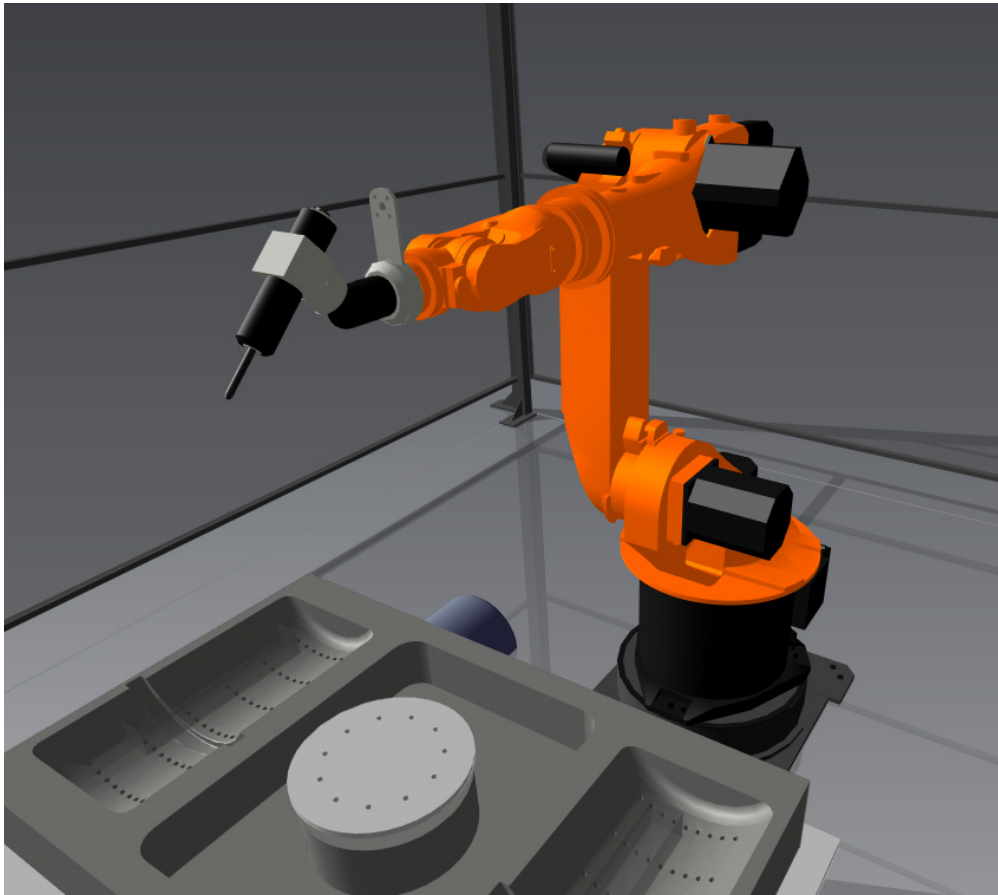


# Quick Start for Visual Components

Essentials | Version: February 12, 2016 | Example: Files Available Upon Request



**Support**  
[support@visualcomponents.com](mailto:support@visualcomponents.com)

**Community**  
[community.visualcomponents.net](http://community.visualcomponents.net)

# Contents

Summary	3	Module 4 - Model a Component	50
Requirements	3	Structure of components	51
Installation and Setup	4	To create a root node and feature	55
Interaction	4	To transform a feature	56
Getting Started	5	To save a component	58
Security	6	To create and connect behaviors	59
Collections	6	To create locations for behaviors	61
Preferences	7	To create new nodes	64
Workspace	8	To assign a joint	66
Module 1 - Layouts and Simulations	9	To define a joint	67
To open a layout	10	To detect components and signal events	68
To navigate the 3D world	11	To physically connect components	71
To select a component	12	Review	73
To run and document a simulation	13	Module 5 - Modeling with Python	74
To edit the properties of a component	16	Python in Visual Components	75
To interact with a component	17	To trigger specific actions in a simulation	76
Review	18	To turn off behaviors and move objects	79
Module 2 - Create and Save a Layout	19	To grab and move objects	80
To create a new empty layout	20	To clean up your code	82
To add components to the 3D world	21	Project	83
To move components	22	Review	85
To rotate components	23		
To connect components	24		
Inheritance	25		
To create components during a simulation	26		
To save a layout	27		
To create a drawing	28		
Review	33		
Module 3 - Create a Robot Program	34		
Concepts related to robot programming	35		
To open and select a robot program	38		
To interact with joints in a robot	41		
To edit base and tool frames	44		
To create a sequence of statements in a routine	46		
Review	49		

