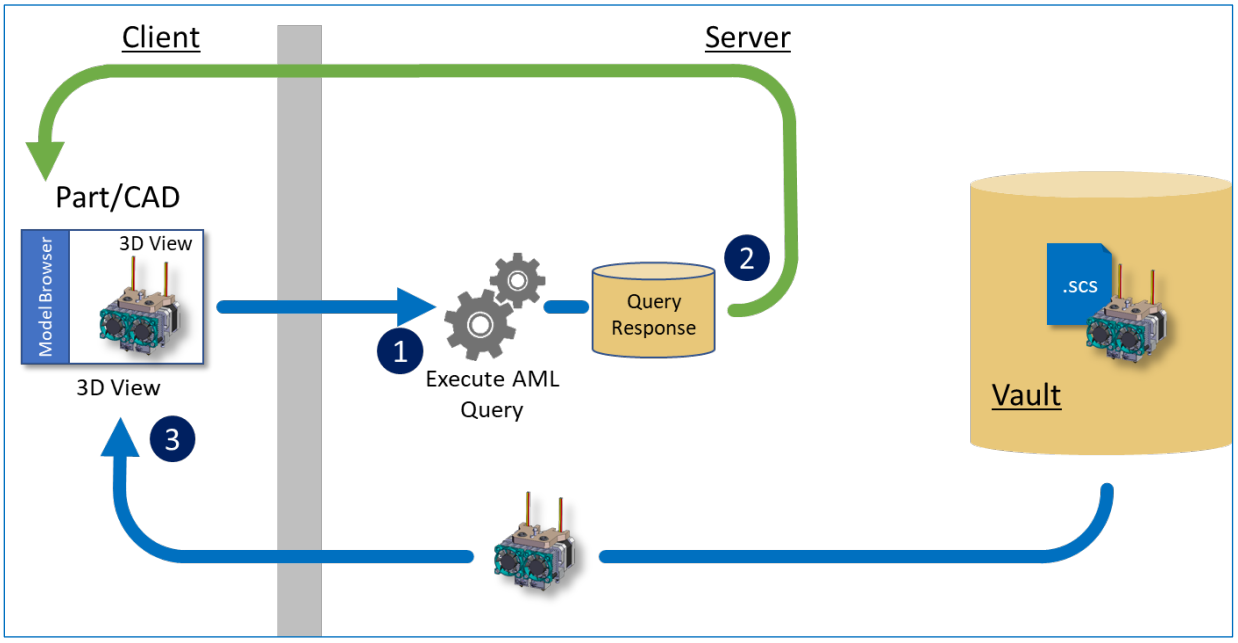


Figure 7.

When the Monolithic Viewer is opened, a static AML query is executed to retrieve the multi-level CAD structure of an opened **CAD** Item. The results are displayed in a tree user interface called the simple Model Browser of the Viewer. A request is made to load a 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 simple Model Browser with the associated 3D components in the 3D View scene. This provides selection synchronization between the Model Browser and the 3D View.

3.1.4 Dynamic Viewer Process Overview

The Dynamic Viewer is used only for assemblies and displays SCS view files as determined by the results returned from the execution of an associated Query Definition (QD). Unlike a monolithic view file for an assembly, the Query results for a dynamic Query target the view files for assembly parts and the instance and transformation data required to properly position each file given the returned assembly hierarchy. Figure 8 shows the process of rendering the view file in the Dynamic Viewer.

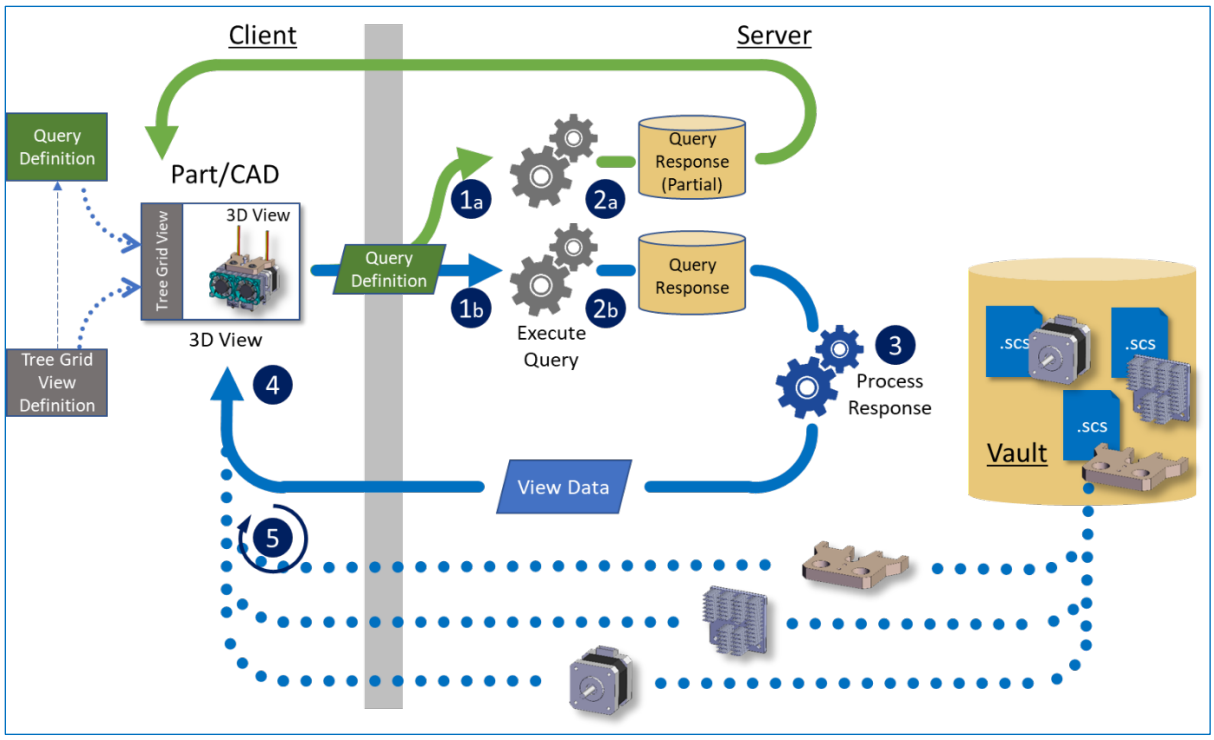


Figure 8.

The Dynamic Viewer consists of two main components: Tree Grid View (TGV) and the 3D View. The TGV displays the results of the associated QD 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** Item from which the Dynamic View was opened.

Upon refreshing the View, the system executes the associated QD in two simultaneous operations. The first executes the Query to populate the TGV. The second executes the Query and processes the complete response.

The TGV uses the partial response based on the configuration of the associated TGV Definition.[2]. By default, the TGV is lazy-loaded: only a portion is returned to a client and displayed. The full response is processed to generate the view data so that all component parts are identified.

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 files in the Aras Innovator Vault.

2 TGV Definitions provide settings to determine the maximum number of peer elements and depth of related content. These settings can be overridden using the TGV toolbar.

The 3D Viewer processes the view data (XML) and sends subsequent requests to the Aras Innovator Vault to retrieve each of the corresponding view files to render. The 3D Viewer processes and renders the view files individually and in sequence.

3.2 Streaming Viewer

This section describes the Streaming Viewer and compares features with the Dynamic Viewer. Streaming Viewer has the same functionality as the Dynamic Viewer with improved rendering speed.[3] Streaming Viewer requires a separate Windows Service called Hoops Server.

3.2.1 CAD Data Model

Before reviewing the Streaming Viewer features, it is 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 CADFile conversions using the following diagram.

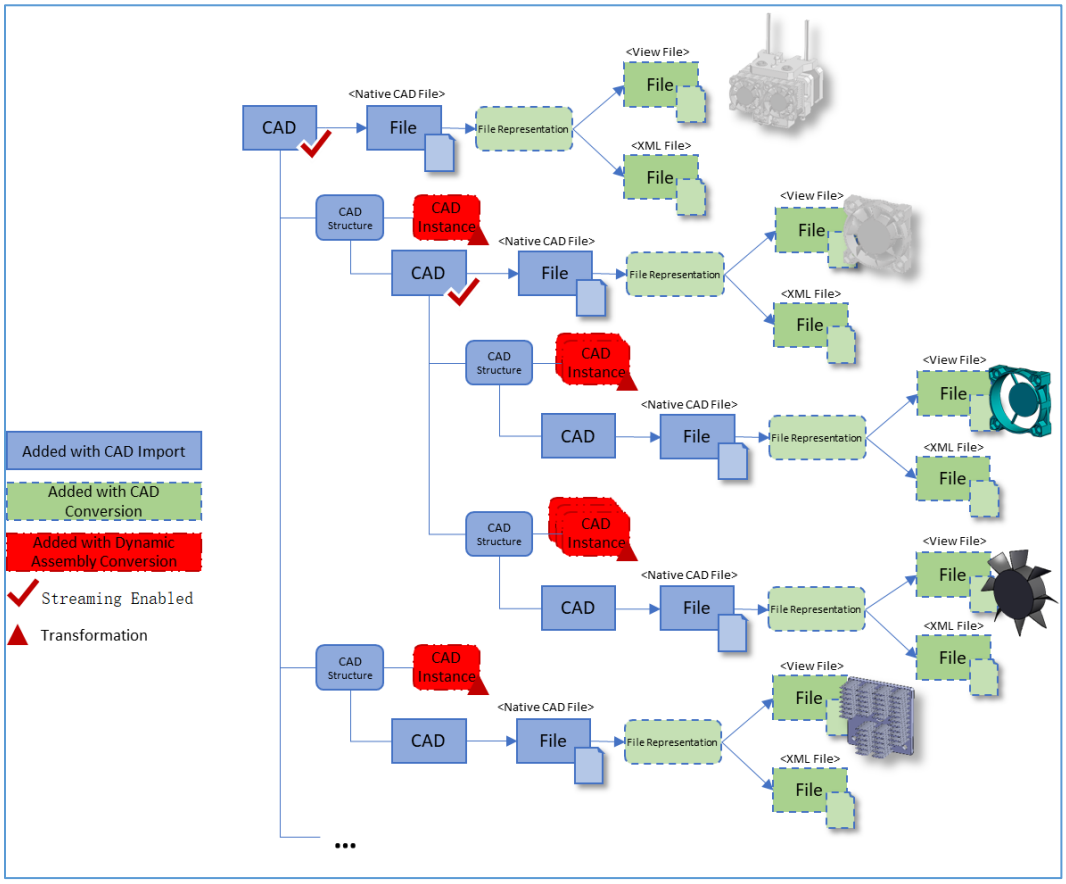


Figure 9.

The CAD Items and CAD Structure Relationship Items are typically created by a CAD Connector when CAD native files are checked into the Aras Innovator. A CAD Item represents a single 3D CAD assembly, part, or 2D drawing. For this document, CAD Items representing 3D CAD assemblies and parts are

[3] Performance results will vary based on network bandwidth, hardware, and load.

assumed. In this case, the native CAD file is attached (via the File ItemType property) to each CAD Item. If this CAD Item represents an assembly, all related subassemblies and 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 is the process of generating alternate versions of native CAD data. It is triggered by the existence of a CAD file. The Streaming Viewer Viewers render 3D component geometry stored in a Compressed Stream Cache (SCZ) 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 in a CAD Item or its related Items. An XML file is also created and stored. This XML file is generated from the conversion process and used to identify all the instances of parts and subassemblies included in a CAD assembly. This information is used to generate CAD Instances and 3D Transformations for the Streaming Viewer. The SCZ view and XML files are stored as related Items attached to a 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 RelationshipTypes, as shown in Figure 5.

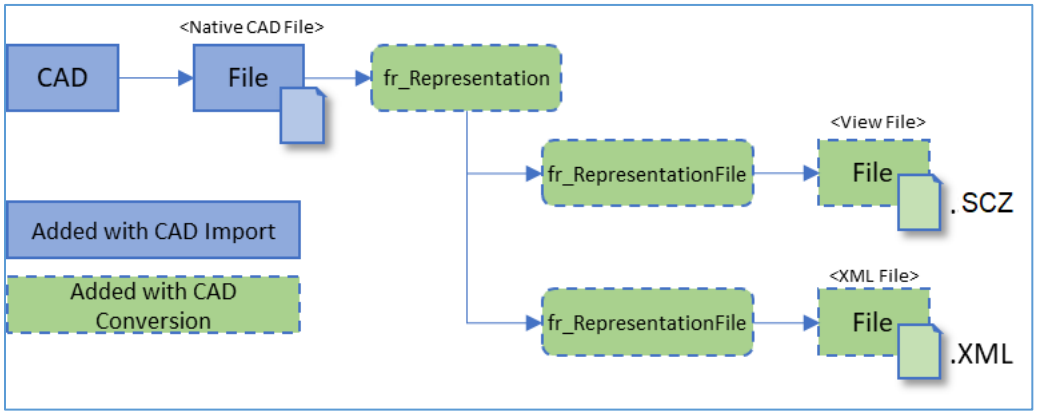


Figure 10.

For the Streaming Viewer, the conversion process appends the 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, the added Streaming Enabled Boolean property is set to true. Note that these Instances and their transformation information are created automatically by the 3D Conversion Process based on information extracted from a native CAD assembly file.

3.2.2 Streaming Viewer Process Overview

The Streaming Viewer is used only for assemblies and displays SCZ view files as determined by the results returned from the execution of an associated Query Definition (QD). Figure 9 shows the process of rendering the view file in the Streaming Viewer.

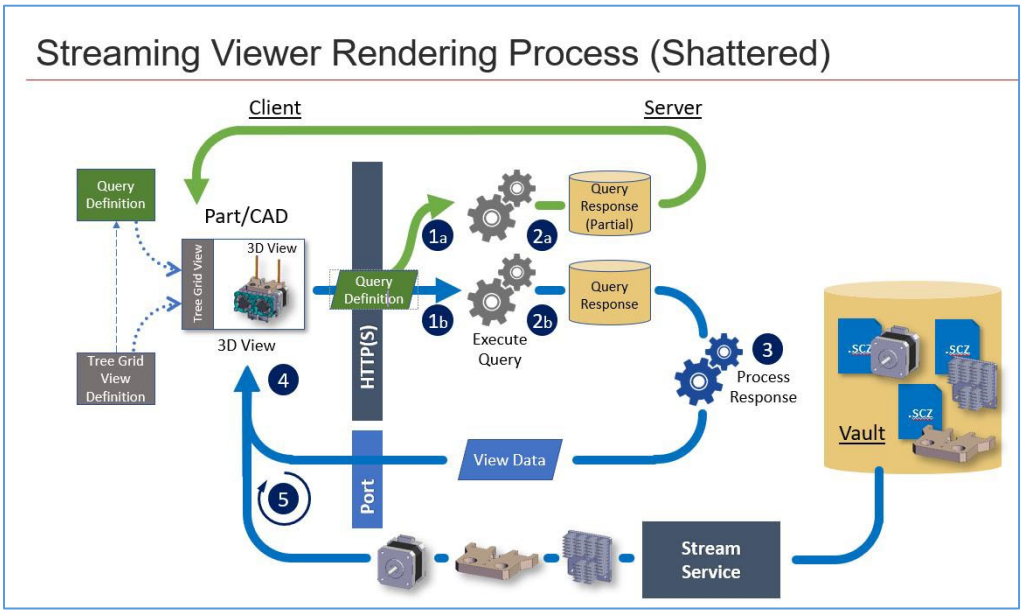


Figure 11.

Similar to the Dynamic Viewer, the Streaming Viewer consists of two main components: Tree Grid View (TGV) and the 3D View. The TGV displays the results of the associated QD as defined by the chosen Tree Grid View Definition; see section 6.2.2. 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 Item from which the Streaming View was opened.

Upon refreshing the View, the system executes the associated QD in two simultaneous operations. The first executes the Query to populate the TGV. The second executes the Query and processes the complete response.

The TGV uses the partial response based on the configuration of the associated TGV Definition. By default, the TGV is lazy-loaded: only a response portion is returned to a client and displayed.

The full response is processed to generate the view data using SCZ files.

While processing the Query results for the 3D View, XML data is constructed that identifies the assemblies, parts, part instances, in single response which has links to the respective view files in the Aras Innovator Vault. The Streaming Viewer is supported only with a single vault.

Streaming visualization relies on a server-side component – Stream Service, that analyzes the current camera position of the client viewer and optimizes which components of each SCZ file to 'stream' to the client to be subsequently rendered. The Streaming Viewer communicates with the Stream Service via a dedicated network port.

3.2.3 Streaming Viewer Rendering Types

The Streaming Viewer Rendering Types can be configured using the `HOOPS_Viewer_Renderer_Type` variable.

3.2.3.1 Client-Side Rendering

In Client-Side rendering type, the 3D model is built on the client side. The Stream Cache Server streams the 3D model data to the Hoops Web Viewer enabling the client hardware to render the 3D graphics using WebGL technology.

To select the Client-Side Rendering Type, in the **HOOPS_Viewer_Renderer_Type** variable, change the **Value** field to **Client**.

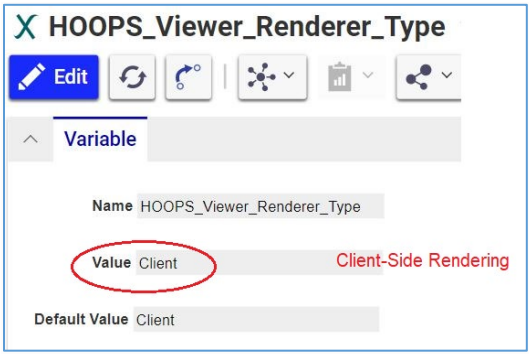


Figure 12.

3.2.3.2 Server-Side Rendering

When a Web Viewer is set up for server-side rendering, all rendering of the 3D model is performed by the Graphic Processor Unit (GPU) on the web server. As the user interacts with the 3D model, the server renders each frame and sends an image back to the client's web browser for display. This minimizes hardware requirements for the client. The images are sent in real time, so it appears as if the rendering is being done in real time on the local machine.

To select the Server-Side Rendering Type, in the **HOOPS_Viewer_Renderer_Type** variable, change the **Value** field to **Server**.

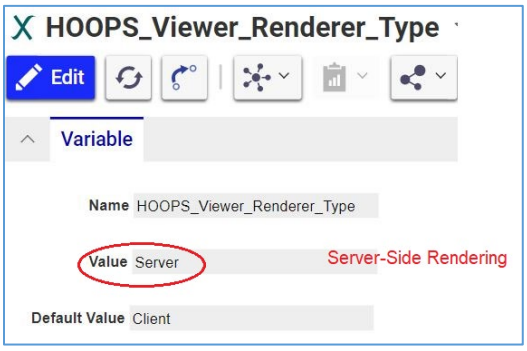


Figure 13.

Note: If the server where Streaming Viewer is installed has Graphic Processing Unit (GPU), then the **windowsServiceRespawnEnabled** parameter in the **Config.js** file should be set to "**False**".

3.2.4 Streaming Viewer vs Dynamic Viewer

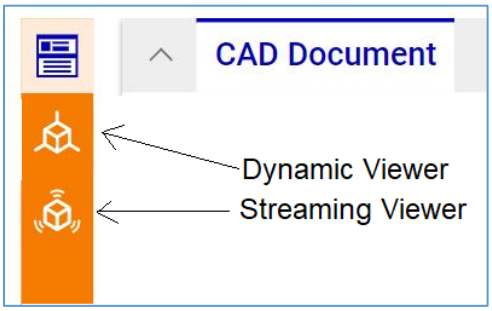


Figure 14.

The Dynamic and Streaming Viewers have the similar functionalities. The difference between them is as follows:

- Dynamic Viewer supports SCS viewable files whereas the Streaming Viewer supports SCZ viewable files. Each is designed for a specific purpose.
- For Streaming Viewer, a separate window service called Hoops Server is used. The HOOPS Server initiates individual Stream Services to support each Streaming Viewer instance.
- Larger datasets can be streamed from a server to a client in a single request with the help of the Streaming Viewer unlike Dynamic Viewer which uses individual client/server HTTP requests.

Warning Dynamic Visualization and Streaming Visualization each use different view files, and each are not compatible with the different Viewers.
Therefore, if the Streaming 3D Viewer is installed in an environment where the Monolithic or Dynamic 3D Viewers have been previously installed and used, all existing native files of existing CAD Documents need to be re-converted. There is no automated means to perform this reconversion!

Warning The Streaming Viewer currently cannot be deployed in a cloud environment.

The HOOPS Server must be deployed with networked file access to a single vault containing view files for rendering.

Note: Only one Streaming Viewer can be installed on one machine at a time.

3.3 Tree Grid View vs. Model Browser

The concept and vision of the Dynamic Product Navigation (DPN) require an alternate mechanism for displaying business objects (Items) associated with 3D components displayed in the 3D View:

- The *dynamic* part of DPN specifies that some conditional and customizable logic should determine 3D view content. A QD provides this ability.
- The *navigation* part of DPN specifies that the 3D View can be used to access *related* business objects (Items). A TGV Definition provides this ability.

The optimal choice for the Dynamic /Streaming Viewer is the TGV—a user interface component that displays Items and associated properties in a grid.

This section compares the features of the TGV Model Browser of the Dynamic Viewer as well as the Streaming Viewer against the simple Model Browser of the Monolithic Viewer to make it clear how each functions and what information each provides.

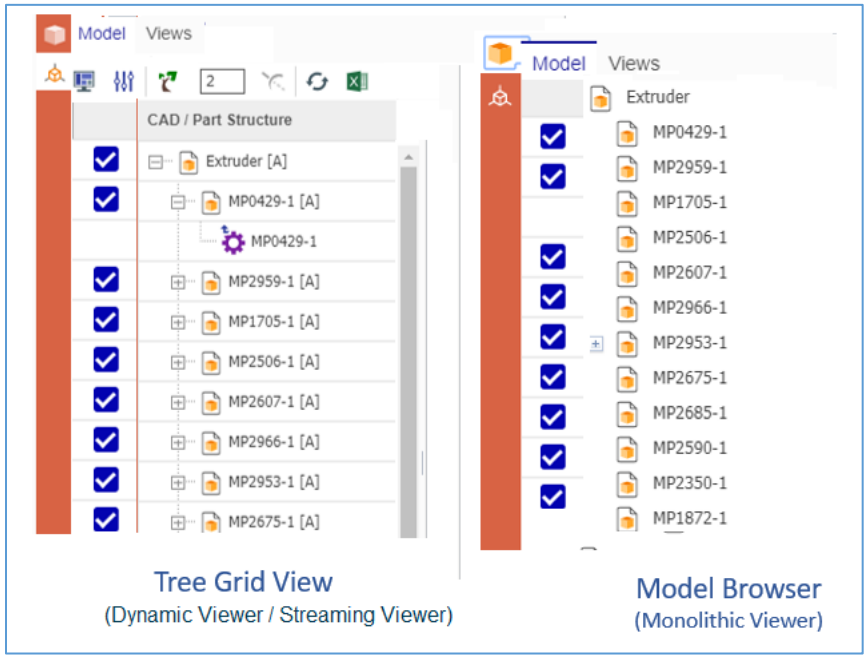


Figure 15.

3.3.1.1 QD vs. AML

A TGV is populated with a response generated from the execution of an associated QD. In addition to the **CAD** Items that form the Query basis, administrators can add other related ItemTypes whose Properties should be displayed in the TGV UI. Using TGV Definitions, administrators can 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 left-most Tree column with nodes for the following Items that display the following properties for a given Item:

- **CAD**: a **CAD** Item **Name** and revision in square brackets.
- **Part**: a **Part** Item **Name**.

In addition, two columns were added for the `is_released` and `is_current` Boolean properties that both are displayed as checkboxes.

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

3.3.1.2 Partial vs. Full CAD Structure display

By default, a TGV is populated using only a portion of QD response results. When related content has multiple levels, a user determines which content to query based on selected nodes in the Tree column. Expanding a node re-executes the query from this node and adds new rows to the TGV. On the contrary, the simple Model Browser uses an AML to execute an exhaustive (fully recursive) search for the entire CAD BOM structure. The Tree is populated with the full list of CAD Items from the query.

Each approach has a trade-off. For the TGV, query execution is generally faster, much faster for large results. However, after the simple Model Browser is populated, all data exists on the client and can be readily viewed. This affects data synchronization between the simple or TGV Model Browser and 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 TGV with all levels of the hierarchy up to the selected part component; see Figure 39. Once the data is populated, the query should not be re-executed, but the initial population can take several seconds.

In general, and especially for large and deep assembly hierarchies, the lazy-loading approach used by the TGV 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 TGV Definition—administrators have control over the granularity of query response data.

3.3.2 View Function Comparison

The Monolithic, Dynamic and Streaming 3D Viewers use the same technology— HOOPS Communicator. The main difference is how the 3d View is populated. However, there are also limitations associated with the available view functions that this section explains.

3.3.2.1 Product Manufacturing Information (PMI)

The product management information (PMI) is typically stored at the assembly level within CAD native files. When dynamically generating assemblies in the Dynamic Viewer or Streaming Viewer, only view files of parts are included. Therefore, only PMI stored with a native file and included with the converted data in a view file[4] can be viewed in the 3D View. PMI is not guaranteed to be positioned such that it does not interfere with other rendered 3D component geometry.

3.3.2.2 Configurations

The Dynamic Viewer is only accessible for CAD assemblies. Like PMI, configuration data stored in an assembly view file is not accessible and likely not applicable when rendering an assembly based on some arbitrary query. Therefore, the Configurations interface is not included in the Dynamic Viewer.

The Streaming Viewer is accessible for both CAD assemblies and CAD Components.

3.4 Visual Collaboration for Aras 3DV

The Aras 3DV is integrated with Visual Collaboration to help visualize complex product information using a simple UI. End users can freeze a 3D View scene, add 2D graphic and text markups, and save a resulting image with a comment in a discussion thread, which may be related to 3D design, for example.

4 PMI is included by default in the CAD conversion process.

There is an ability to add a 3D Markup to a snapshot on an SSVC comment. And then, they can reconstruct the original 3D View scene from the message.

The 3D View reconstruction requires information about the View state: camera zoom and rotation, hidden and shown 3D components, and so on. This information is contained with the graphic in a message. For DPN, restoring a 3D View scene necessitates the re-execution of the query used to define the View, parameters used, View mode, and camera location.

3.5 View Modes

View Modes are a mechanism to visualize 3D geometry data using alternate colors and opacities. View Modes are enabled exclusively using the Query Processor API with the addition of Rendering Configurations; see sections 7 and 7.2.7 respectively.

3.6 Saved Views

Saved Views are a mechanism to restore an original 3D View scene from a modified 3D View scene by re-applying the functions that resulted in the original scene at the time it was saved. These include the selected Dynamic View Definition, used Parameter Values, selected View Mode, added parts and assemblies (see section 3.6), and current camera position.

A Saved View includes only *input* information as opposed to the IDs, for example, of all displayed view files. As a result, a restored 3D View scene is not guaranteed to match an original 3D View scene because the results returned by QD execution may include a different result set.

Not all 3D View scene parameters are restored. For example, the Display Style, Exploded Increment, Measurements, and Cutting Planes are not included.

Given Saved Views will remain associated with all versions of a given target Item.

Saved Views are stored using the **SavedView** ItemType with source types derived from the Dynamic View Definition context ItemTypes collected under the **SVTargetItem** poly ItemType. On Dynamic View Definition creation, a context ItemType will be added to the **Poly Sources** list of the **SVTargetItem** ItemType, if it has not been already added. The **SVTargetItem** ItemType will be created during the **DynamicModelConstrator** AML package import.

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

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

The restoration of the camera position for the **Saved Views** can be turned on and off by the administrators using the `3D Visualization.RestoreCamera` variable. To restore the current camera position of the Saved View in the 3D Viewer, set the variable to **True**. When the variable is set to **False**, the camera position of the Saved View in the 3D Viewer is not restored when the Saved View is rendered.

In future, end users will have the ability to configure sub section names under the Saved View section. This will enable users to organize the Saved Views.

3.6.1 Modifying Default Saved Views Label

The following steps outline the process to change name of the **Saved View**s label under the **View** tab of the 3D CAD model by root user:

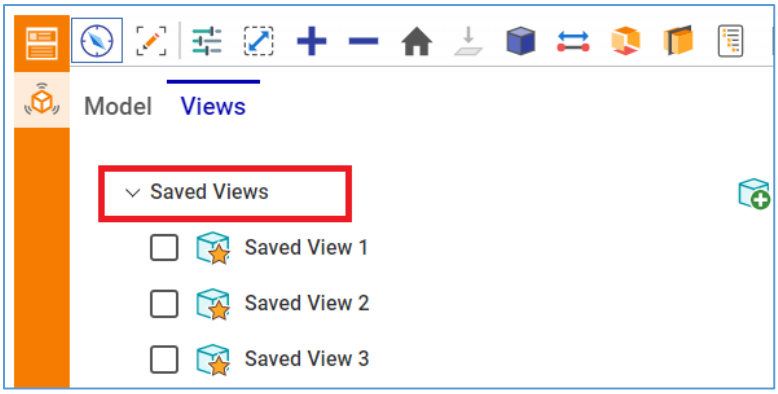


Figure 16.

1. From the **Table of Contents,** expand **Administration** and select **ItemTypes.**
2. In the Quick Search, search for **SavedViewSection** ItemType.

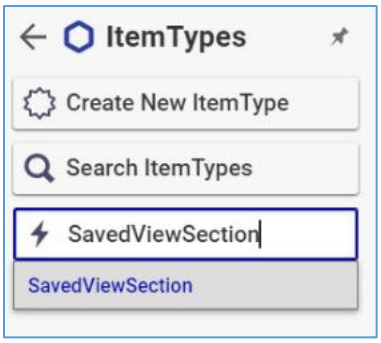


Figure 17.

3. In the **SavedViewSection** ItemType, click **More** and select **Add to TOC**.

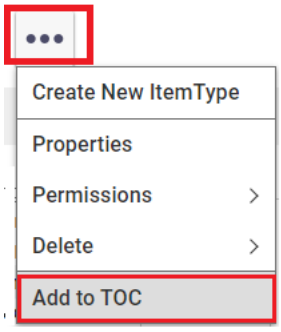


Figure 18.

The **SavedViewSection** ItemType should appear in the TOC section.

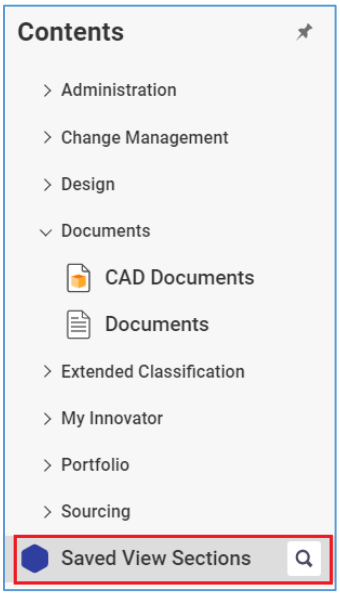


Figure 19.

4. From the **Table of Contents**, click **Saved View Sections** ItemType and Click **Search Saved View Section**. The **Saved View Section** grid view appears.

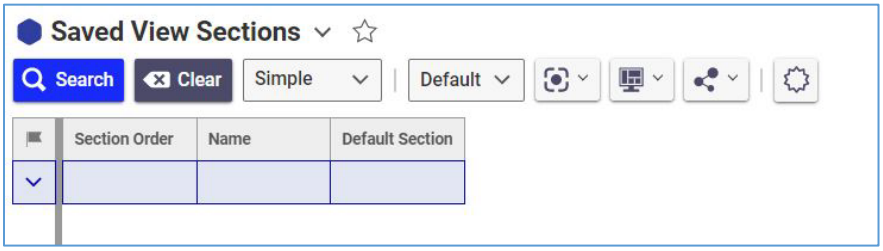


Figure 20.

5. Click **Search**.
6. Select **Saved Views** instance from the grid and click **Edit**.

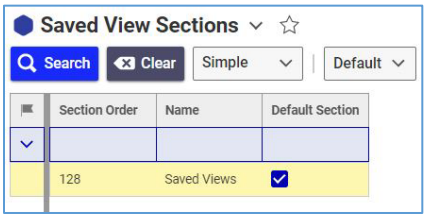


Figure 21.

7. Change the name of the **Saved View** from the **Name** field.

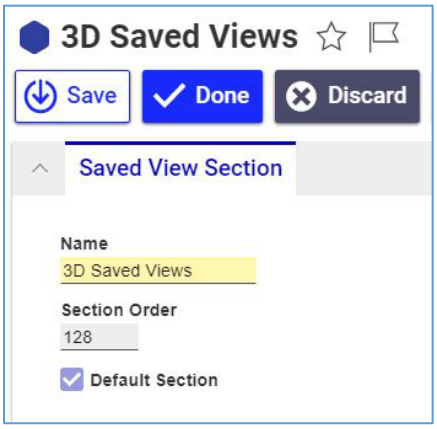


Figure 22.

8. Click **Done** to save the changes.

   To confirm the changes, go to the **View** tab of the CAD model.

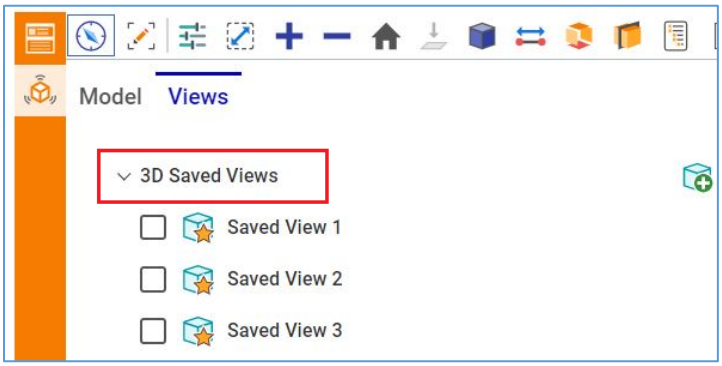


Figure 23.

3.7 Digital Mockup

In Aras 3DV, a digital mockup is an ad-hoc arrangement of 3D component geometry on the 3D scene for analysis, review, or other purposes. End users can visualize collections of 3D assemblies, subassemblies, and parts in a manner that may be different from how these objects were defined within a CAD editor.

The Dynamic/Streaming Viewers provides the users with the abilities to:

- Add additional assemblies, subassemblies, and parts onto a single 3D scene
- Manipulate the position, orientation (by 3D rotation), and display of a selected 3D geometry.

Whenever an additional assembly, subassembly, or part is added to the Dynamic/Streaming Viewers:

- An associated Dynamic View Definition is re-executed using selected Parameters.
- The Tree Grid View and 3D View scene are refreshed.
- All defined View Modes and Parameters are applied to the context and added assemblies, subassemblies, and parts.

- No transformation is applied to the root of the added additional assembly, subassembly, or part.

An additional assembly, subassembly, or part is added to the TGV as a separate root item. The context item may not be at the top in the TGV because this grid is sorted using the sorting logic of an associated QD.

3.8 Context Menu Functions

The Dynamic Viewer and Streaming Viewer supports context menu functions in the TGV and 3D View that can be extended as outlined in sections 7 and 9.2.

3.9 Assembly Level Geometry Support for NX Files

The Dynamic and Streaming Viewers support the ability to view **Assembly Level Geometry** for **NX** files. The **NX Assembly** view files directly contain 3D geometry at the root level (assembly level), rather than parts that are normally used to define the assembly.

The native CAD assembly files can contain 3D component geometry. To create Assembly Level Geometry **FileRepresentation** view files for **NX** files, the **ConversionServerConfig** file in the root Aras Innovator folder is updated with the command-line option. The **FileRepresentations** view file is generated during the conversion process, and the default Query Definition supports viewing this structure in the Dynamic Viewer or Streaming Viewer.

The following figure describes the process.

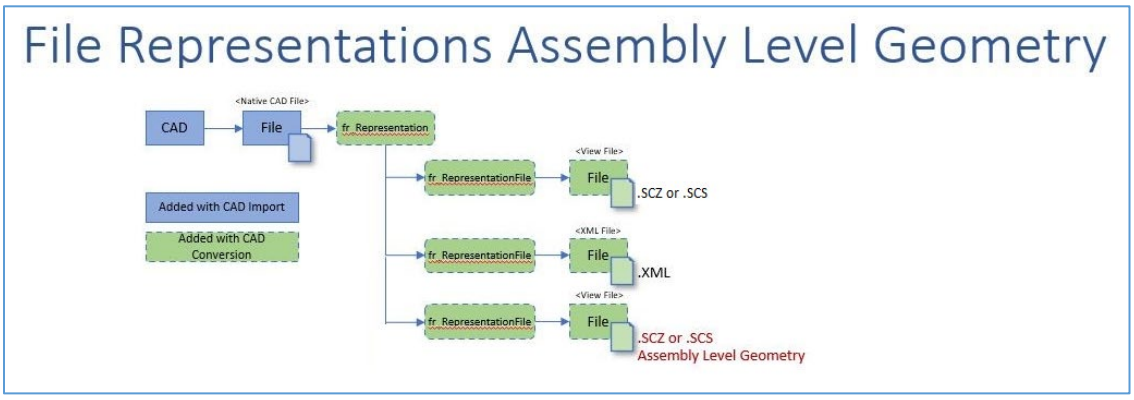


Figure 24.

The default Tree Grid View Definition for Dynamic and Streaming Viewers is modified with a row for **Assembly Level Geometry** view file. The new row contains a text called **body**.

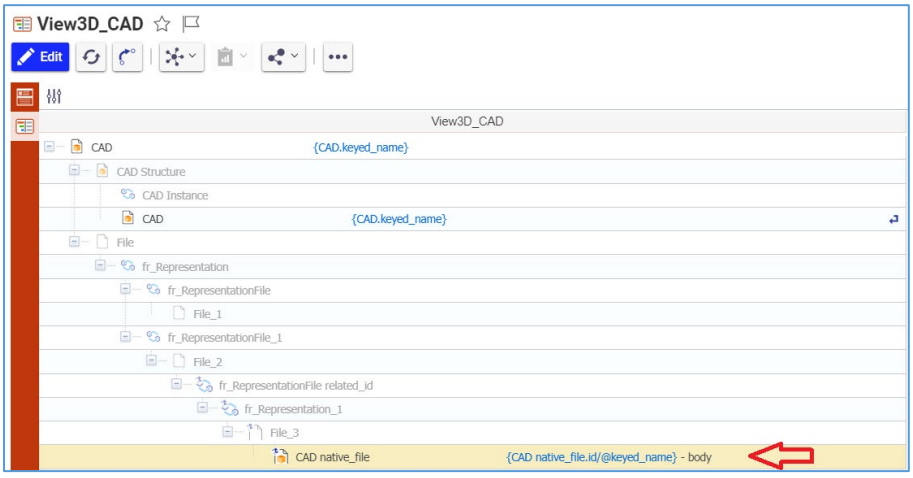


Figure 25.

The following steps outline the configuration to view the **Assembly Level Geometry** for **NX** files:

1. Install Aras 3D Visualization package. Refer to the *Aras 3D Visualization 26 – Installation Guide.*
2. In the root Aras Innovator folder, change the **Assembly Command** tag in the **ConversionServerConfig.xml** file to the following:
   - For **Dynamic 3D Viewer:**

     ```
     <AssemblyCommand dynamicEnabled="True" arguments="--
     sc_compute_bounding_boxes 'All' --input_pdf_template_file
     'C:\Aras\14sp10\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%'" isolatedModelArguments="--
     output_scs '%filepath%\%filename%.scs' --import_hidden_object 'True' --
     import_pmi false --output_logfile '%filepath%\%filename%_isolate.log' "
     />
     ```

   - For **Streaming 3D Viewer**:

     ```
     <AssemblyCommand arguments="--sc_compute_bounding_boxes 'All' --
     input_pdf_template_file 'C:\Aras\14sp10\Templates\Blank_Template_L.pdf'
     --output_pdf '%filepath%\%filename%.pdf' --output_png
     '%filepath%\%filename%.png' --output_png_resolution '150x150' --
     output_xml_assemblytree '%filepath%\%filename%.xml' --output_prc
     '%filepath%\%filename%.prc' --background_color '1.0, 1.0, 1.0' --
     output_logfile '%filepath%\%filename%'" isolatedModelArguments="--
     sc_create_scz 'true' --output_directory '%filepath%' --
     import_hidden_object 'True' --output_sc '%filepath%\%filename%' --
     import_pmi false --output_logfile '%filepath%\%filename%_isolate.log' "
     streamingEnabled="True" />
     ```

     The **Assembly Level Geometry FileRepresentation** view files are created. After the above configuration step, if a user converts **NX** assemblies which contains a 3D geometry at the assembly level, an SCS or SCZ file is generated which represents the newly added geometry.
     In the Default Dynamic View Definition, this file is represented as an assembly with associated child components and an assembly level geometry associated with a text called **Body.**

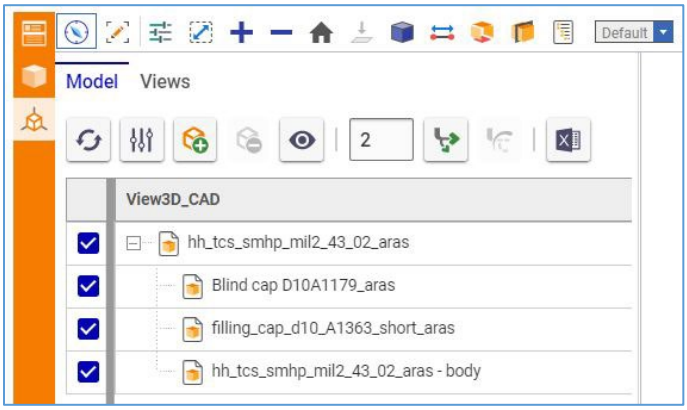


Figure 26.

4 Dynamic Enabling

There is an Action 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, 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, the **Dynamic Enabled** flag is set on each CAD assembly. The combination of this information will enable the Dynamic Viewer. The **Dynamically Enable Legacy CAD Items** Action is added so that any **CAD Structure** Item that was generated in Aras Innovator versions from 11.0 SP11 and higher and 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 to use the Dynamic Viewer.

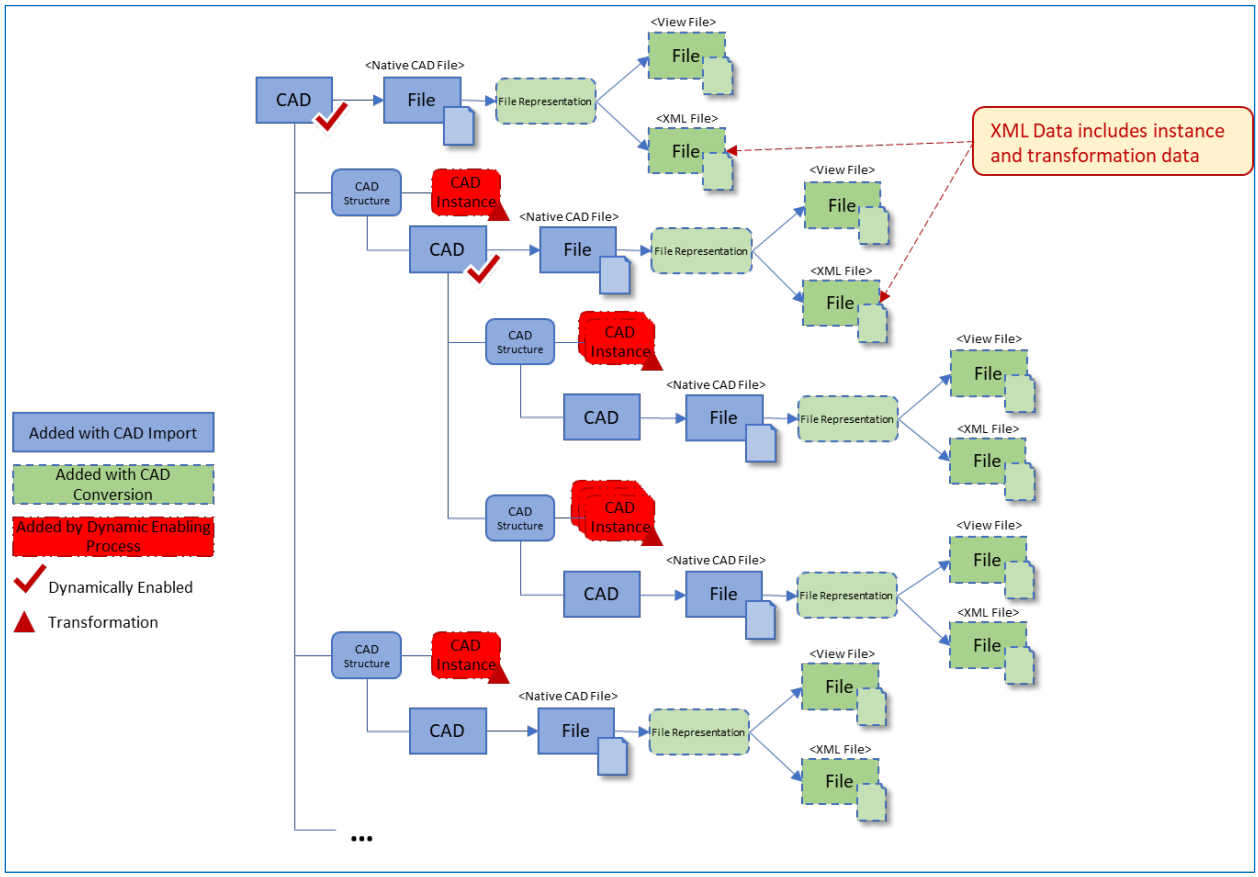


Figure 27.

4.1 Dynamic Enable Process

The Dynamic Enable process uses the XML data created from the Conversion Process (see Section 3.1.1 and figure 20) 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 **View3D_DynamicallyEnablingCAD** Query Definition 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.

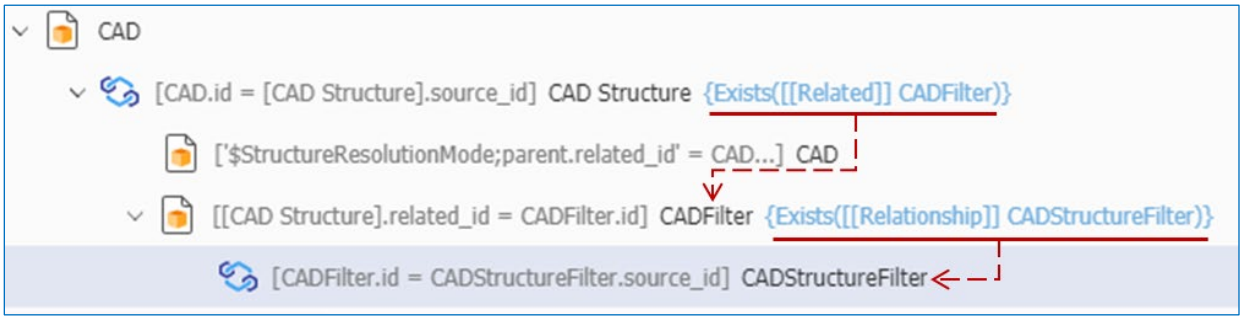


Figure 28.

Warning Federated Properties cannot be used in Where clauses for a Query Definition.

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.

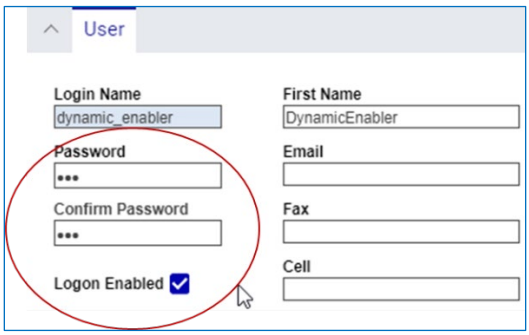


Figure 29.