# Summary

Visual Components is a provider of easy-to-use 3D simulation products. Our products have applied uses for sales and marketing, engineering, manufacturing, robotics, education and computer-aided design. Essentials, our next generation product, has key tools and standard instructions to make learning easy and consistent for new and existing users.

In this guide you learn how to:

- Build layouts for factory planning and industrial design.
- Run and record simulations for multi-faceted business, engineering and marketing solutions.
- Perform data collection and analytics for simulation processes.
- Create robot programs.
- Model 3D components.

The main outcome of this guide is to provide an opportunity for 3D simulation and modeling.

Key results for you include:

- Skill set in automation and 3D simulation.
- Expertise and competency with industry-standard, progressive simulation tools.
- Practical experience in 3D modeling, drafting and testing.

# Requirements

Before you get started verify you have:

## **A working computer with at least Windows 7 system requirements.**

This includes a standard graphics card with at least 2/4 Gigabytes of dedicated memory and a CPU similar to an Intel Core i5 or i7 processor.

## **An active Internet or network connection.**

Essentials can be used offline at any time with a valid license after completing installation and setup. Remote libraries and files would need to be downloaded locally for offline use.

## **A premium license for Essentials.**

This allows you to complete sections requiring our modeling and robotics toolkits.

## **Administrator rights to install software.**

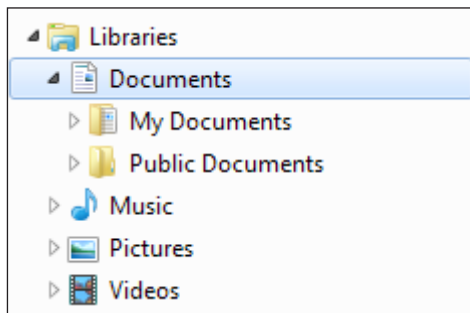
You must have user permissions to install software on your computer.

# Installation and Setup

Essentials is a 64-bit software that requires a license/product key.

1. Download the Essentials installer from the Visual Components website.
2. Run the installer, and then complete the steps in the Wizard.
3. Launch or run Essentials.
4. When prompted, choose a licensing option, and then follow the prompts to activate a standalone license or connect to a local license server and borrow an available license.

You have now completed installation and setup.



In your Documents library there are two folders for Visual Components. One folder is **public** and allows you to share your documents with other users. The other folder is **personal** and the default folder for containing documents that you create and use with Essentials.

## Path to public Visual Components folder

C:\Users\Public\Documents\Visual Components

## Path to personal Visual Components folder

C:\Users\[Your User Name]\Documents\Visual Components

## Interaction

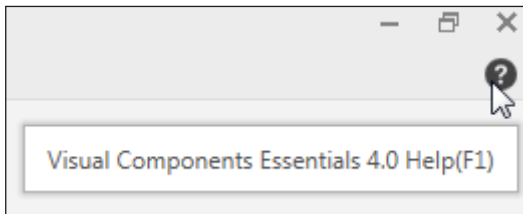
Your main source of interaction and input with Essentials is a mouse and keyboard.

You may use touch surfaces, track pads and 3D mouse devices. If you are using a 3D mouse, ensure any needed driver for that device is already installed on your computer. This guide assumes you are using a 2D mouse.