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 and Streaming Viewers to determine what should be included in a 3D View when the View is processed (see Section 3.1.4). There are restrictions on the Query Items included and the specific relationships for the Dynamic Viewer and Streaming Viewers 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

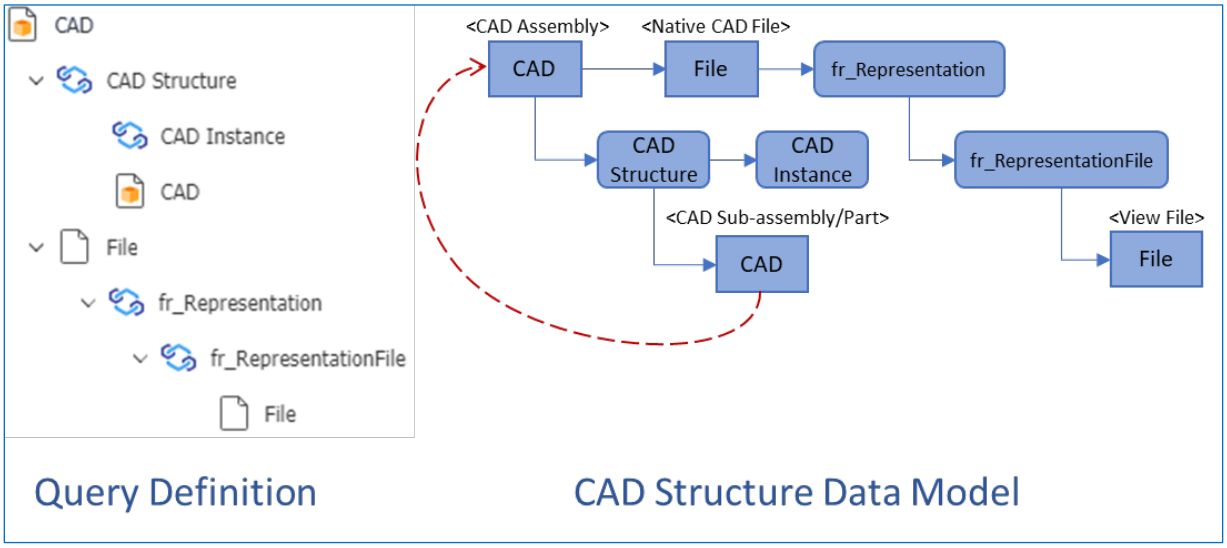


Figure 30.

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 3.1.1) is also stored as a File Representation. The Transformation is stored as a String but parsed as a 4X4 matrix of floating-point values.[5] 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.

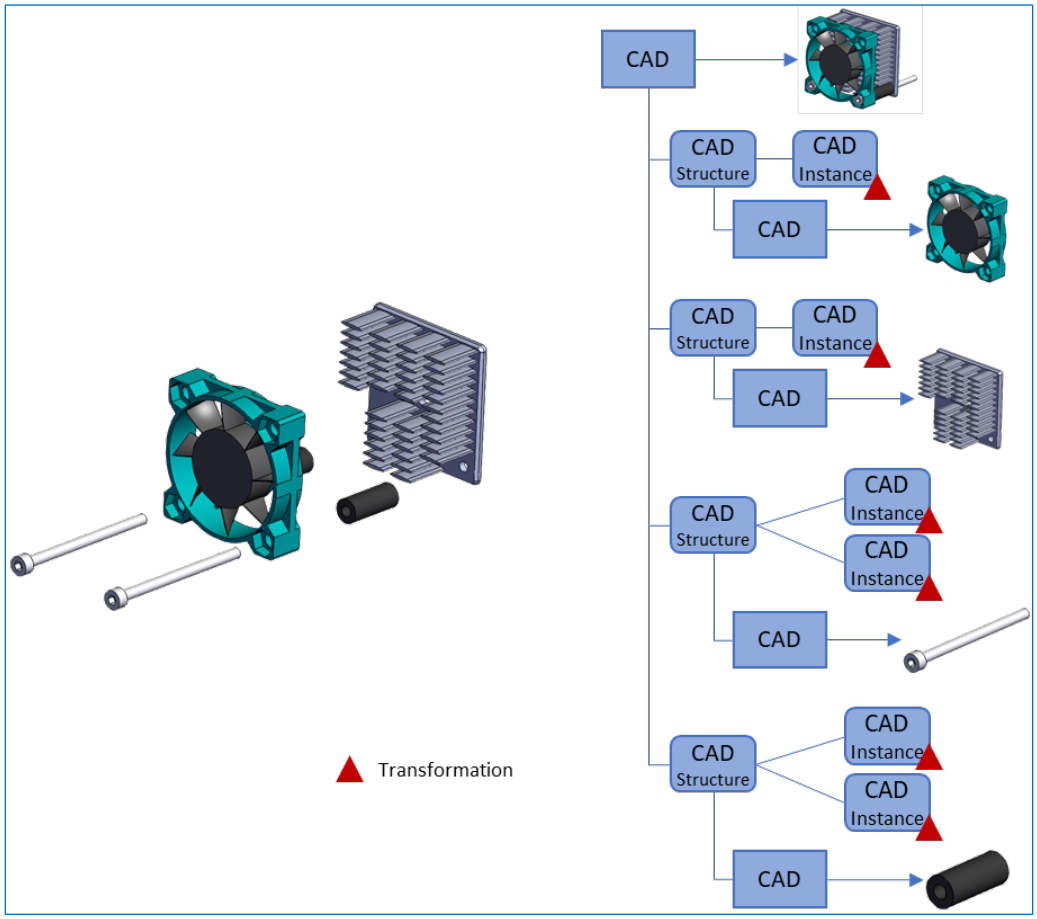


Figure 31.

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.

5 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 24 shows the Query Items for the Base Query Definition. These elements correspond to the diagram. 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 and Streaming Viewers (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 and Streaming Viewers.

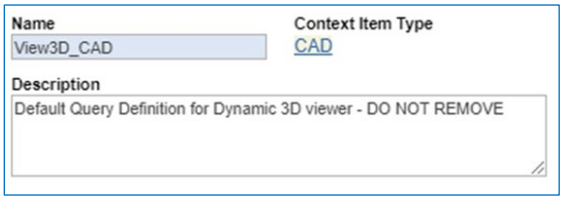


Figure 32.

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 and Streaming 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 properties given in the following table are required to be included in a Query Definition for the Dynamic Viewer and Streaming Viewer.

ItemType	Alias	Property
CAD	CAD	`id, keyed_name, x_min, x_max, y_min, y_max, z_min, z_max, dynamic_enabled, config_id, is_current`
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 or Streaming Viewer (see Section 3.1.1 and 3.2.1). 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 3.1.4).

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.[6]

5.1.3.1 Adding Properties

Any Property can be added to the base Query Definition if the Core Properties (see Section 5.1.2 Base Query Definition) 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 or Streaming Viewer.

Note: You must add Properties to the Query Definition 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 and Streaming 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 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.

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

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 16 and Figure 17. 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.

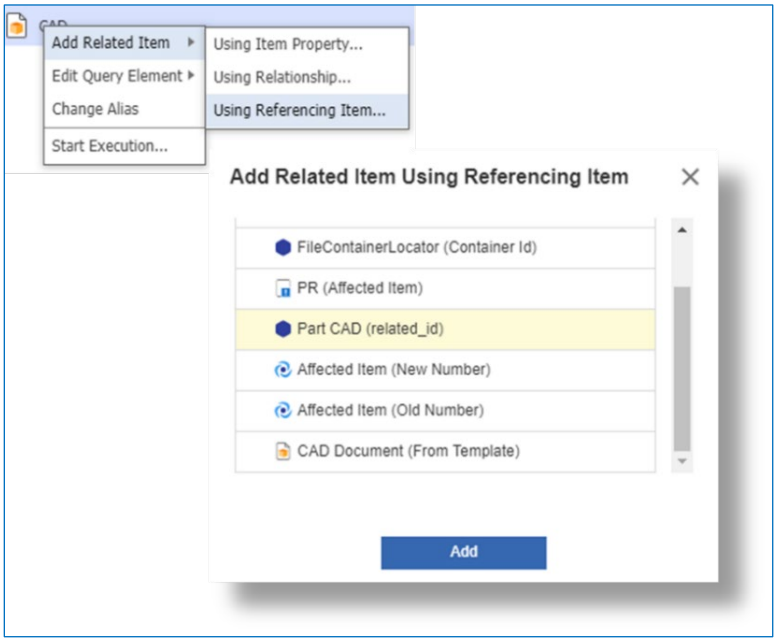


Figure 33.

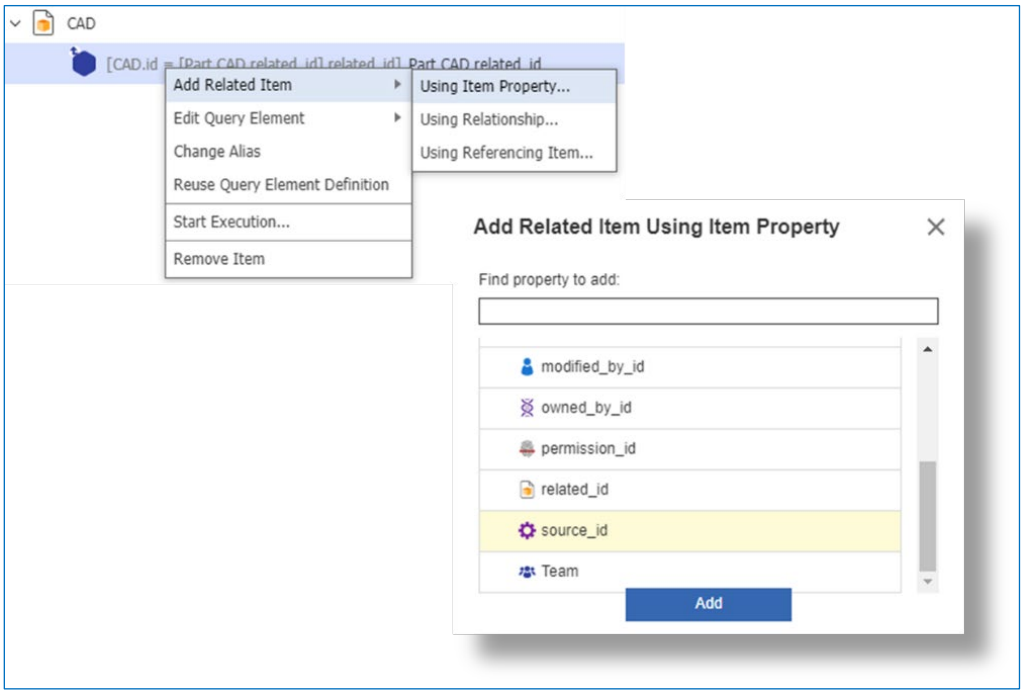


Figure 34.

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.

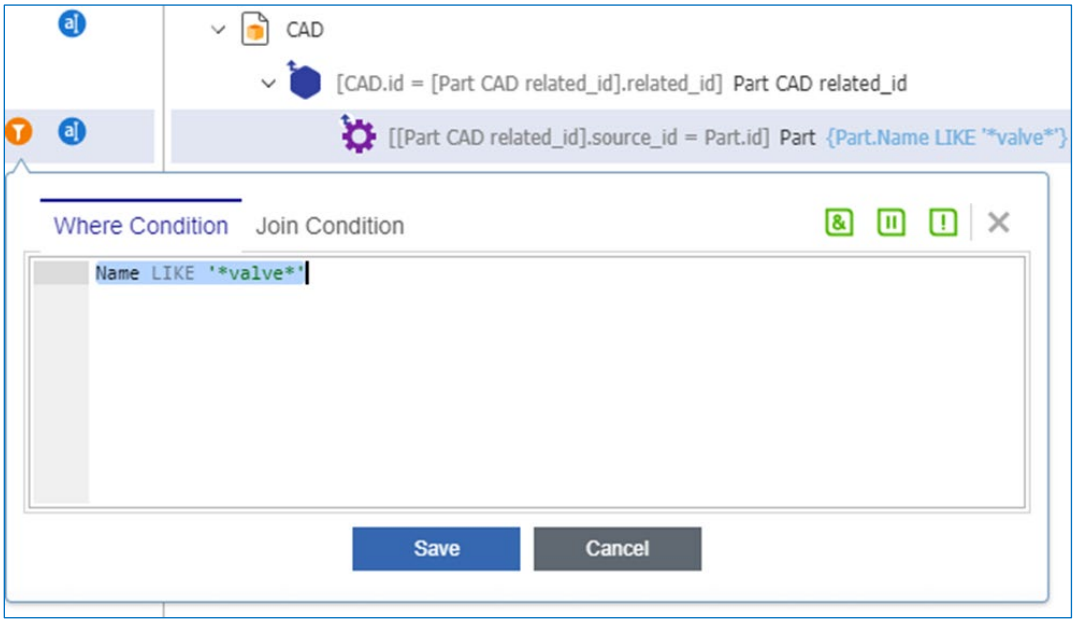


Figure 35.

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.

Warning Federated Properties cannot be used in a Where Condition.

Conditional logic applied to any ItemType that is *added* to the Base Query Items can be done without affecting the Dynamic Viewer or Streaming 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 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:

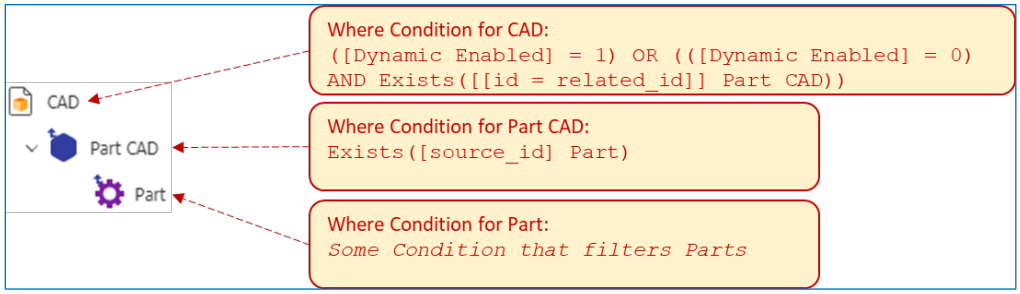


Figure 36.

   - 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.

Warning Federated Properties cannot be used in Where Conditions.

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 18 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 30 shows the added Parameter '`partNameFilter`' which can then be used in place of the value 'valve' in Figure 28. 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.6 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.

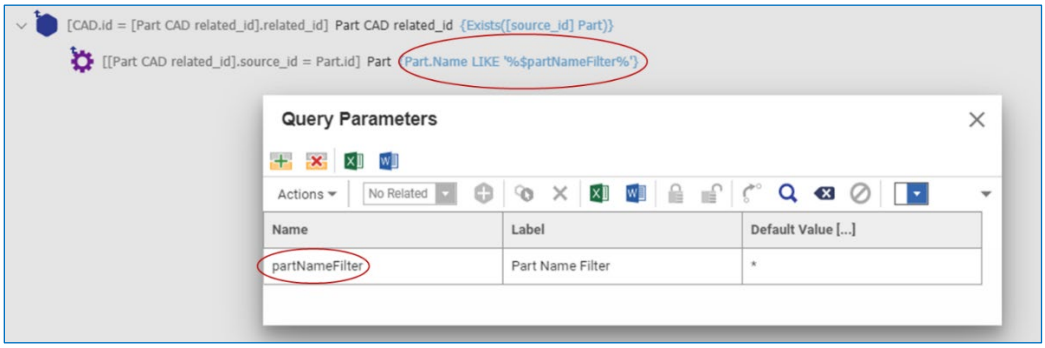


Figure 37.

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.

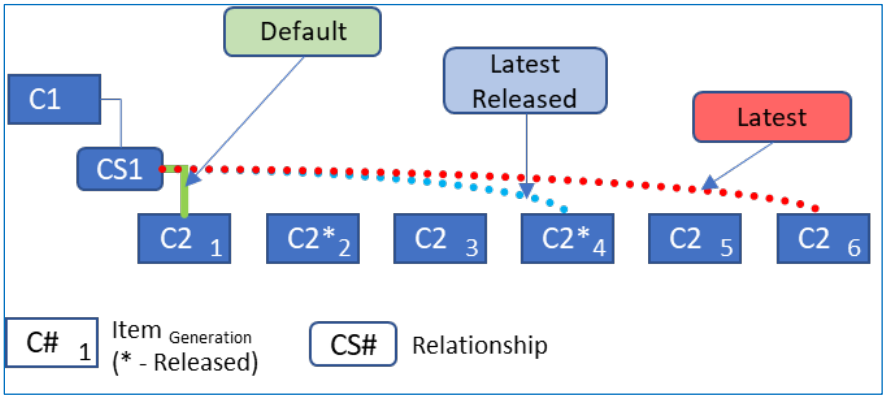


Figure 38.

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.

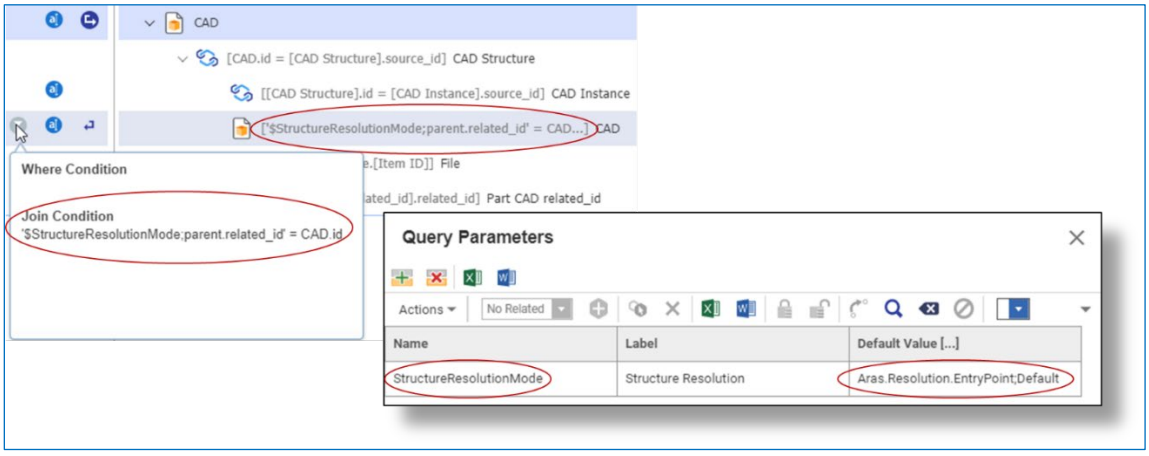


Figure 39.

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 22. 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 23.

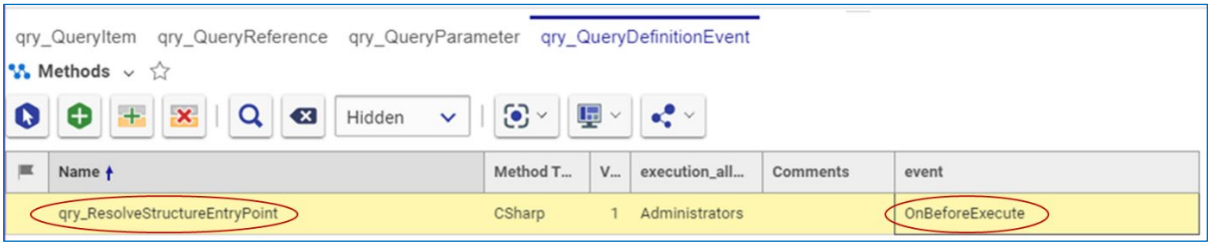


Figure 40.

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

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`.

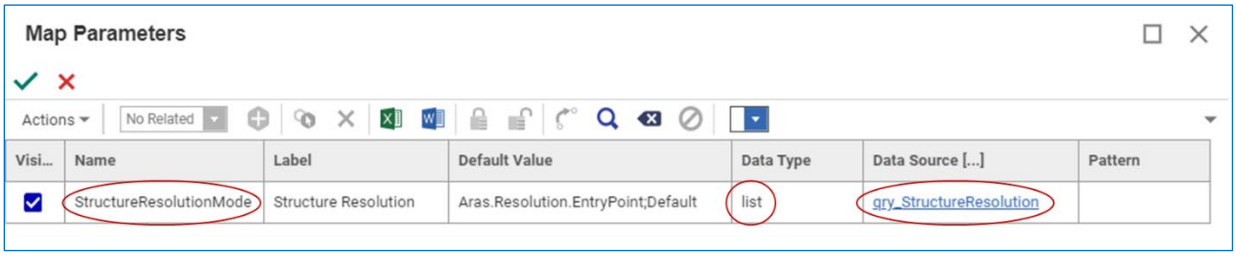


Figure 41.

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.

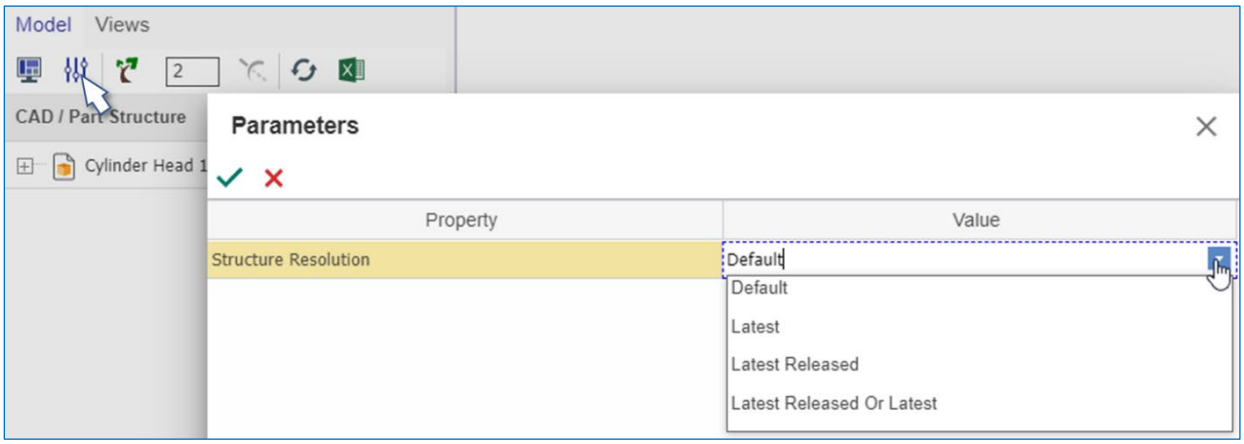


Figure 42.

5.1.3.7 Applying a Where Condition Against Resolved Items

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:

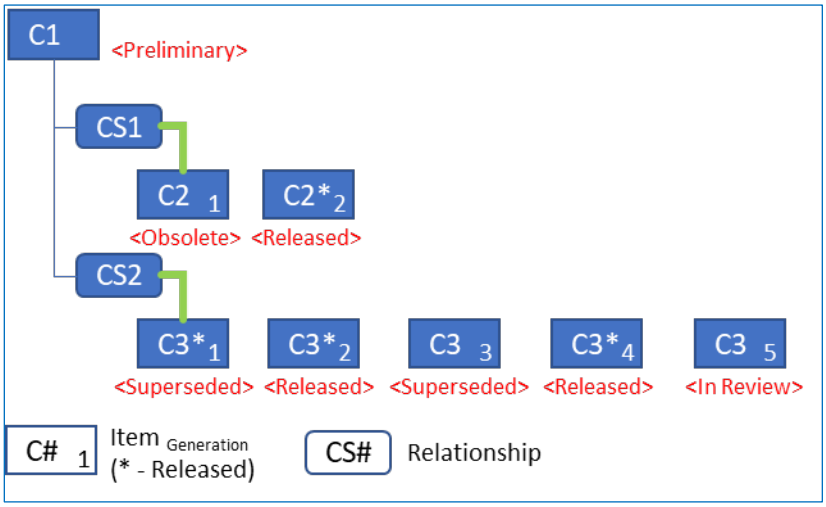


Figure 43.

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:

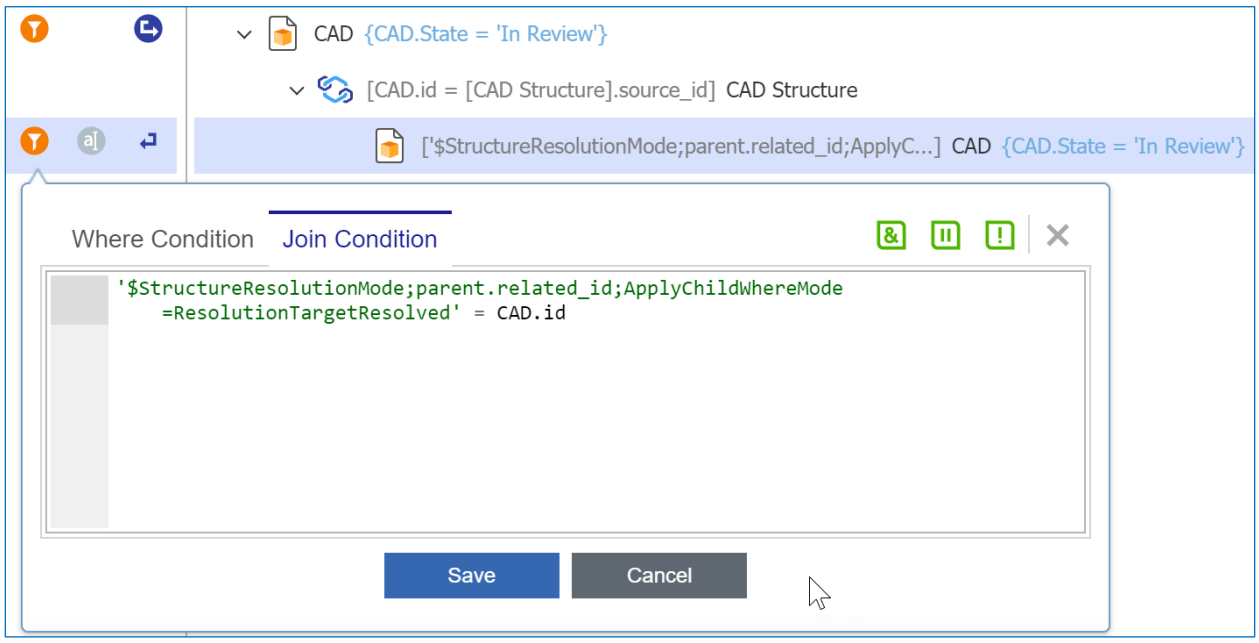


Figure 44.

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.

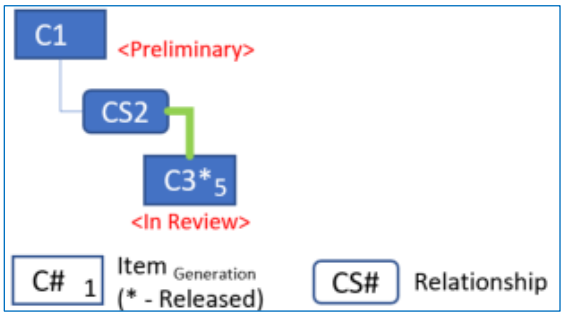


Figure 45.

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 and Streaming Viewers, 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 3.1.2). 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 and Streaming Viewers. This Tree Grid View Definition can be used as a guide when creating alternative Views for use with the Dynamic and Streaming Viewers.

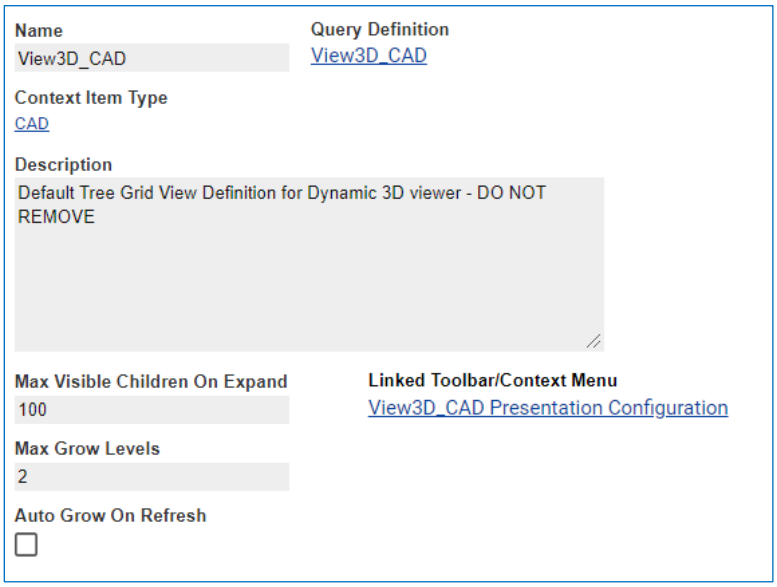


Figure 46.

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 and Streaming Viewers.

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, the **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 30 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.

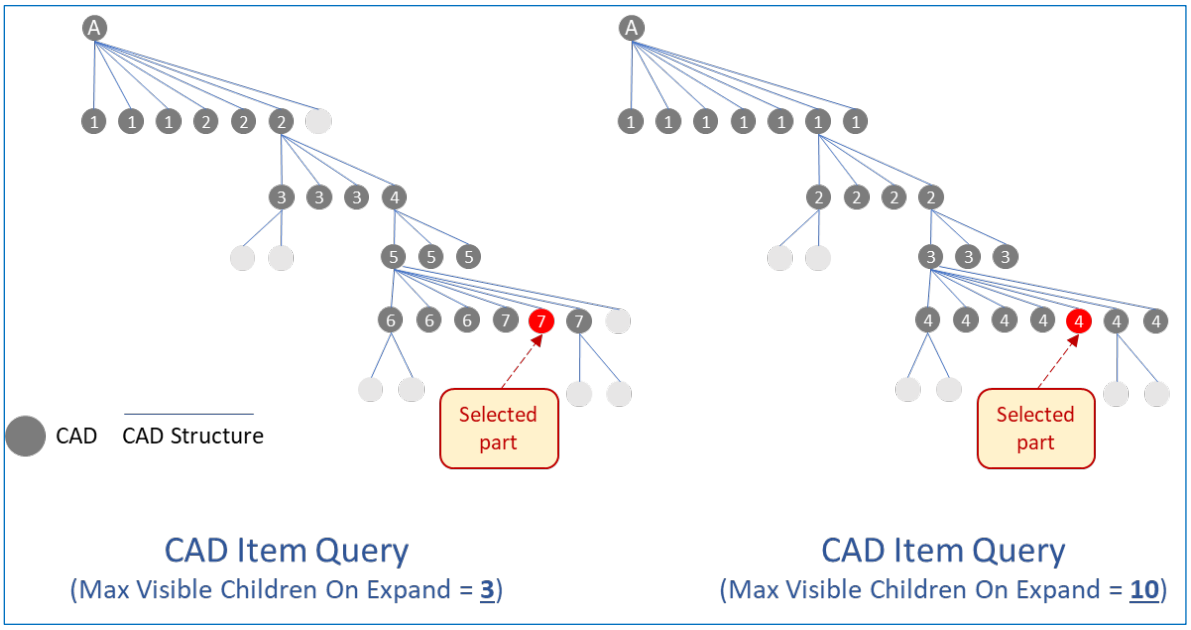


Figure 47.

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.

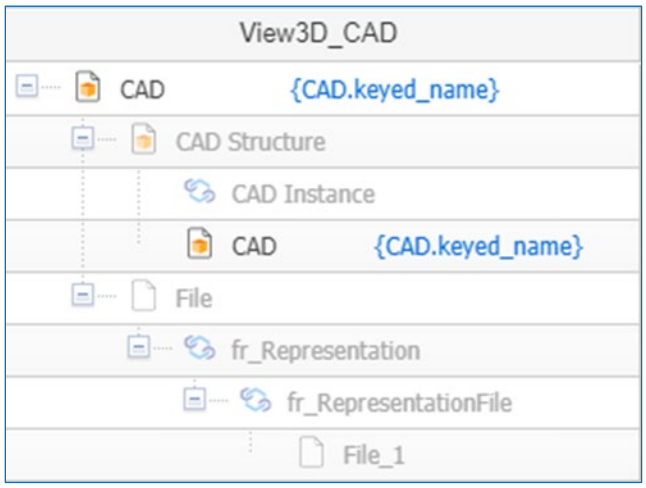


Figure 48.

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 and Streaming Viewers:

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 and Streaming Viewers.

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.

6 Assign a Linked Toolbar/Context Menu. This is required for all Tree Grid Views associated with Dynamic View Definitions, created in the following step. If a Tree Grid View does not contain a Linked Toolbar/Context Menu, the user will receive an error. The default Linked Toolbar/Context Menu, **View3D_CAD** Presentation Configuration, should be reused to maintain access to default Toolbar buttons and Tree Grid View Context Menu, described in section 3.7.
Create a Dynamic View Definition. The Dynamic View Definitions should be assigned to the specific viewer (Dynamic or Streaming Viewer)

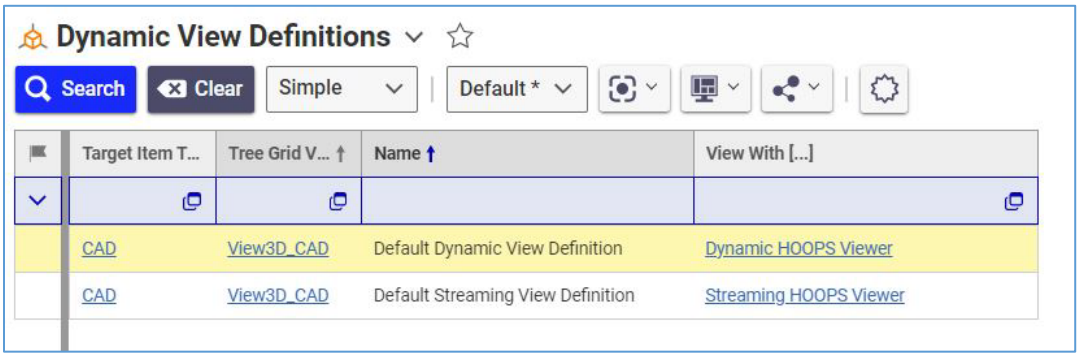


Figure 49.

7 Click on **New Dynamic View Definition**.

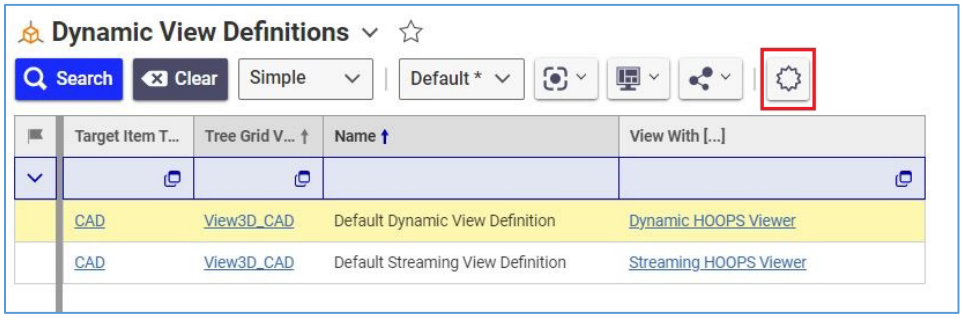


Figure 50.

8 Enter the **Name** or the Dynamic View Definition Item which is used in the context menu for the Dynamic Viewer /Streaming as shown below.
9 Select the **Tree Grid View Definition**.
10 Select the viewer (Dynamic or Streaming) in **View With** field.

11 Assign a **Data Processor** method. "**dpn_GraphAPIQueryProcessorMethod**" is available by default to process CAD.

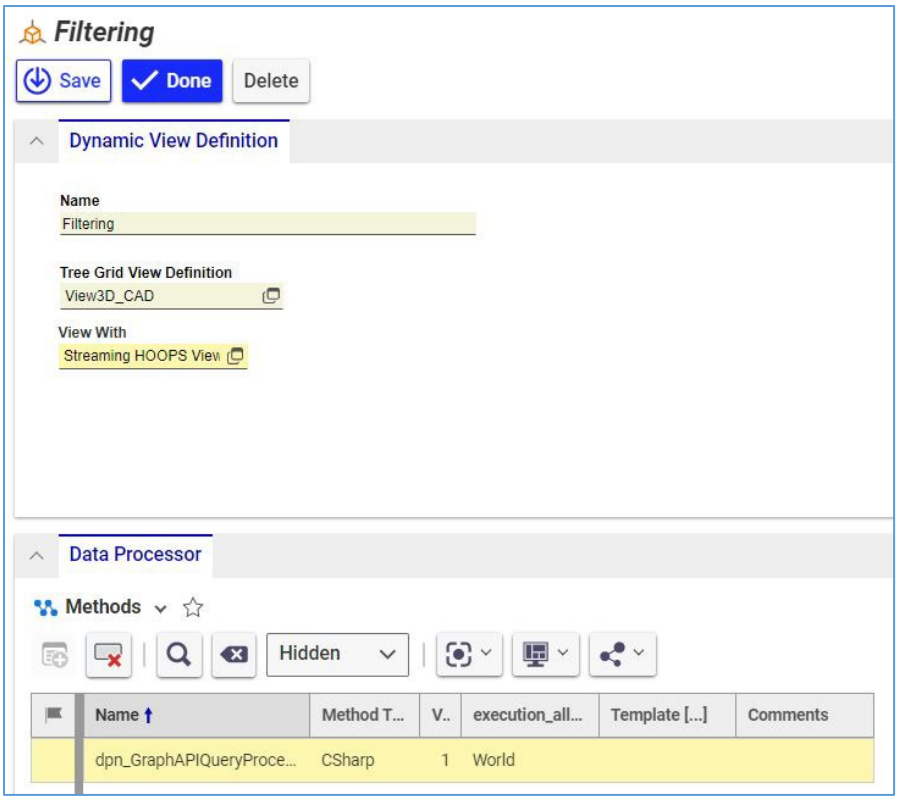


Figure 51.

The Dynamic Definition for Dynamic Viewer appears as follows:

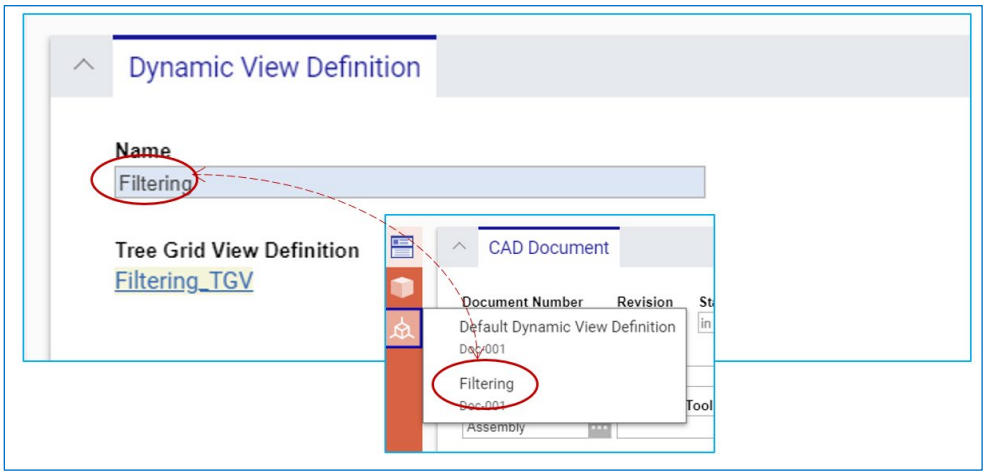


Figure 52.

The Dynamic View Definition for Streaming Viewer appears as follows:

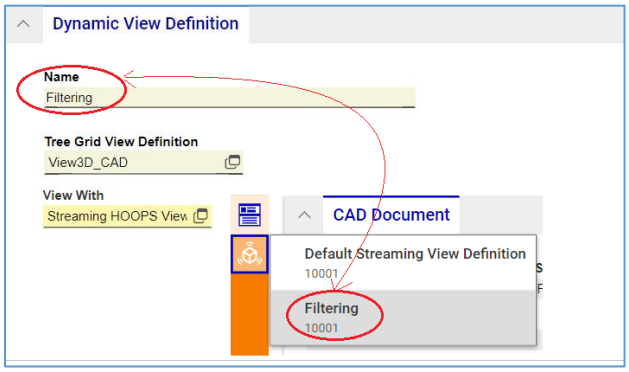


Figure 53.

Note: The CAD Query Item must be mapped for the Tree Grid View to function properly in the Dynamic Viewer and Streaming 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 and 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. The default Dynamic View Definition does not support the 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 34 and Figure 35). To configure the Tree Grid View to allow for selecting single instances, refer to section 5.2.7.

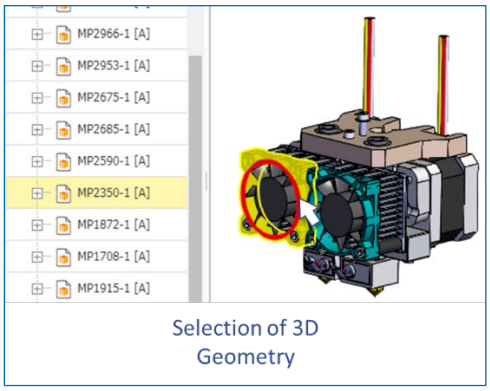


Figure 54.

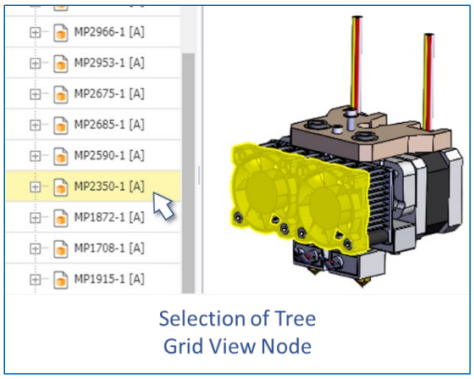


Figure 55.

5.2.4 Selecting a Dynamic View Definition

Note: The default DVD will be displayed as an option in the Dynamic Viewer/Streaming 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.

The Dynamic Definition for **Dynamic Viewer** appears as follows:

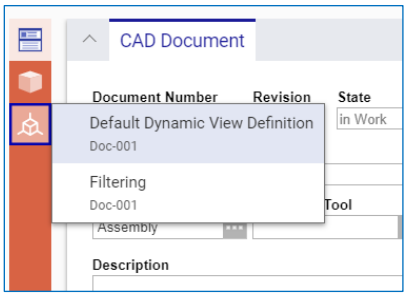


Figure 56.

The Dynamic View Definition for Streaming Viewer appears as follows:

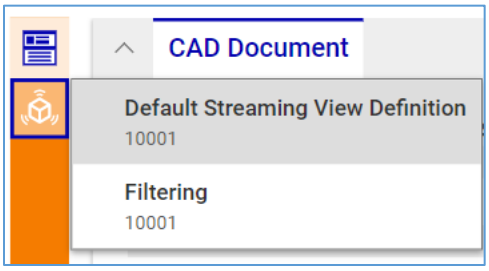


Figure 57.

If there is a Dynamic View Definition that is currently in use, it will be highlighted.

5.2.5 Auto-Execution

By default, users must 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 37).

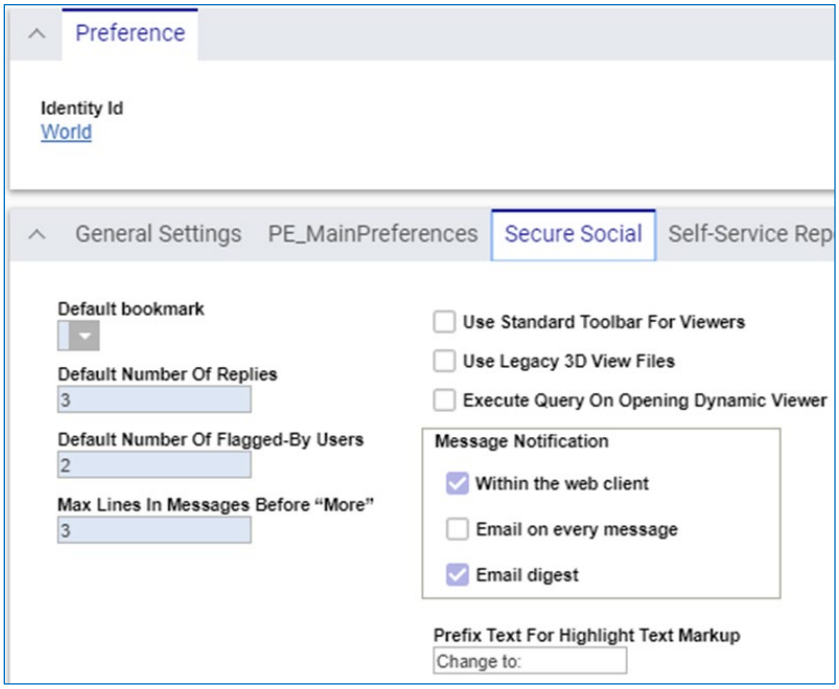


Figure 58.

5.2.6 Enabling the Open Context Menu Item

Tree nodes (rows) in the Tree Grid View have a default context menu with the **Open** menu item for opening the associated Item. This menu item is not enabled by default for the Tree Grid View Definition because the associated rows in the Tree Grid View Definition could be combined and thus reference more than one Item; see the *Aras Innovator – Tree Grid View Administrator Guide* for more information.

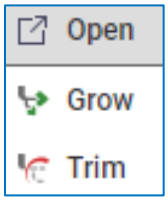


Figure 59.

To enable the **Open** menu item 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 39). The software that executes the **Open** function requires the

ItemType and the ID of the specific Item 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

a. `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.

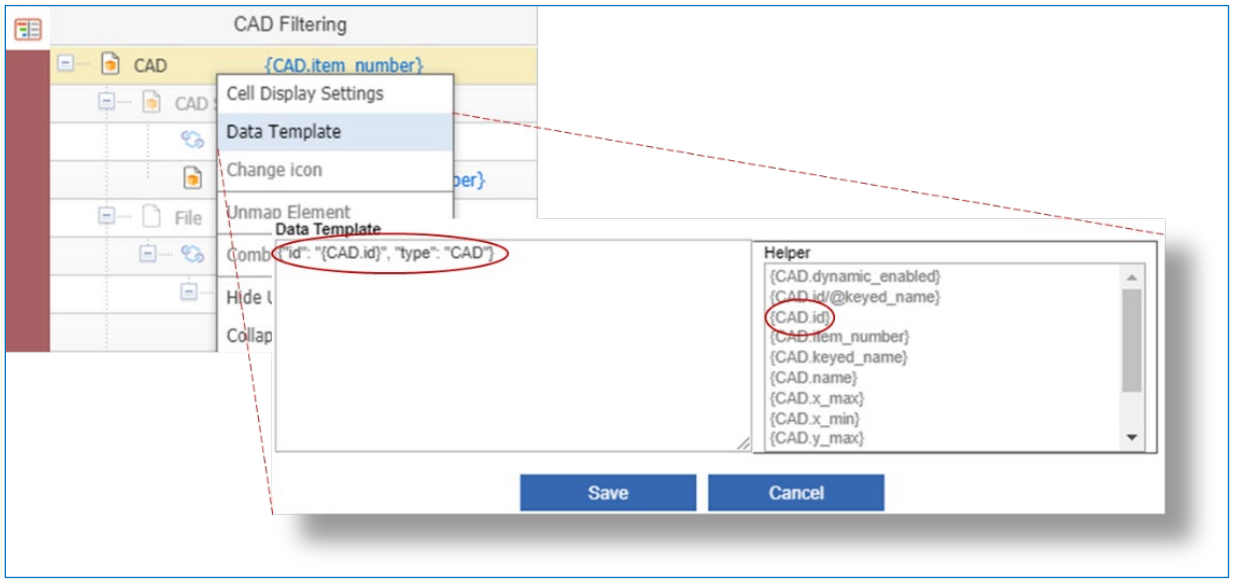


Figure 60.

5.2.7 Displaying Instances with Synchronization

It is possible to customize the Tree Grid View to support the display of instances by using the Tree Grid View ability to combine rows. The ability to synchronize the combined rows between the Tree Grid View and the Dynamic/Streaming Viewer relies on the reference path of the instance.

The following steps outline the process of configuring CAD for Instance Synchronization between the Tree Grid View and Dynamic/Streaming Viewer using the default Query and Query Processor:

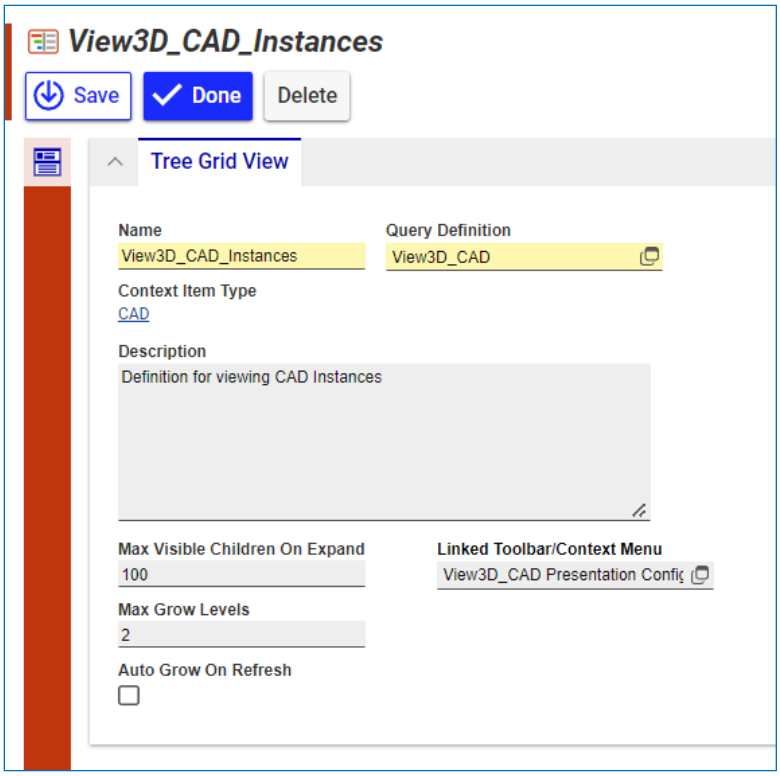


Figure 61.