# Getting Started

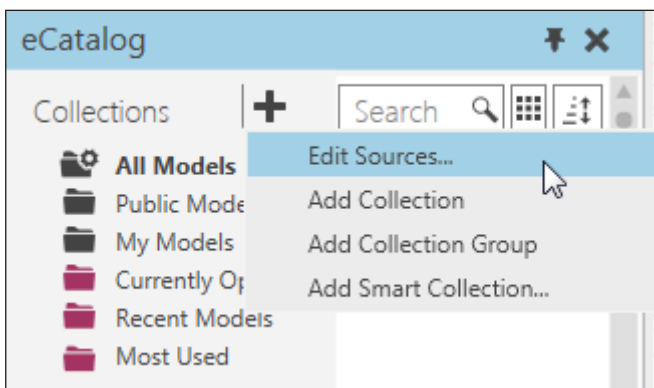
We recommend performing the following steps to help you complete the modules in this guide.

1. On the top-right corner of Essentials main window, click the **question mark**.

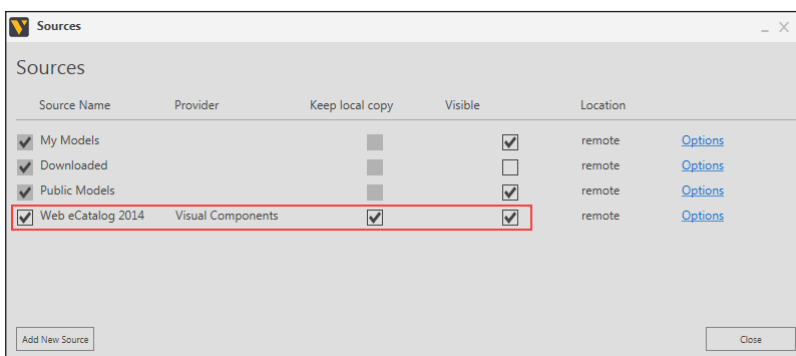


This opens the **Help File** of Essentials. Refer to this document when you need help or more information on certain items and workflows. You may also point at a command or option in the UI to show a tooltip.

2. In the eCatalog panel, Collections view, click the **plus sign**, and then click **Edit Sources**.



3. In Sources, find a source named **Web eCatalog 2014**, and then select its **Keep local copy** check box, and then click **Yes** to download a local copy of that remote library. Next, select its **Visible** check box, and then click **Close**.



This downloads the **Visual Components Web eCatalog**, a remote library of components, layouts and other supported files. The downloaded files are stored in the Local Copies folder in your Documents library. After the download is complete, you can effectively work offline.

## Path to Local Copies folder

C:\Users\[Your User Name]\Documents\Visual Components\2014\Local Copies

## Security

The eCatalog panel uses HTTP protocol and metadata to link, index, sort, filter, and display sources of files. Each source is either a *filepath* to a local folder or a *URL* to a remote repository or resource. For example, the URL of the Visual Components Web eCatalog is a web address to an XML file. Basically, the eCatalog panel acts like a web browser and serves as a file manager.

In some cases, a firewall or bandwidth restriction may prevent the eCatalog panel from accessing source files. If that happens and you are unable to use and/or download the Visual Components Web eCatalog, try the following:

**Contact your network or system administrator to fix/allow the connection.**

This is sometimes seen as an unwanted risk.

**Allow Essentials to communicate through the Windows Firewall of your computer.**

This might not be possible based on your user permissions.