1. Create a Tree Grid View Definition:
    a. Go to the **Table of Contents** --> **Administration** --> **Configuration** --> **Tree Grid View Definitions** --> **Create New Tree Grid View Definition**.
    b. Fill in the **Name** field.
    c. Fill in the **Description** field.
    d. Select the default **View3D_CAD** Query Definition.
    e. Click **Save**.

2. Modify the Tree Grid View:
    a. Click the **Editor** in the sidebar.
    b. Select the first four rows (**CAD**, **CAD Structure**, **CAD Instance CAD**), right-click, and click **Map**.

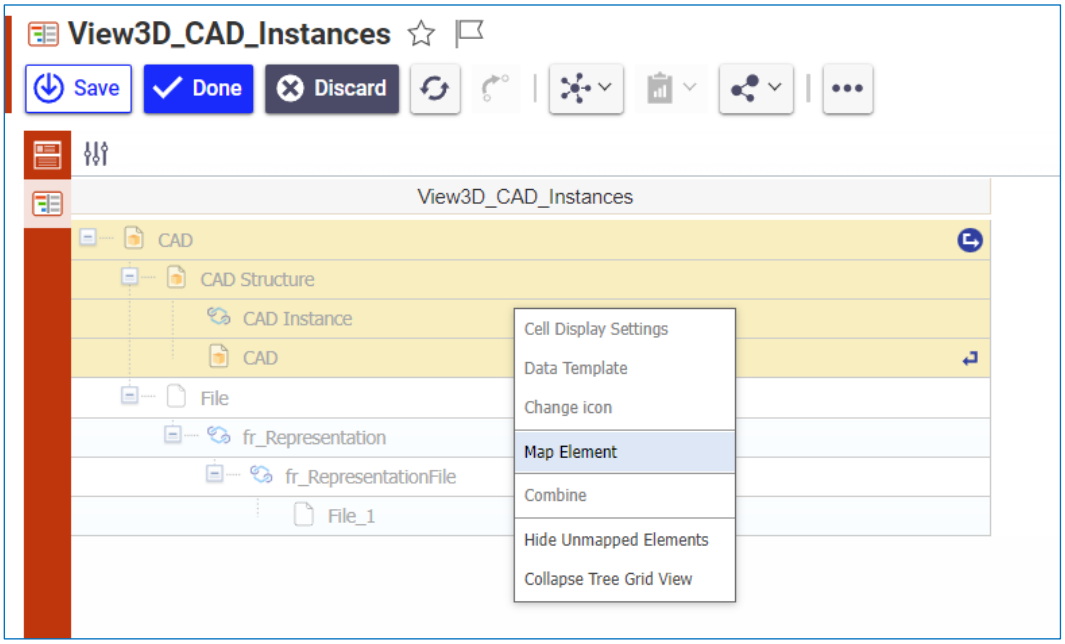


Figure 62.

    c. Select the second, third, and fourth rows (**CAD Structure**, **CAD Instance**, **CAD**), right-click, and click **Combine**.

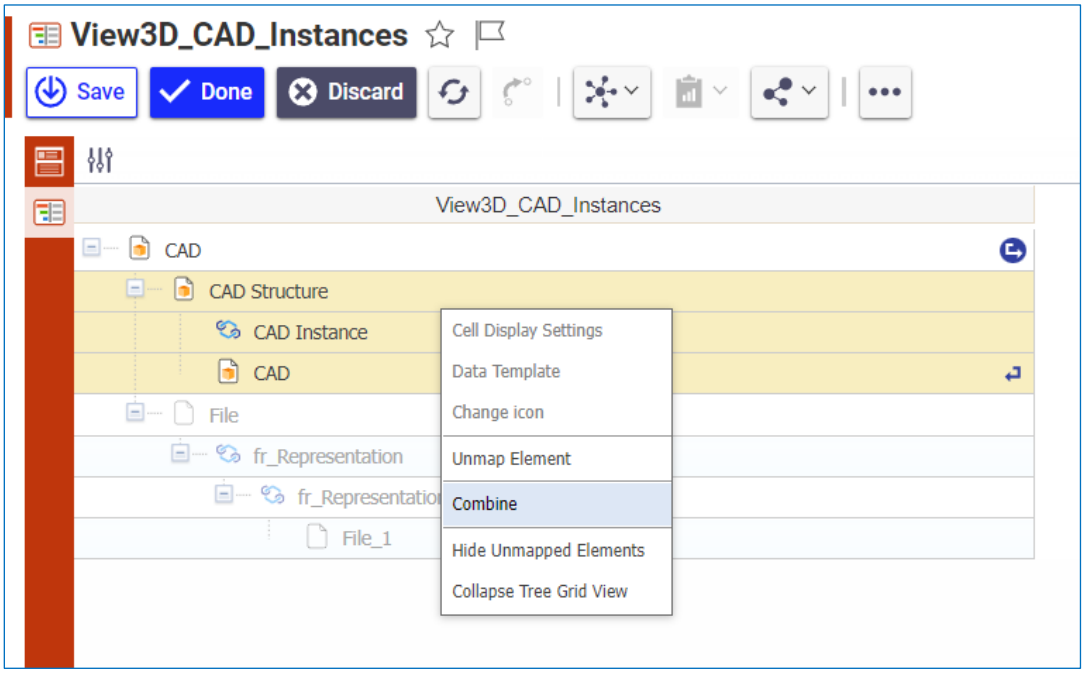


Figure 63.

d. Select the first row (**CAD**), right-click, and click **Cell Display Settings**.

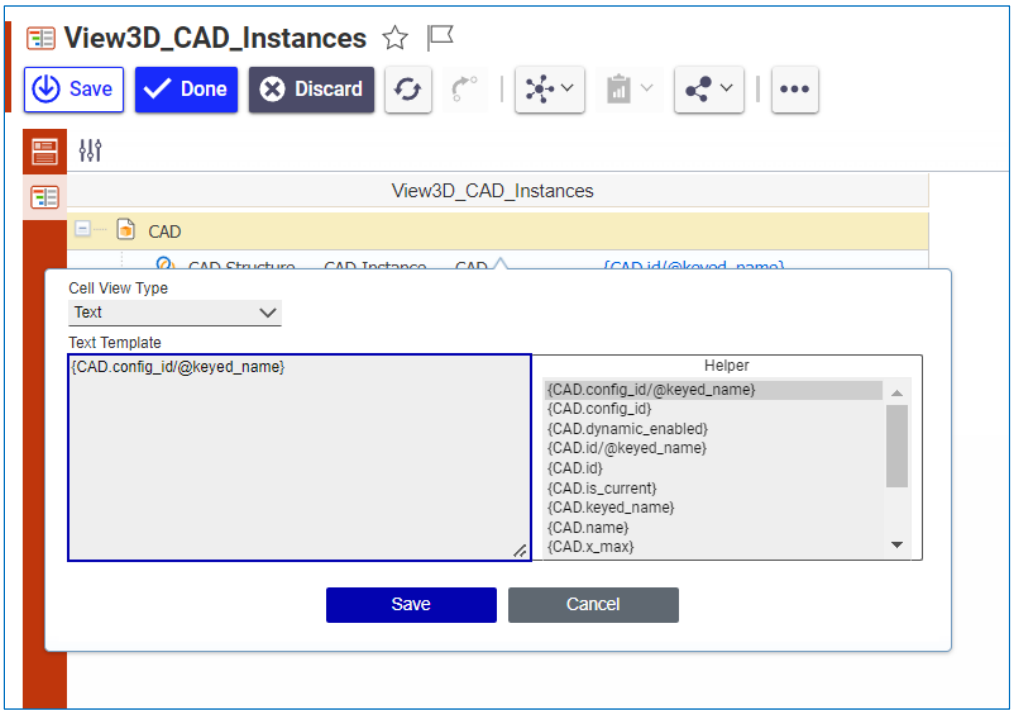


Figure 64.

e. Select **CAD.Keyed Name**.

f. Select the second row (**CAD Structure – CAD Instance – CAD**), right-click, and click **Cell Display Settings**.

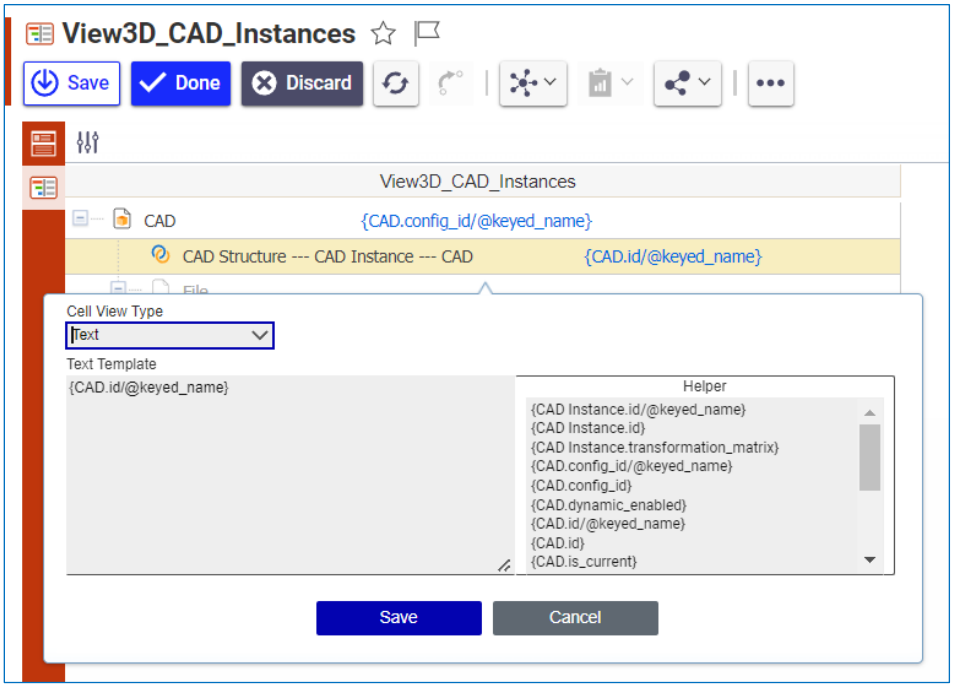


Figure 65.

g. Select **CAD.Keyed Name**.

h. Optional: select an icon for the row.

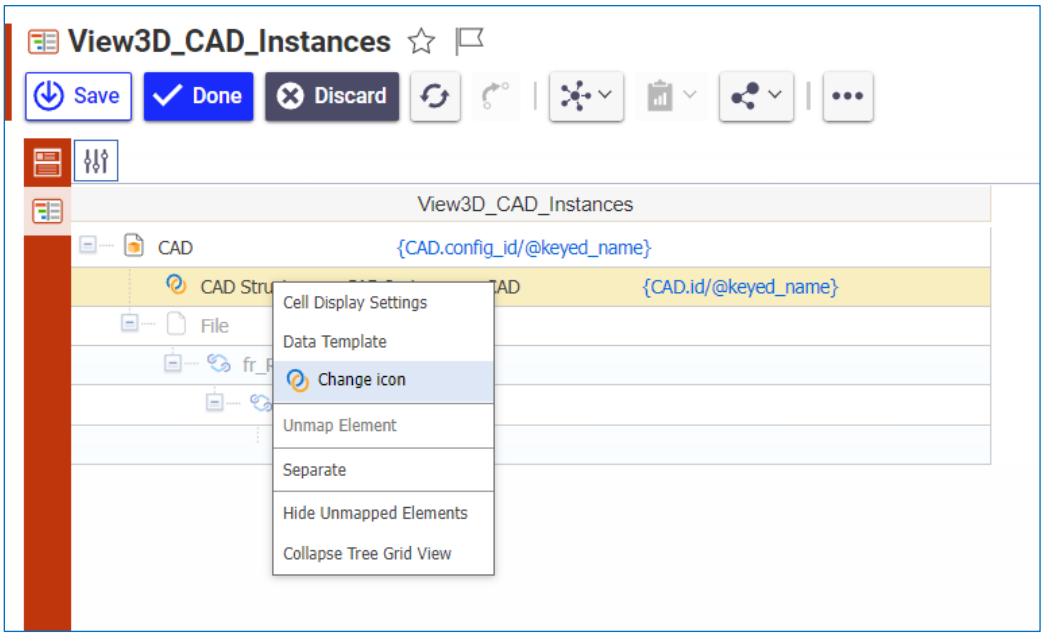


Figure 66.

3. Turn on Structure Resolution:

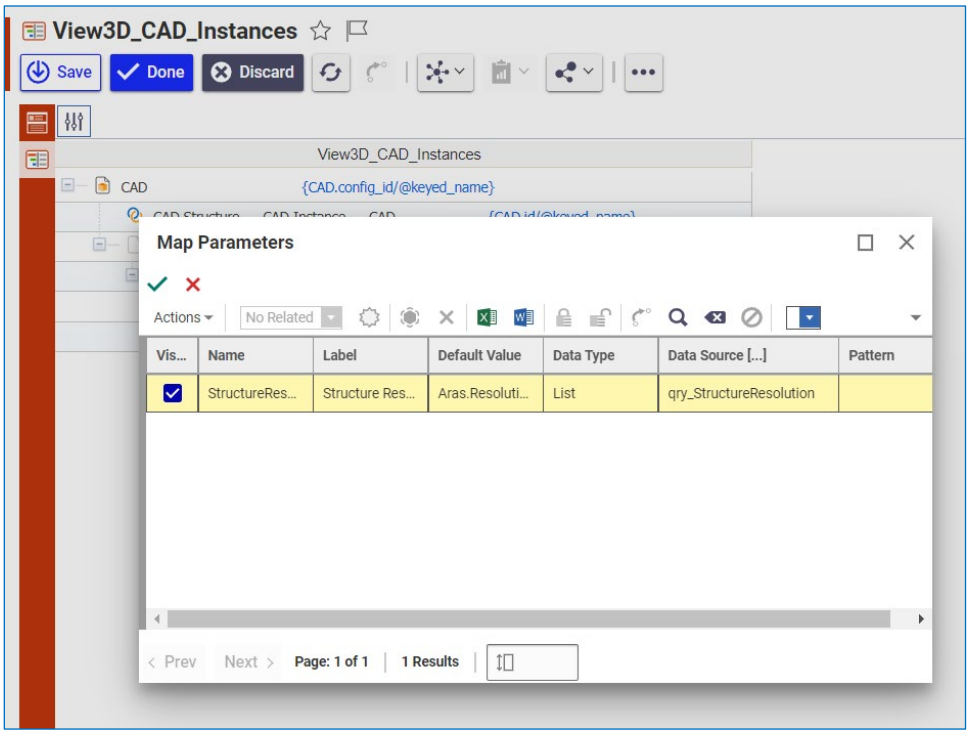


Figure 67.

a. Click **Map Parameters** 
b. Select the **Visibility** check-box next to the **StructureResolutionMode** parameter.
c. Set **Data Type** to **List**.
d. Set **Data Source** to **qry_StructureResolution**.
e. Click **Save** to close the **Map Parameters** dialog.
f. Click **Done**.

4. Create a Dynamic View Definition:
    a. Go to the **TOC** --> **Administration** --> **Configuration** --> **Dynamic View Definition** --> **Create New Dynamic View Definition**.

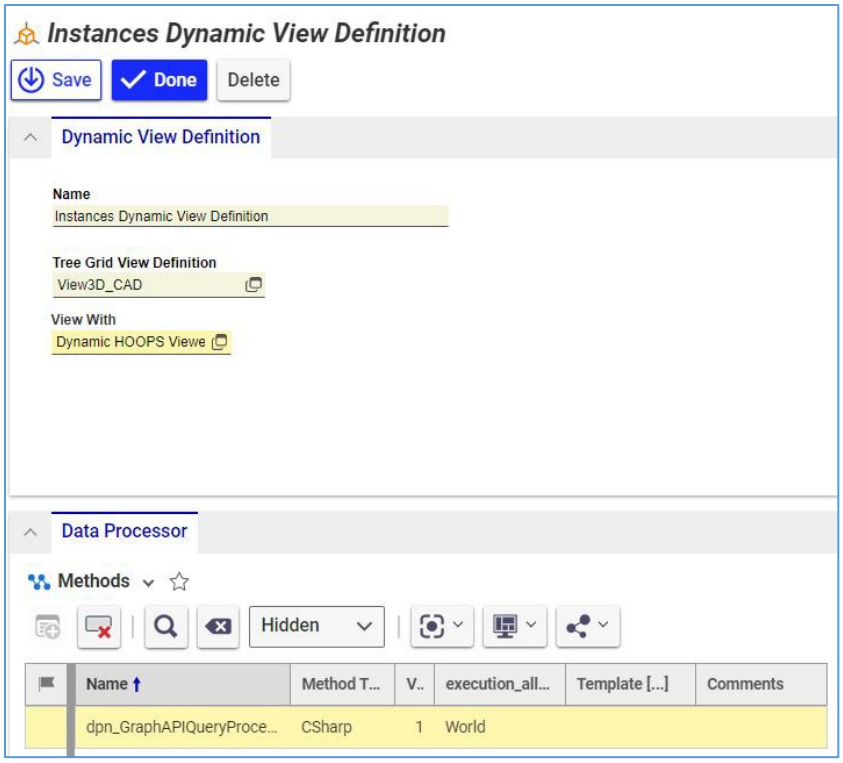


Figure 68.

    b. Fill in the **Name** field.
    c. Select the Tree Grid View Definition created in Step 1.
    d. Select a specific viewer (Dynamic or Streaming Viewer) in **View With** field.
    e. Select the default **dpn_GraphAPIQueryProcessor** data processor Method.
    f. Click **Done**.

5. Open a CAD Assembly with the **Dynamic Enabled** or **Streaming Enabled** flag and select the Dynamic View Definition created

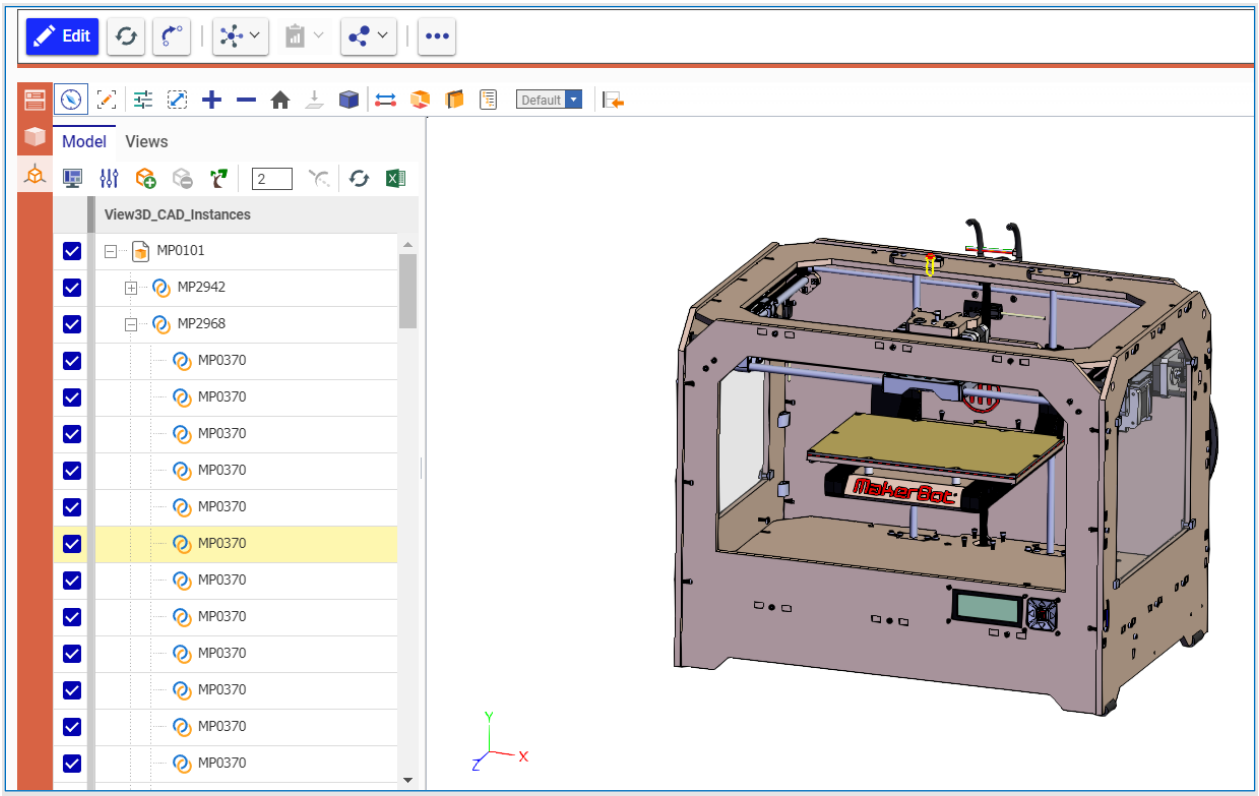


Figure 69.

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 or Streaming 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

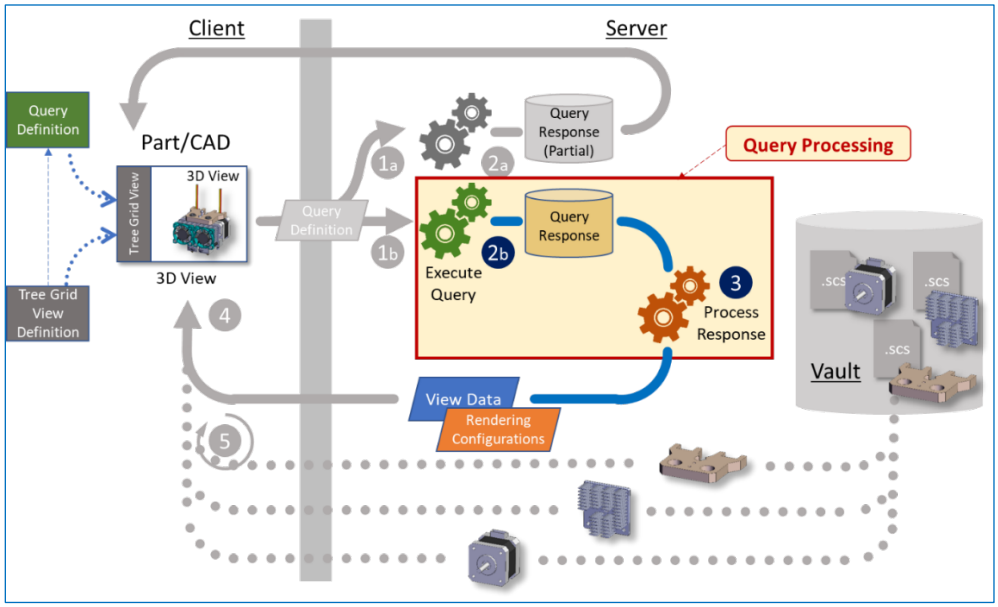


Figure 70. (Dynamic Viewer)

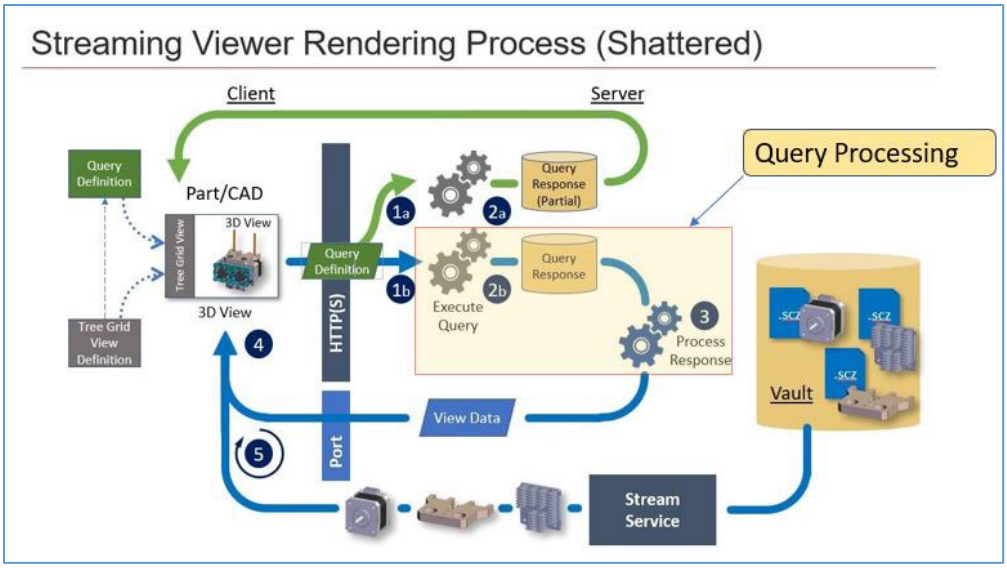


Figure 71. (Streaming Viewer)

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.

Note: Query Processing for Streaming Viewers require SCZ files

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 or Streaming Viewer icon to the sidebar so the Dynamic or Streaming 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.

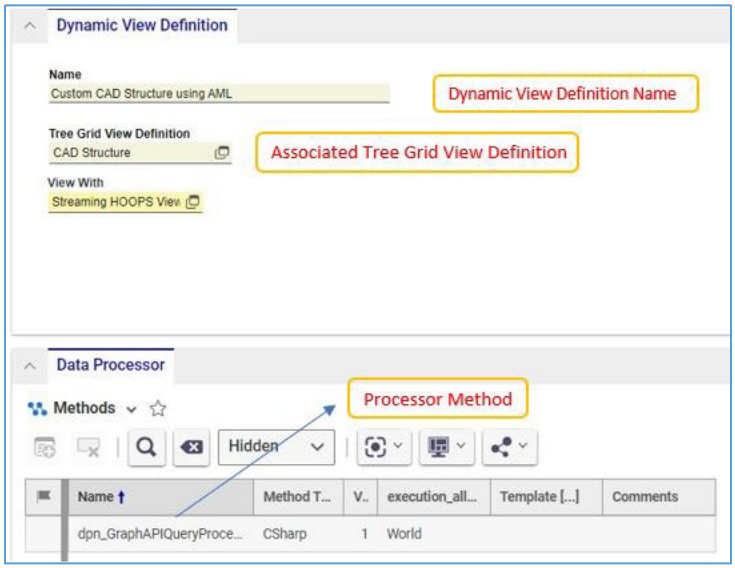


Figure 72.

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 **dpn_GraphAPIQueryProcessor** Mehtod.

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.

Note: It is recommended to make methods OS agnostic for use with Linux and Windows. The main incompatibility issues in operating systems that need to be considered when implementing the method are path case-sensitivity, path separators, OS-specific line-endings, OS-specific code. For more information about cross-platform development, see section *2.3 Cross-platform development* in the *Aras Innovator 26 - Programmer's Guide*.

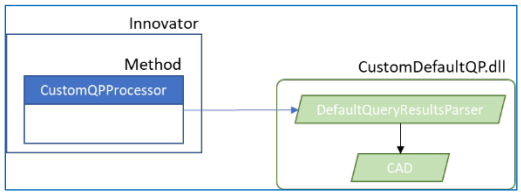


Figure 73.

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=Csharpe:Aras.Public.Events.EventHandler<Aras.DynamicModelViewer.DataModel.DataProcessorArgs>;

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

3 //Get Innovator Connection
  var connection = Innovator.getConnection();

4 //Get Query Definition Helper
  Aras.DynamicModelViewer.DataModel.Helpers.QueryDefinitionHelper qdHelper = eventArgs.QueryDefinitionHelper;

5 // Get Query Definition Item
  Item qdItem = qdHelper.GetQueryDefinitionItem(eventArgs.QueryDefinitionId);

6 // Apply Query Definition and get result
  Item qdResult = qdHelper.GetQueryDefinitionResultItem(qdItem, eventArgs.ItemId, eventArgs.QueryDefinitionParams);

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

8 //Process the query response
  Var result = new Aras.DynamicModelViewer.DataModel.QueryProcessingResult();
  Result.SetProductOccurrenceList(defQryResParser.processQueryResults(qdResult));

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

Figure 74.

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 Innovator connection.
4. Retrieves the Query Definition Helper used for this Query Processor.
5. Retrieves the Query Definition Item.
6. Execute the Query Definition using the given Item ID and the query parameters used.
7. 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/Streaming Viewer.
8. The results of the processing – Product Occurrence List – is constructed by the custom DLL and stored in the `QueryProcessingResults` Object…
9. …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 53 and discuss each relevant XML Element when processing.

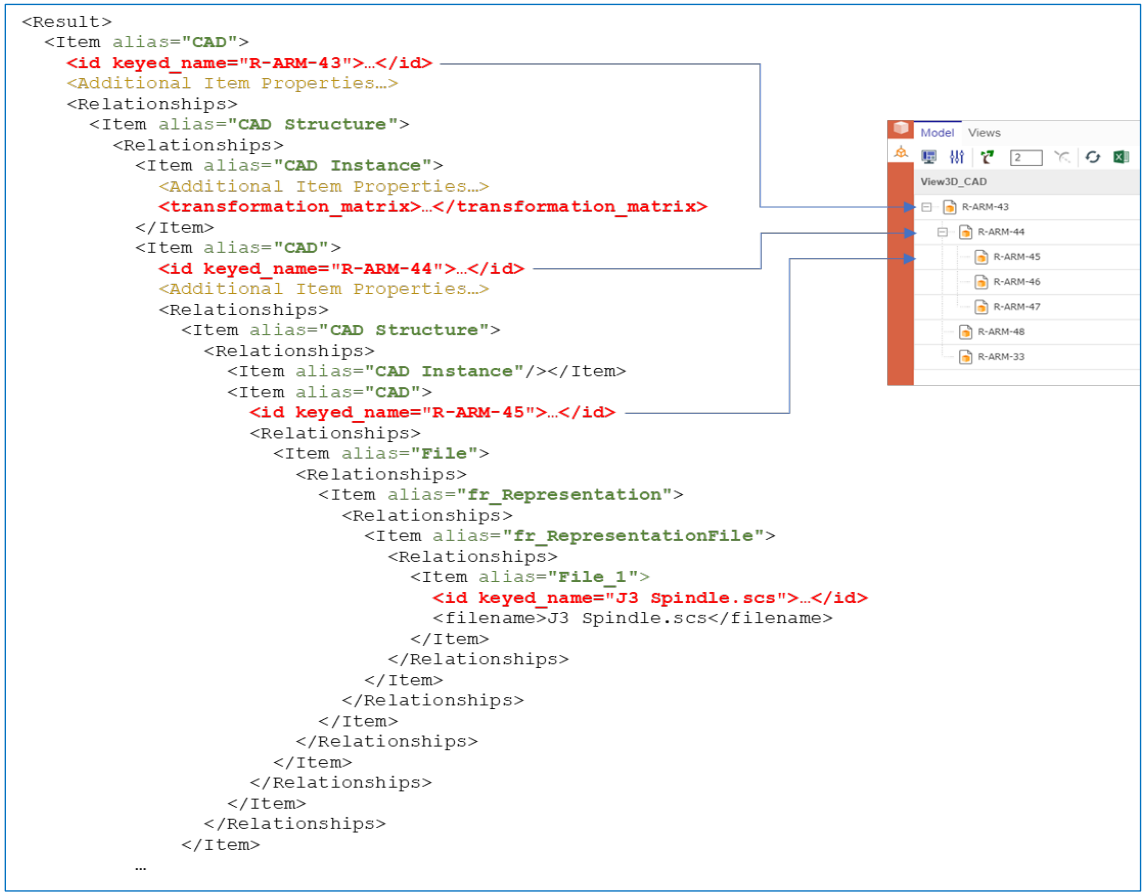


Figure 75.

Figure 53 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. The `<Additional Item Properties>` XML Element 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 section 5.1.2 Base Query Definition). 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 8.1.

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.

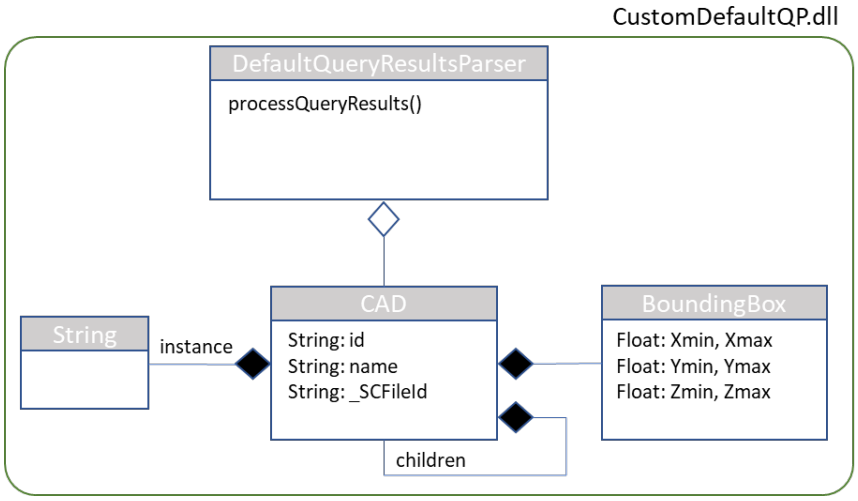


Figure 76.

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

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.

It is recommended to make methods OS agnostic for use with Linux and Windows. The main incompatibility issues in operating systems that need to be considered when implementing the method are path case-sensitivity, path separators, OS-specific line-endings, OS-specific code. For more information about cross-platform development, see section *2.3 Cross-platform development* in the *Aras Innovator 26 - Programmer's Guide*.

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

```
   internal void processCAD(QueryBuilderNode qbItem)
    {
      QryRefId = qbItem.QueryItem.RefId; // Query Item Reference ID
      try
      {
        ID = qbItem.GetProperty("id"); // required
        Name = qbItem.GetProperty("name"); // required
      }
      catch
      {
        throw new ArgumentException("Query Definition is not defined properly: 'id' and
'name' Properties are required for CAD Items");
      }

      // Bounding Box Data. Alternate values ensure that there is a non-empty
      // bounding volume defined
      _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.      // Add Transformation Matrix
        poInst.TransformationMatrix = cadInstInfo.XForm;

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


8.      poInst.Name = c.Name;
9.      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);
          }

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

          // Recurse through child hierarchy
12.       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 55.
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 8.1.
4. Set the bounding box values for each of the *min* and *max* values for X, Y, and Z
5. Add a Rendering Configuration. This is optional, but necessary to add one or more View Modes to the Dynamic or Streaming Viewer (section 6.2.7). Rendering Configurations add alternate rendering parameters for the geometry associated with the Product Occurrence. This includes color and opacity (transparency). Together they can be used to visually isolate/highlight certain 3D geometry to show some information about the model or about related Items. There can be multiple rendering configurations for each Product Occurrence, although each must have a unique Name.

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

   Note: For each Product Occurrence Instance, it is necessary to set the service properties `SetServiceProperty()` method. See the full example in Section 8.1.

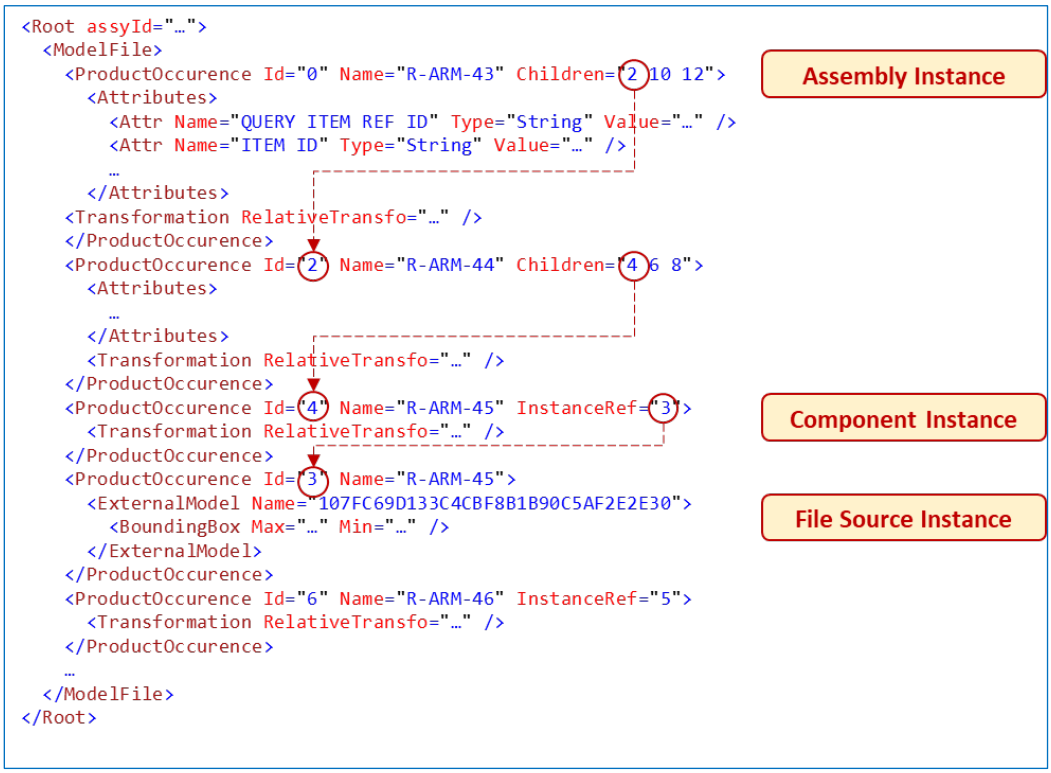
8. The name used for the Product Occurrence. It's helpful to either use the document number or name of the CAD Item.
9. For the CAD Data Model Object used, if there are no children, then it is assumed that the CAD Item is a component; in which case it should have an associated view file. In this case,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

    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.

11. If a valid parent was passed into the method, then add the newly created Occurrence to it.
12. 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 like the following example:




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

Figure 77.

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 Processing Combined Rows

Processing combined rows in a custom query processor, such as when synchronizing CAD instances (described in section 5.2.7), is possible by assigning the required properties (Query Reference Id and Item Id) for each combined row item to the relevant Product Occurrence instance.

For example, when combining the CAD Instance, CAD Structure, and CAD rows, the Items for all three rows will need to have `SetServiceProperty()` called to be able to synchronize the combined rows between the TGV and the Viewer.

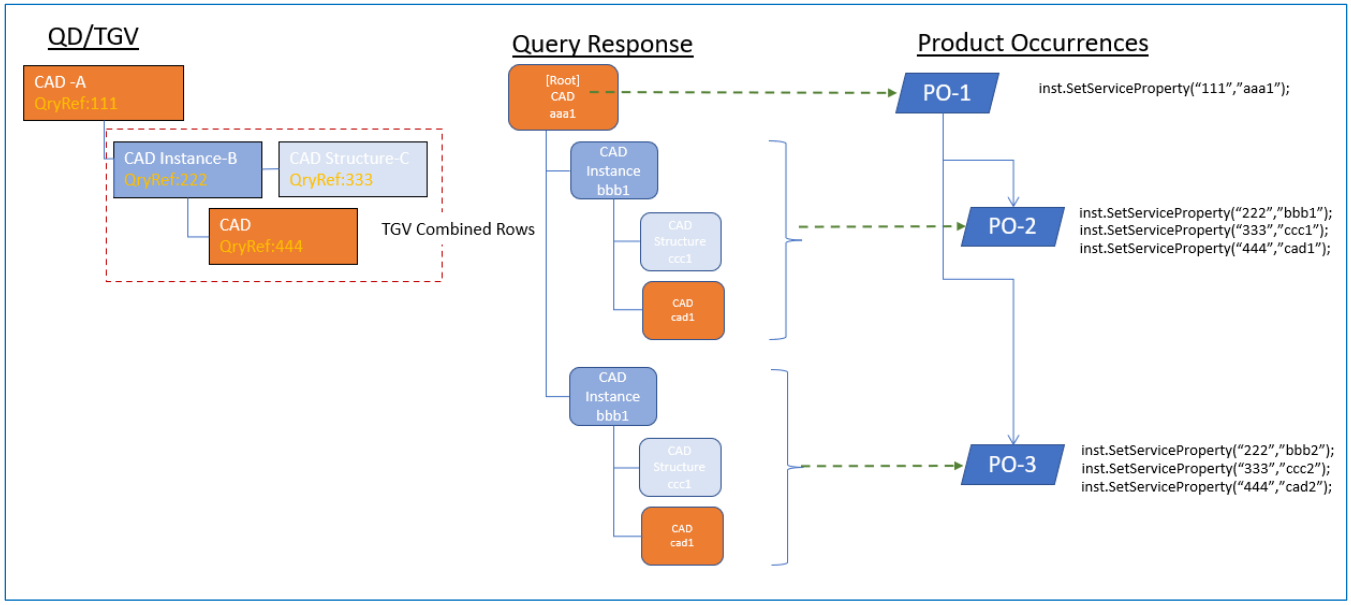


Figure 78.

6.2.7 Rendering Configurations

Rendering Configurations are used to define alternate rendering properties (color, transparency) to apply to 3D geometry in the Dynamic or Streaming 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 or Streaming Viewer via the View Modes dropdown on the viewer toolbar.

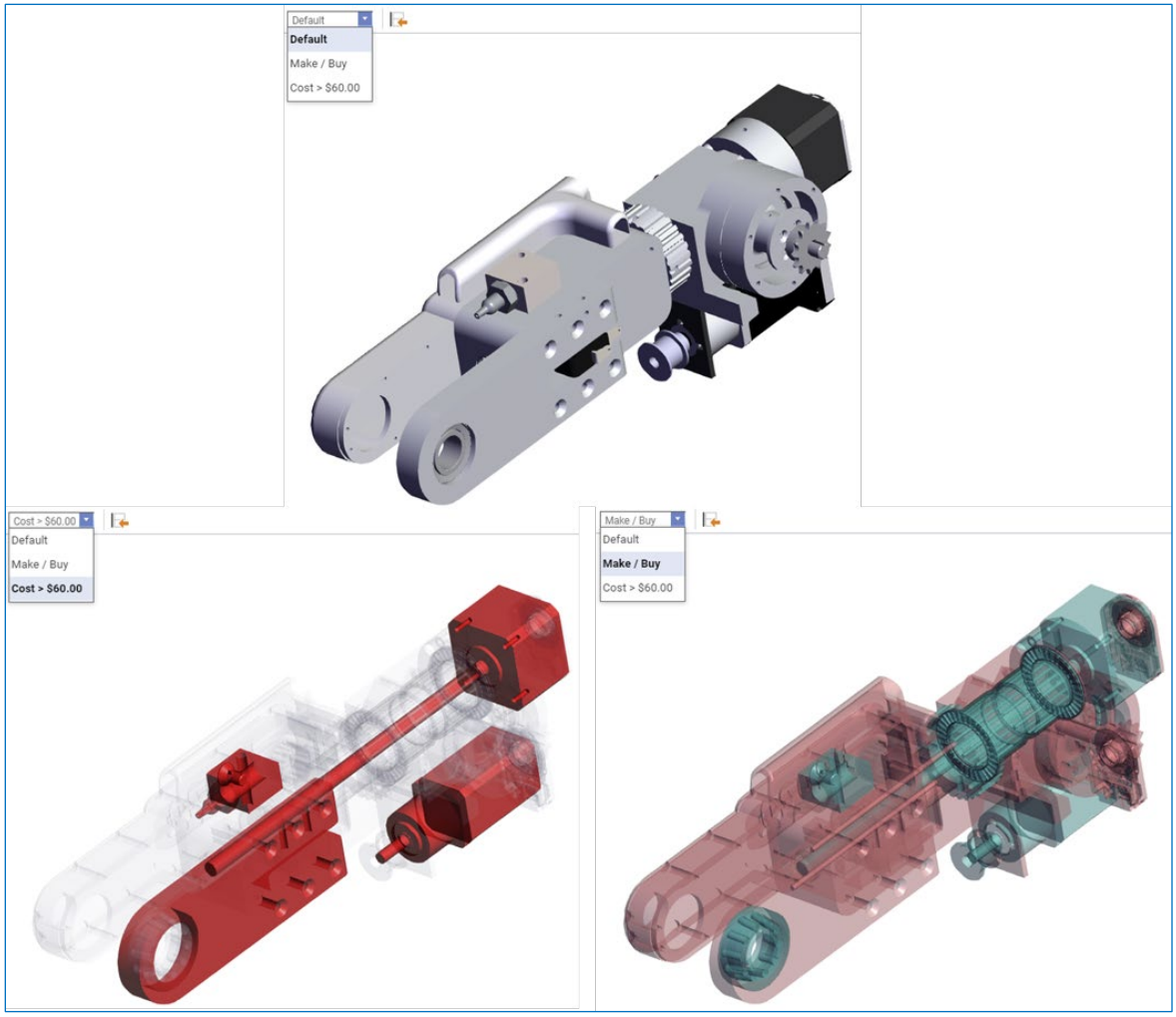


Figure 79. View Modes

Figure 57 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 or Streaming 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 8.1.

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 or Streaming 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. See the following figure for an 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 80.

Note: Due to the differences between Windows and Linux file systems it is required to use OS-specific path separator in paths. Remember that the Linux file system is case sensitive so there is significant difference between file names **./path/to/file.xml** and **./path/to/File.xml**. For more information about cross-platform development, see section *2.3 Cross-platform development* in the *Aras Innovator 26 - Programmer's Guide*.

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 0 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 and Streaming Viewer contain a directory `/QueryProcessorAPI` with API help files describing the classes/methods of the API.

7 DynamicView Client-Side API

The `DynamicView` base class is a JavaScript (JS) object to execute functions associated with the Dynamic and Streaming Viewer, such as **Refresh**, **Isolate**, **Hide All**, etc. Its instances are accessible via JS Methods invoked by CUI Action Items for the Dynamic and Streaming Viewer Tree Grid View, Toolbar, and 3D View Canvas.

Developers can use the `DynamicView` client-side API to customize context menu commands and toolbar buttons to execute functionality that uses or processes 3D View or 3D Model data. The subsections of this section describe the `DynamicView` functions. Section 8.2 Customization Examples with DynamicView API provides customization examples.

7.1 Accessing DynamicView

To access the `DynamicView` base class, use the following code snippet:

```
// MethodTemplateName=CUI;
const dynamicView = options.dpnContext.getDynamicView();
```

7.2 dynamicView.addModel(itemIds)

The `dynamicView.addModel(itemIds)` function adds a model to a Dynamic or Streaming 3D Viewer scene as a separate geometry in 3DV and tree node on Tree Grid View.

The function has the following parameter:

- `itemIds`: an array of respective item IDs that contain 3D geometry.

7.3 dynamicView.removeModel()

The `dynamicView.removeModel()` function removes a model from a Dynamic or Streaming 3D Viewer scene and Tree Grid View.

7.4 dynamicView.isolateSelectedNodes()

The `dynamicView.isolateSelectedNodes()` function does the following:

- Fits selected nodes into a Dynamic or Streaming 3D Viewer scene.
- Makes unselected nodes transparent.

7.5 dynamicView.setVisibilitySelectedNodes(isVisible)

Depending on a passed parameter value, the `dynamicView.setVisibilitySelectedNodes(isVisible)` function toggles the following:

- The visibility of selected nodes in a Dynamic or Streaming 3D Viewer scene.
- The visibility state of selected nodes in the Tree Grid View.

The function has the following parameter:

- `isVisible`: a Boolean flag:
  - `true`: to show selected nodes in a Dynamic or Streaming 3D Viewer scene and set their state to `visible` in the Tree Grid View.
  - `false`: to hide selected nodes in a Dynamic or Streaming 3D Viewer scene and set their state to `hidden` in the Tree Grid View.

7.6 dynamicView.hideAllOtherNodes()

The `dynamicView.hideAllOtherNodes()` function does the following:

- Fits selected nodes into a Dynamic or Streaming 3D Viewer scene.
- Hides unselected nodes in the Dynamic or Streaming 3D Viewer scene.
- Sets the visibility state of unselected nodes to `hidden` in the Tree Grid View.

7.7 dynamicView.fitAllNodes()

The `dynamicView.fitAllNodes()` function fits all nodes into a Dynamic or Streaming 3D Viewer scene.

7.8 dynamicView.resetView()

The `dynamicView.resetView()` function resets a model and Dynamic or Streaming 3D Viewer scene to their original states:

- Sets the camera to its default position.
- Makes all nodes visible in the Dynamic or Streaming 3D Viewer scene.
- Sets the visibility state of all nodes to `visible` in the Tree Grid View.

7.9 dynamicView.displayAllNodes()

The `dynamicView.displayAllNodes()` function does the following:

- Makes all nodes visible in the Dynamic or Streaming 3D Viewer scene.
- Sets the visibility state of all nodes to `visible` in the Tree Grid View.

7.10 dynamicView.updateContextMenu()

The `dynamicView.updateContextMenu()` function updates the visibility of context menu elements according to the nodes currently selected in the Tree Grid View.

8 Using JT/STEP Converter

This section describes the installation steps of an optional JT/STEP Converter. This installation requires a System Administrator to make changes on the server where Aras Conversion Server is installed.

Note: There is no Out of the Box Conversion Rule that will implement the JT/STEP Converter. In order to make use of this capability, custom **Conversion Rules** will need to be created. Please refer to Aras Innovator 26 - Example for Developing Conversion Server Tasks for guidelines.

The following steps outline the process of installing the JT/STEP Converter:

1. Copy and unzip **Aras 3D Visualization 26 CD Image** package on the server where Conversion Server is installed.
2. Go to **Aras 3D Visualization 26 CD Image\Packages** directory and copy the **HOOPS Exchange** folder.
3. Paste the copied folder to a permanent location on the server.
   Example: C:/HOOPS Exchange
4. Open **ConversionServerConfig.xml** file for edit and add the following tags:
   a. In the child `sectionGroup name="ConverterSettings"` tag of the `configSections` tag, add the following `section` tags with attributes:

```
<configSections>
   <sectionGroup name="ConverterSettings">
      …
<section name="DpnCadConverterStepJt"
type="Aras.ConversionFramework.Converter.Hoops.Configuration.HoopsCo
nverterConfiguration, ArasCadConverter" />
   </sectionGroup>
</configSections>
```

   b. In the child `Converters` tag of the `ConversionServer` tag, add the following `Converter` tags with attributes:

```
<ConversionServer>
   <Converters>
      …
<Converter name="JT Step CAD Converter"
type="Aras.CadConverter.StepJtCadConverter, ArasCadConverter" />
   </Converters>
</ConversionServer>
```

c. In the `ConverterSettings` tag, add the following tags with child tags and attributes:

```
<ConverterSettings>
    …
    <DpnCadConverterStepJt>
          <Application converterPath="C:\HOOPS
    Exchange\bin\win64_v142" />
    </DpnCadConverterStepJt>
</ConverterSettings>
```

5. Save and close **ConversionServerConfig.xml** file.

9 Appendix

9.1 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,
          poInst.SetServiceProperty(c.QryRefId, c.ID); // add service properties
          poList.Add(poInst); // add to the PO list
        }

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


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

```
        poInst.Attributes.Add(new ProductOccurrenceAttr("ITEM ID", c.ID));

        // bounding box
        poInst.ProductOccurrenceSource.BoundingBox.Max.X = c.BBox.MaxX;
        poInst.ProductOccurrenceSource.BoundingBox.Max.Y = c.BBox.MaxY;
        poInst.ProductOccurrenceSource.BoundingBox.Max.Z = c.BBox.MaxZ;
        poInst.ProductOccurrenceSource.BoundingBox.Min.X = c.BBox.MinX;
        poInst.ProductOccurrenceSource.BoundingBox.Min.Y = c.BBox.MinY;
        poInst.ProductOccurrenceSource.BoundingBox.Min.Z = c.BBox.MinZ;

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

       // **** ADD SetServiceProperty METHOD CALLS ****
       poInst.TransformationMatrix = cadInstInfo.XForm;
       poInst.SetServiceProperty(cadInstInfo.RefID, cadInstInfo.ID);
       poInst.SetServiceProperty(c.CADStrQryRefId, c.CADStructureID);
       poInst.SetServiceProperty(c.QryRefId, c.ID);
      // **** ADD SetServiceProperty METHOD CALLS ****

        poInst.Name = c.Name;
        if (c.Children.Count == 0 && c.SCFileID != null) // Assume this is a component
part
        {
          poInst.ProductOccurrenceSource.Name = c.SCFileID; // use File Item Id for name
          poInst.ProductOccurrenceSource.FileReferenceByTypeMap.Add(SourceModelType.Scs,
c.SCFileID);
        }
        if (c.Children.Count == 0 && c.SCFileID == null)
          poInst = null;  // Invalid - no view file attached
        else
          parent.Children.Add(poInst);

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

    } // build()

  } // class DefaultQueryResultsParser
} // namespace CustomDefaultQP
```

The following `CAD` class is used to collect information from the Query Results. It contains the `processCAD` method discussed in section 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 information for each CAD Instance Query Item
    /// </summary>
    class CADInstanceInfo
    {
        public CADInstanceInfo()
        {
            XForm = "1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1";  // Default Identity Matrix
            ID = "0";
        }

        /// <summary>
        /// CAD Instance transformation matrix
        /// </summary>
        internal String XForm { get; set; }

        /// <summary>
        /// CAD Instance Id
        /// </summary>
        internal String ID { get; set; }

        /// <summary>
        /// CAD Query Reference Id
        /// </summary>
        internal String RefID { get; set; }

    } // class CADInstanceInfo


  /// <summary>
  /// Stores all data necessary for creating 3D View data assuming
  /// it is contained within 'CAD' Query Items
  /// </summary>
  class CAD
  {
    private BoundingBox _BBox; // Bounding volume
```

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

    /// <summary>
    /// Processes a given CAD Item contained within the given
<code>QueryBuilderNode</code>
    /// </summary>
    /// <param name="qbItem">Item containing properties for CAD Items</param>
    /// <exception cref="ArgumentException">Required Properties not included in
results</exception>
    internal void processCAD(QueryBuilderNode qbItem)
    {
      QryRefId = qbItem.QueryItem.RefId;
      try
      {
        ID = qbItem.GetProperty("id"); // required
        Name = qbItem.GetProperty("name"); // required
      }
      catch
      {
        throw new ArgumentException("Query Definition is not defined properly: 'id' and
'name' Properties are required for CAD Items");
      }

      // Bounding Box Data. Alternate values ensure that there is a non-empty
      // bounding volume defined
      _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<CADInstanceInfo > curInstances = new List<CADInstanceInfo>();      CAD
nextCAD = new CAD();
         nextCAD.CADStructureID = qbItem.ItemId;
         nextCAD.CADStrQryRefId = qbItem.QueryItem.RefId;


      // Loop through all CAD Instances and CAD Items
      foreach (var child in qbItem.ChildNodes)
      {
        try
        {
          if (child.QueryItem.Alias == "CAD Instance" &&
              child.TryGetProperty("transformation_matrix", out string tmpXForm) &&
              tmpXForm != null) {
             CADInstanceInfo cadInfo = new CADInstanceInfo();
             cadInfo.XForm = tmpXForm;
             cadInfo.ID = child.ItemId;
             cadInfo.RefID = child.QueryItem.RefId;
             curInstances.Add(cadInfo);
}
          if (child.QueryItem.Alias == "CAD")
          {
            nextCAD.processCAD(child);
            Children.Add(nextCAD);
          }
        } // try
        catch { }
      } // foreach()
      if (curInstances.Count > 0)
        nextCAD.Instances = curInstances;
    } // _processCADStructure(QueryBuilderNode qbItem)
```

```
    /// <summary>
    /// Returns the first found <code>QueryBuilderNode</code> with the
    /// given alias that is a descendant of the given Item.  Note that if
    /// the given Item has the alias, it is returned
    /// </summary>
    /// <param name="qbItem">Item containing descendants to search</param>
    /// <param name="alias">Alias Name of Item to search for</param>
    /// <returns>NULL, if not found; first found node with given alias
otherwise</returns>
    private QueryBuilderNode _findChildWithAlias(QueryBuilderNode qbItem, String alias)
    {
      if (qbItem.QueryItem.Alias == alias)
        return (qbItem);
      else
      {
        foreach (var child in qbItem.ChildNodes)
        {
          QueryBuilderNode descItem = _findChildWithAlias(child, alias);
          if (descItem != null)
            return (descItem);
        } // for()
      } // else
      return (null);
    } // findChildWithAlias(QueryBuilderNode qbItem, String alias)

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


    /// <summary>
    /// Returns the name of the CAD Query Response Item
```

```
    /// </summary>
    internal String Name { get; private set; }

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

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


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

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


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

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

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

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

    /// <summary>
    /// Bounding Volume
    /// </summary>
    internal BoundingBox BBox { get { return _BBox; } }

  } // class CAD
} // namespace CustomDefaultQP
```

9.2 Customization Example with DynamicView API

With the `DynamicView` client-side API, developers can customize the Dynamic or Streaming Viewer context menu commands and toolbar buttons. This section provides an example of such customization to give a solid understanding of the customization process. For the `DynamicView` client-side API description, see section 7 DynamicView Client-Side API.

The customization process has two stages:

1. Writing custom JS functions for the required functionality.
2. Creating context menu commands using the standard Aras Innovator capabilities.

The example shows how to create the custom **Add Item(s) To Change** TGV Context Menu item that calls the **Choose Change Item** dialog for selected nodes.

9.2.1 Writing a Custom JS Function

1. In **Innovator/Client/Solutions/3DV/Scripts/Controls/TGV/DynamicView.ts** add the following function:

```
this.getSelectedItemsHandler = function () {
    var ccItem;
    const tgvGrid = mainPageExtension.getTgvGrid();
    const treeGrid = mainPageExtension.getTreeGrid();
    const selectedRowsIds = treeGrid.settings.selectedRows;
    const count = selectedRowsIds ? selectedRowsIds.length : 0;

    // Add a dummy item to start the result collection (removed later)
    var resultItem = aras.newIOMItem("DeleteMe", "DeleteMe");
    for (let i = 0; i < count; i++) {
        const rowId = selectedRowsIds[i];
        const selectedRow = tgvGrid.getRow(rowId);
        const dataStr = selectedRow.cells[0].data;
        if (dataStr) {
            const data = JSON.parse(dataStr);
            var thisType = data.type;    // this.getType();
            var thisId = data.id;
            var relshipItemType;
            var sourceItemTypeName;
            var isRelationship = false;

            // Check whether the method is being called from a relationship
grid, the main grid or on an
            // individual item and build an array of item ids
            var relTypeId = aras.getRelationshipTypeId(thisType);
            if (relTypeId) {
                isRelationship = true;

                relshipItemType = aras.getRelationshipType(relTypeId);
                var sourceItemType =
aras.getItemTypeDictionary(relshipItemType.getProperty("source_id"), "id");
                sourceItemTypeName = sourceItemType.getProperty("name");
            }
```

```
                        // When starting from a relationship, get the parent item from
the cache and then retrieve the related item
                        if (isRelationship) {
                            var parentId = relshipItemType.getProperty("source_id", "");
                            var parentItem = aras.getItemById(sourceItemTypeName,
parentId, 0);
                            if (parentItem) {
                                ccItem =
parentItem.selectSingleNode("Relationships/Item[@id='" + thisId + "']");

                                //If item isn't cached, get
                                if (!ccItem) {
                                    ccItem = aras.getItemById(thisType, thisId, 0);
                                }

                                //get related Part
                                if (ccItem) {
                                    ccItem = aras.getRelatedItem(ccItem);
                                }
                            }
                        }
                        else {
                            ccItem = aras.getItemById(thisType, thisId, 0);
                        }

                        // Add the item to the collection to be returned
                        if (ccItem) {
                            if (aras.isNew(ccItem)) {
                                aras.AlertError(aras.getResource("PLM",
"affecteditem.add_not_saved_item", aras.getKeyedNameEx(ccItem)));
                                return;
                            }
                            var tempItem = aras.newIOMItem("", "");
                            tempItem.loadAML(ccItem.xml);
                            resultItem.appendItem(tempItem);
                        }
                    }
                }

                if (resultItem.getItemCount() < 2) { return; }

                // Remove the dummy item
                resultItem.removeItem(resultItem.getItemByIndex(0));

                // Set the ChangeItem attribute for debugging purposes and return the
results
                resultItem.getItemByIndex(0).setAttribute("ChangeItem", "Pending");


                var methodArgs = {};
                methodArgs.results = resultItem;
                results = aras.evalItemMethod(
                    'PE_ChooseCMItem',
                    ccItem,
                    methodArgs
                );
```

```
        return resultItem;
    };
```

This function calls a new function which is an adapted version of the standard **PE_GetSelectedItems** Server Method.

2. Create the **dpn_GetSelectedItems** Method of the **JavaScript** Method Type with its execution allowed to the **World** Identity:

```
// MethodTemplateName=CUI;
const dynamicView = options.dpnContext.getDynamicView();
dynamicView.getSelectedItemsHandler ();
```

This Method will be a menu item click handler.

9.2.2 Creating a New TGV Context Menu Item

1. Go to **Contents** --> **Administration** --> **Configuration** --> **Tree Grid Views** and open the **View3D_CAD** Item.

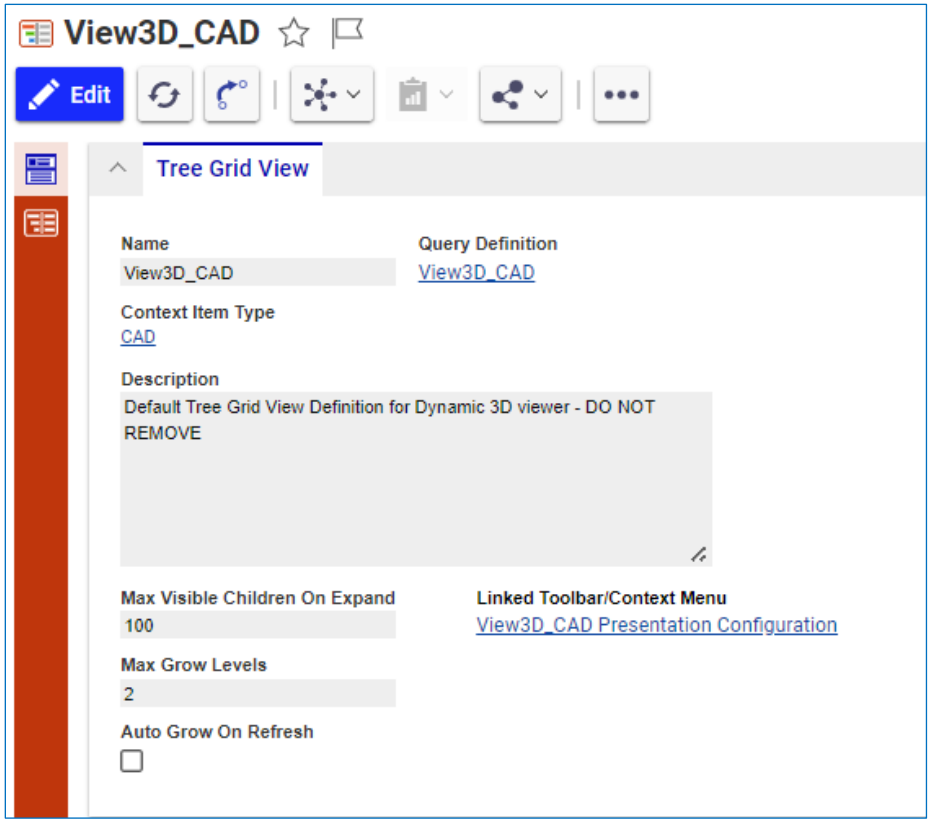


Figure 81.

2. Open the **View3D_CAD Presentation Configuration** Item.

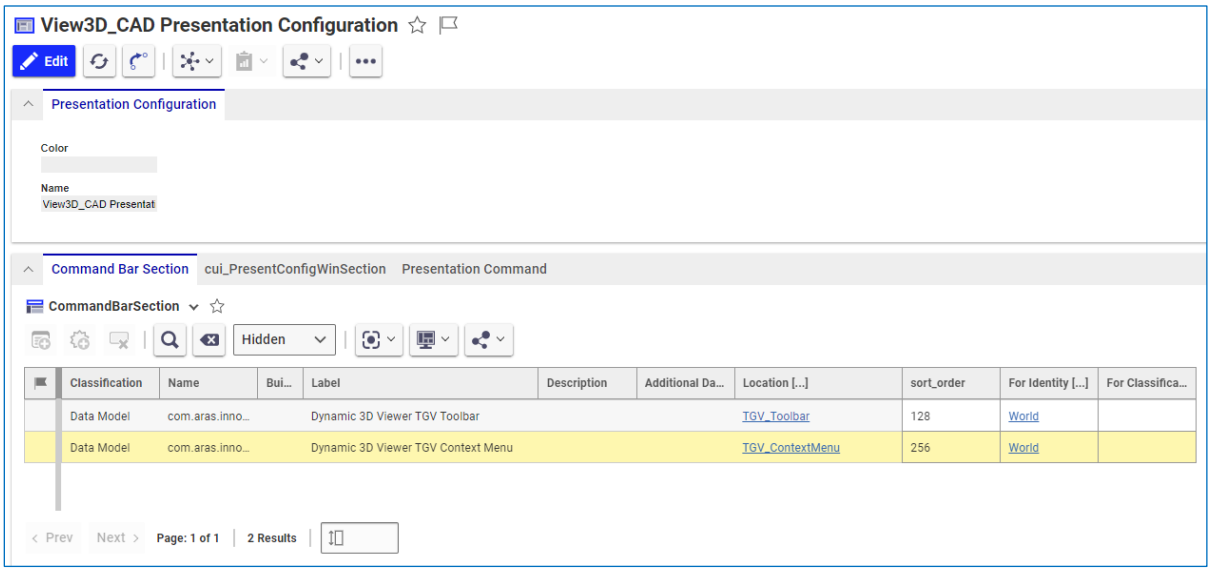


Figure 82.

3. Open the **Dynamic/Streaming 3D Viewer TGV Context Menu** Item from the **Command Bar Section** Relationships Grid.
4. Add a new **CommandBarItem** Relationship Item of the **Menu** ItemType.

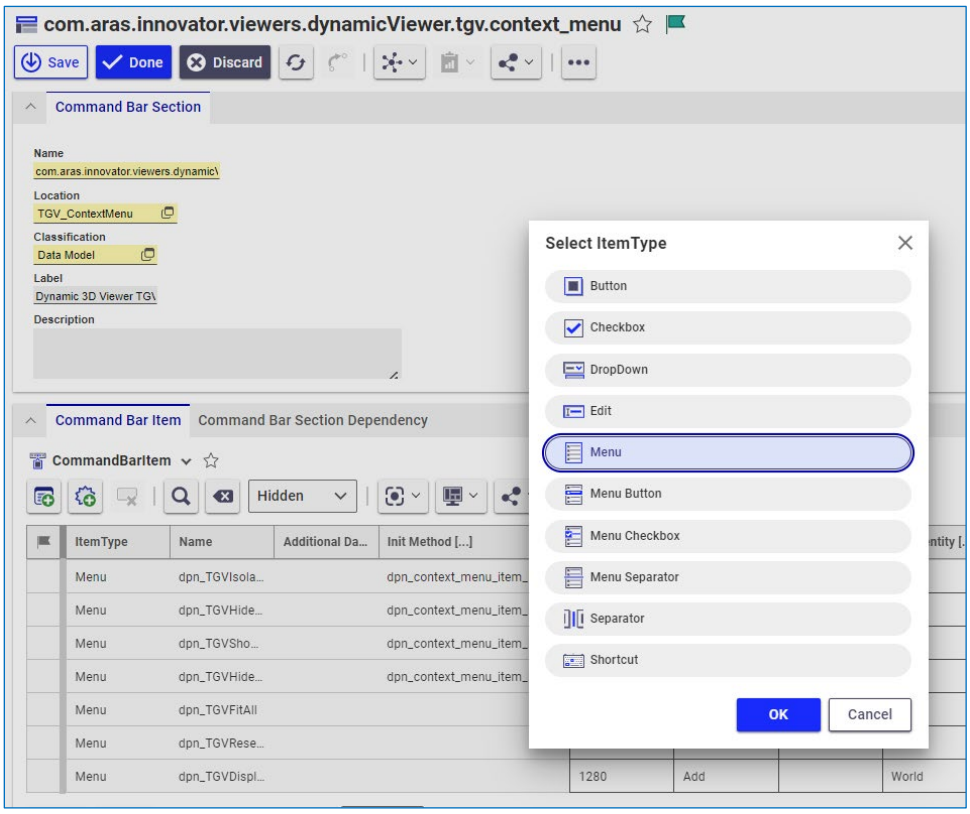


Figure 83.

5. Set the properties of the new **CommandBarItem** Item as follows:
    a. **Name**: **dpn_GetSelectedItems**
    b. **Label**: **Add Item(s) To Change**
    c. **Init Method**: **dpn_context_menu_item_init**
    d. **Click Method**: **dpn_GetSelectedItems**

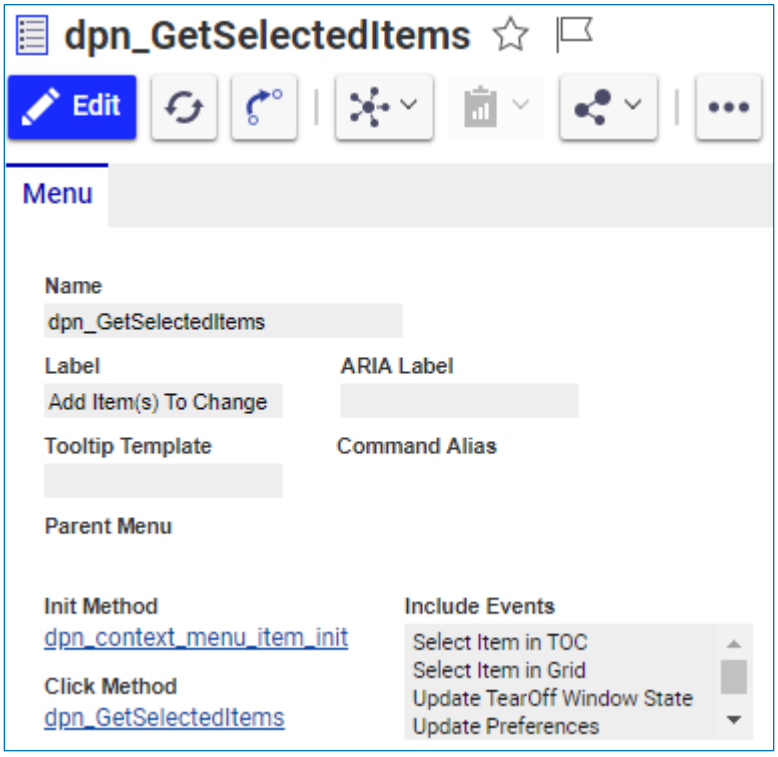


Figure 84.

6. In the **Command Bar Section** Relationships Grid, set the properties of the new row as follows:
    - **Action**: **Add**
    - **Identify**: **World**

9.2.3 TGV Context Menu Customization Results

The TGV Context Menu has the new **Add Item(s) To Change** command.

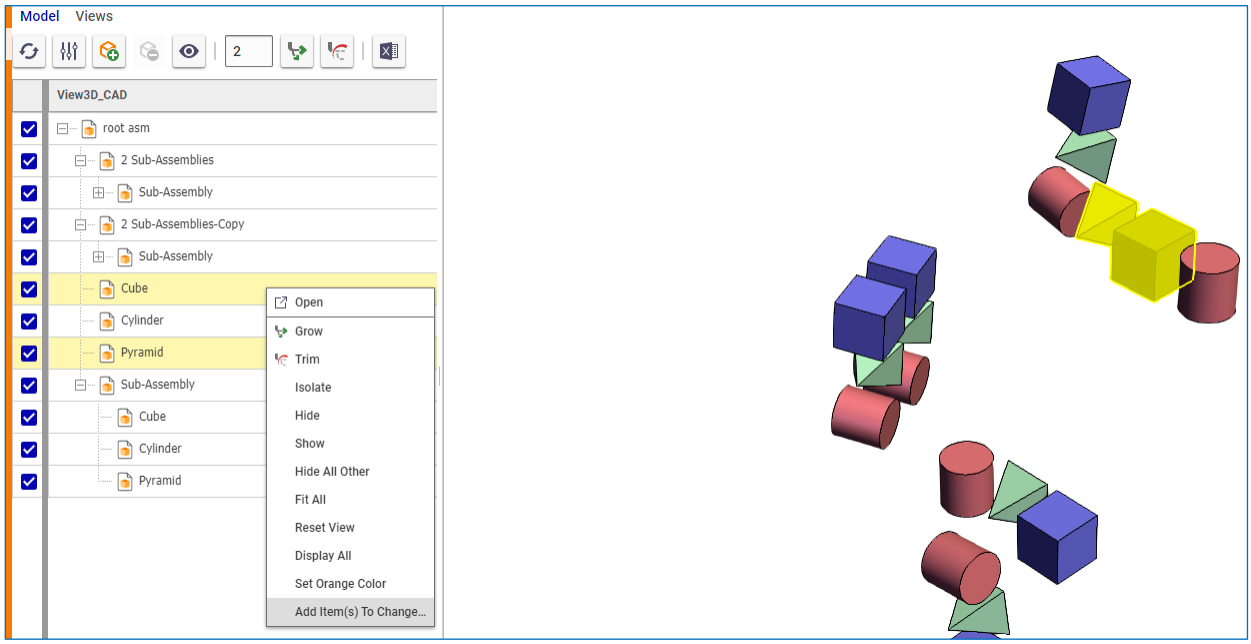


Figure 85.

Clicking this command calls the **Choose Change Item** dialog for selected nodes.

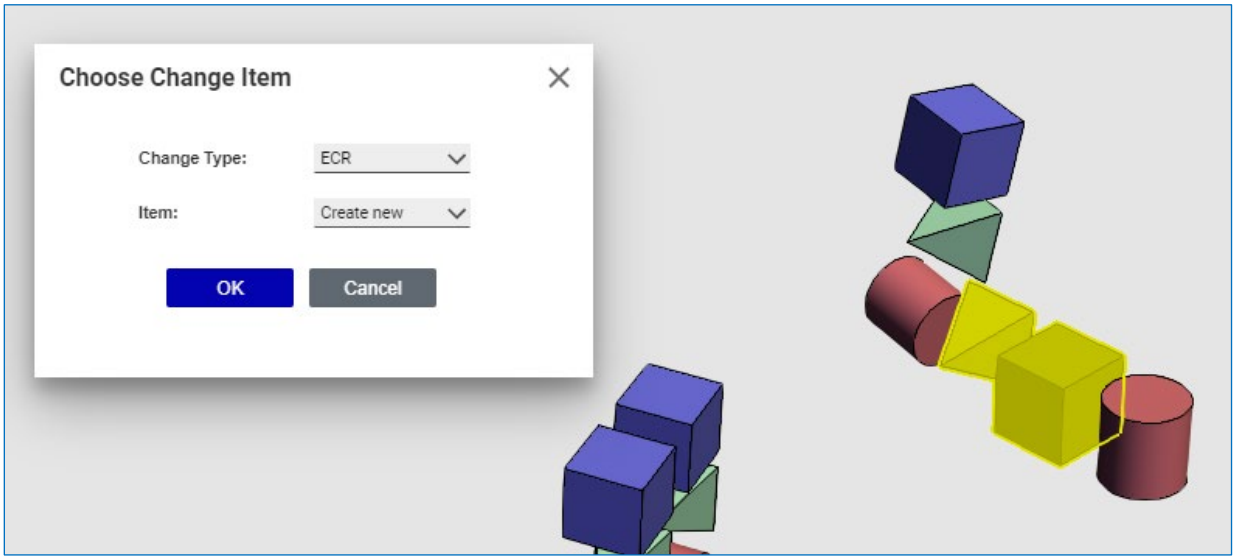


Figure 86.

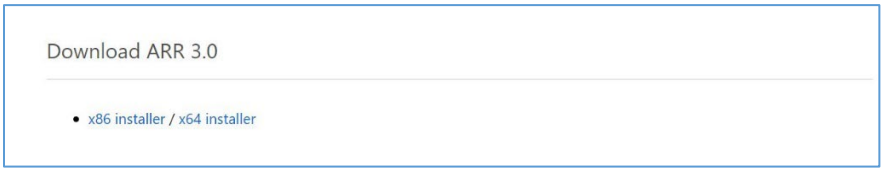
9.3 Reverse Proxy Server

This section is for information purposes only.

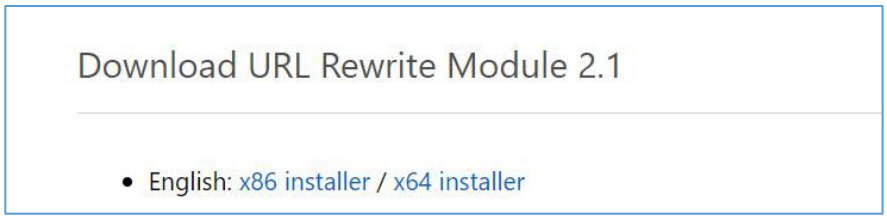
For Streaming Viewer, the Client-side 3D HOOPS Viewer will need to communicate with the Server-side Services using networking ports that are not typically open in customer firewalls. Reverse Proxies allow network traffic to use standard HTTP connections (80, 443) for such information flow.

The following steps outlines the process of installing and configuring the Reverse Proxy for Streaming 3D Viewer:

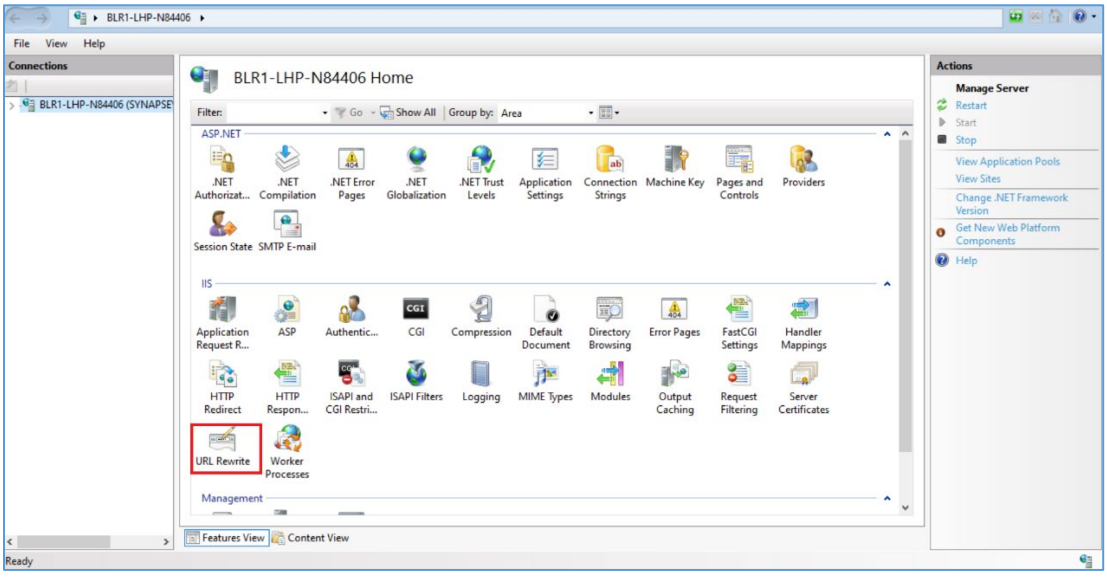
1. Install **Application Request Routing** (ARR) 3.0 from the following link: https://www.iis.net/downloads/microsoft/application-request-routing.



2. Install **URL Rewrite 2.1** from the following link: https://www.iis.net/downloads/microsoft/url-rewrite



3. After installation is complete, open **IIS** from **Windows** browser. A new module appears in IIS Manager.



4. On IIS, click **Application Request Routing Cache**.

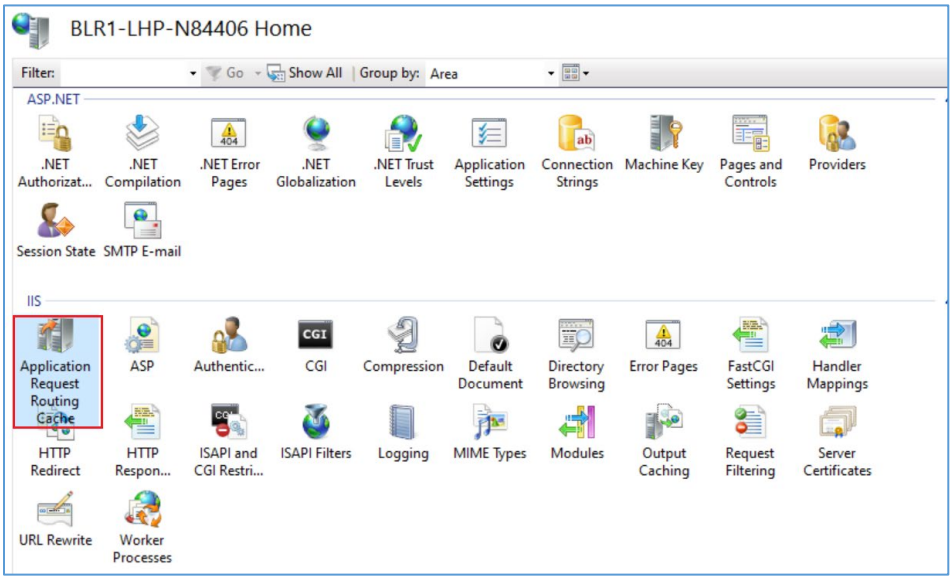


Figure 87.

5. From the Actions column, click **Server Proxy Settings**.

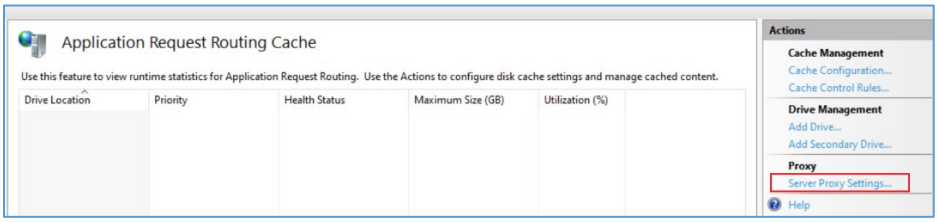


Figure 88.

6. Select **Enable Proxy** checkbox.

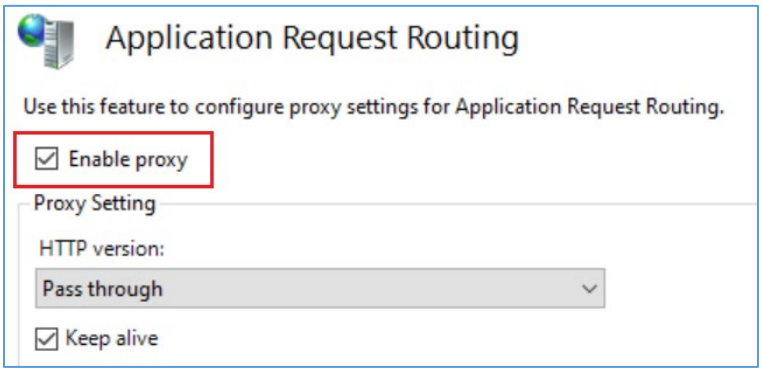


Figure 89.

7. Click **Apply**.

Next step is to Configure URL Rewrites rules. This can be done in two ways:

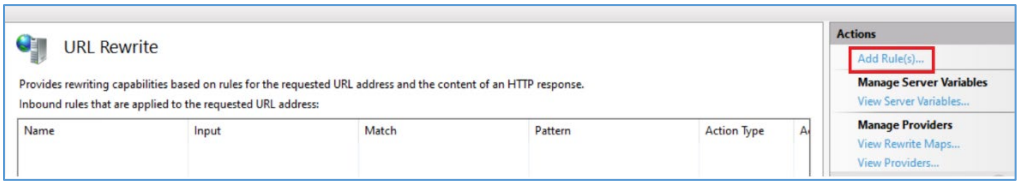
a) Option 1: Configure URL Rewrite rules using **IIS**
b) Option 2: Configure URL Rewrite rules using **web.config** file

Option 1: Configure URL Rewrite rules using IIS:

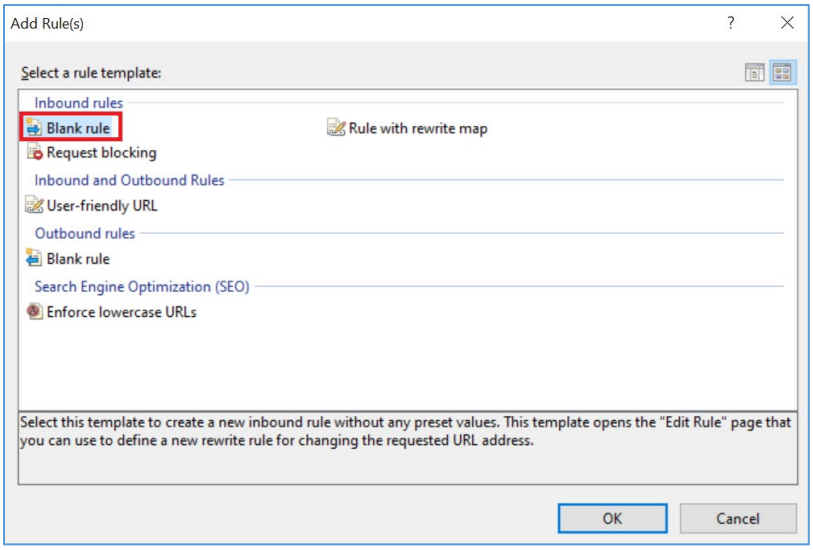
The following two rules must be set up in URL Rewrite:

- Entry point with link to Hoops Server
- Web Socket connections

8. From **Actions** column, click **Add Rules**.



9. Click **Blank Rule**.



10. Add the following details to the rule:
    - Name: Hoops_server
    - Pattern: hoops_server/?(.*)
    - Rewrite URL: http://localhost:{R:1}

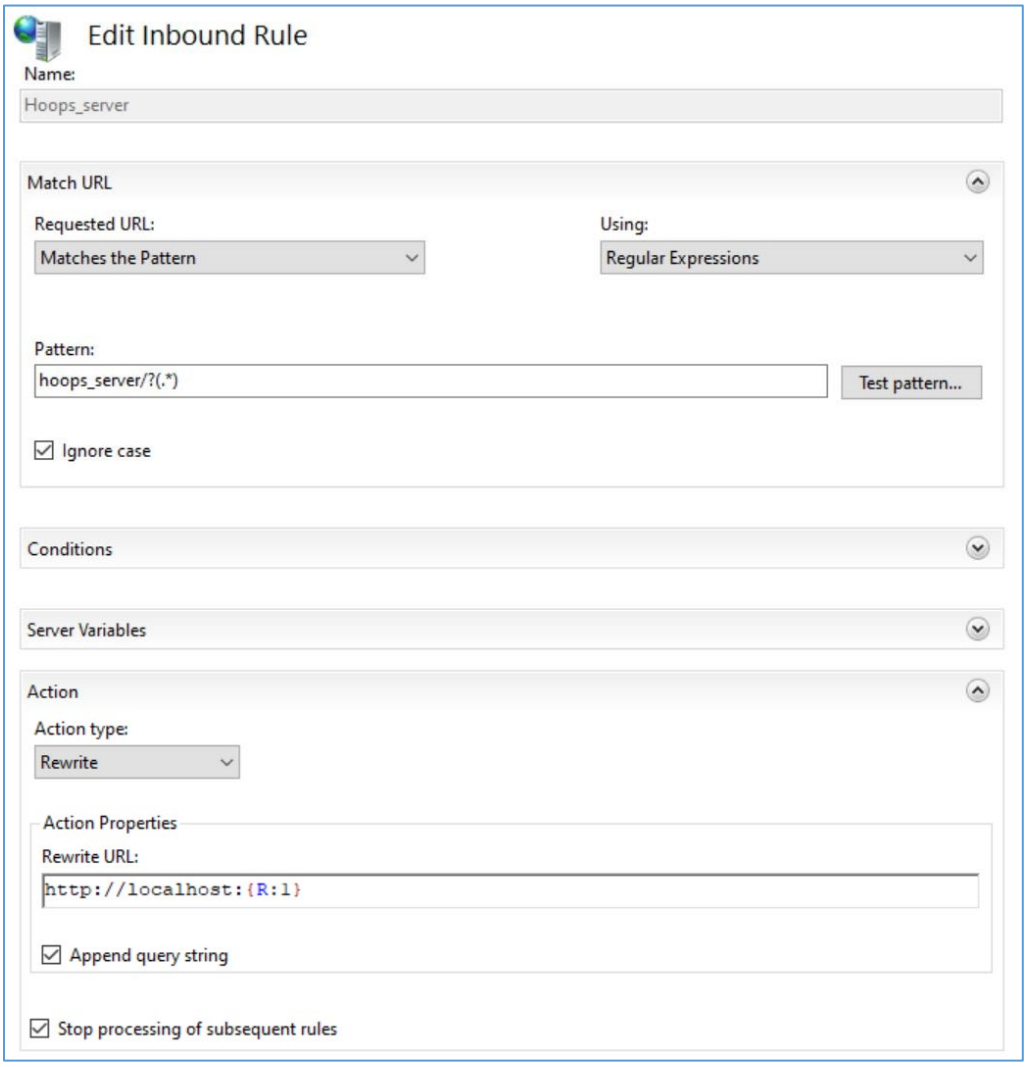


Figure 90.

11. Click **Apply**.

12. Add another Blank Rule with the following details:
    - Name: Web Socket Reverse
    - Pattern: ws(.*)hoops_server/?(.*)
    - Rewrite URL: `ws://localhost:{R:2}`

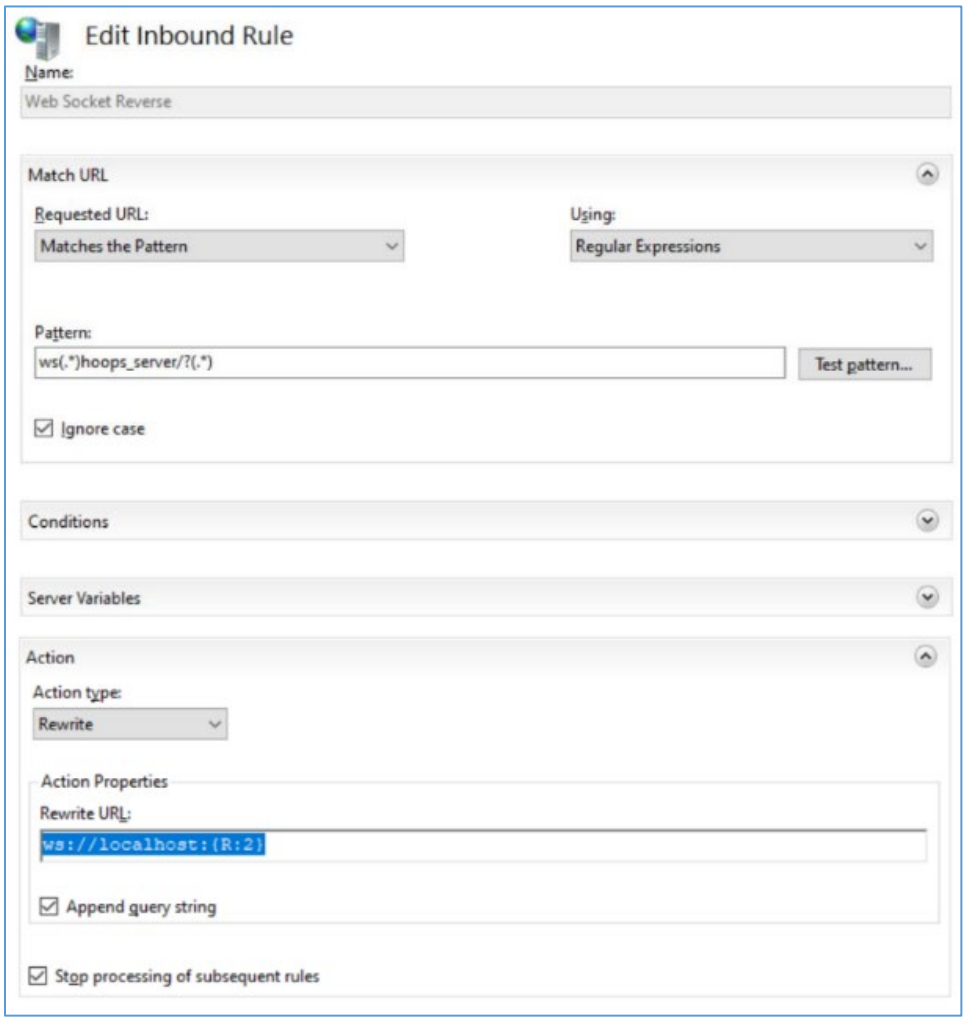


Figure 91.

Option 2: Configure URL Rewrite rules using web.config file:

13. Open **web.config** file from Aras Innovator code tree: "**C:\Aras\InnovatorServer\Innovator\web.config file"**.
14. In the `<system.webserver>` tag, add the following code as shown in the screenshot below:

```
<system.webServer>
    <rewrite>
        <rules>
            <rule name="Web Socket Reverse" enabled="true" stopProcessing="true">
                <match url="ws(.*)hoops_server/?(.*)" />
                <action type="Rewrite" url="ws://localhost:{R:2}" />
            </rule>
            <rule name="hoops_server" enabled="true" stopProcessing="true">
                <match url="hoops_server/?(.*)" />
                <action type="Rewrite" url="http://localhost:{R:1}" />
            </rule>
        </rules>
    </rewrite>
</system.webServer>
```

Figure 92.

- Hoops_server in match url can be different and this will be accounted as "Hoops Server Url".
- Action url should point to Instance of the Hoops Server.

Next step is to adjust **HOOPS_ServerUrl**.

15. In the Innovator Instance, from the **Table of Contents**, expand **Administration** and select **Variables**.
16. Click **Search Variables**.

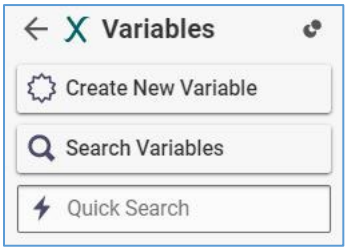


Figure 93.

17. Click **Search** and select **HOOPS_ServerUrl**. The HOOPS_ServerURL form appears.

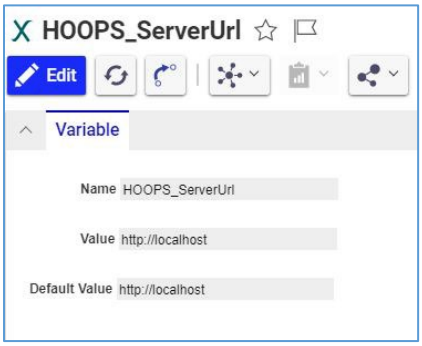


Figure 94.

Change HOOPS_ServerUrl to point to Hoops Server through configured proxy.

18. Click **Edit**.
19. In the **Value** field, add `http://localhost/InnovatorServer/hoops_server/11182`

    where,

    - Localhost – IP Address of the machine
    - InnovatorServer- Innovator instance of the user
    - 11182- Hoops Server port number

Figure 95.

20. Click **Done** to save the changes.
21. Go to "**HOOPS Server\server\node\Config.js"**.
22. For `publicHostname` parameter, make the following change:

    `publicHostname: "localhost/Your innovator instance/hoops_server",`

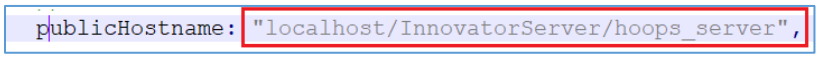


Figure 96.

23. Go to "**HOOPS Server\server\node\lib\SpawnerExtension.js**".
24. For `getEndpoint(req, res)` method,

    Change:
    "endpoint":`${Utils_1.getWsProtocol(this.extConfig.sslEnableScServer)}://${hostname}:${this.extConfig.spawnServerPort}`,

    To:

    "endpoint":`${Utils_1.getWsProtocol(this.extConfig.sslEnableScServer)}://${hostname}/${this.extConfig.spawnServerPort}/`,

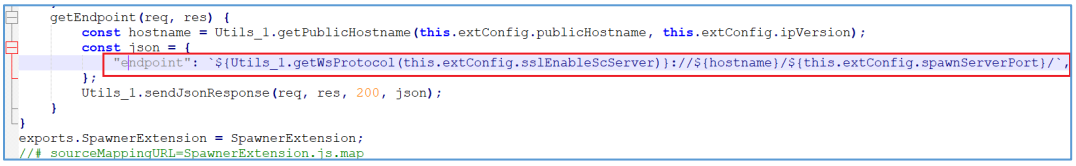


Figure 97.