**Connect to your Home network, and then download local copy.**

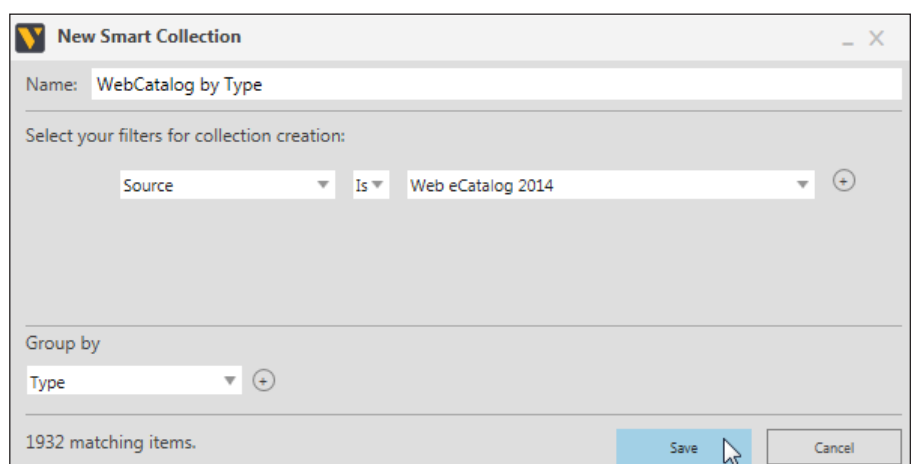
Generally, this is the easiest solution.

## Collections

Metadata allows you to create simple and smart collections that contain links/bookmarks to source files.

1. In the eCatalog panel, Collections view, click the **plus sign**, and then click **Add Smart Collection**.
2. In New Smart Collection, do all of the following:
  - In the Name box, type **WebCatalog by Type**.
  - Set the first query to **Source Is Web eCatalog 2014**.
  - Set the first Group by filter to **Type**, and then click **Save**.

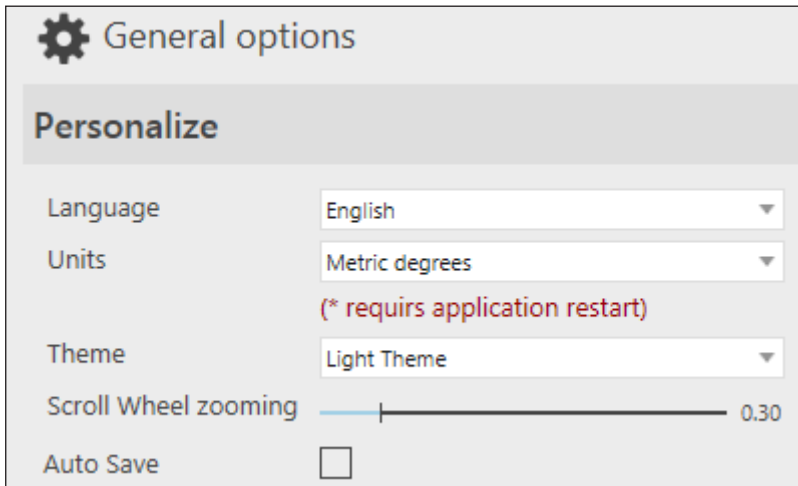
This creates a new smart collection in the eCatalog panel for finding items based on their type.



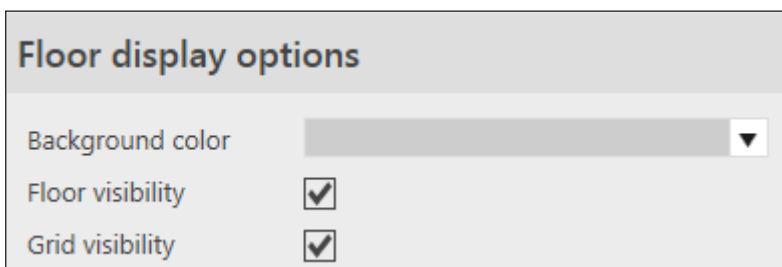
# Preferences

There are some settings in Essentials you may want to customize to match your preferences.

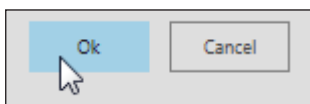
1. Click the **File** tab to go backstage, and then on the Navigation pane, click **Options**.
2. In General options, adjust the language, system of measurements and theme settings to your preference. This guide uses English, Metric degrees and a Light Theme.



3. Click **Display**, and then in Floor display options, verify Floor and Grid visibility are enabled.



4. At the bottom-right corner of Essentials main window, click **OK** to save all changes. This may require restarting Essentials.

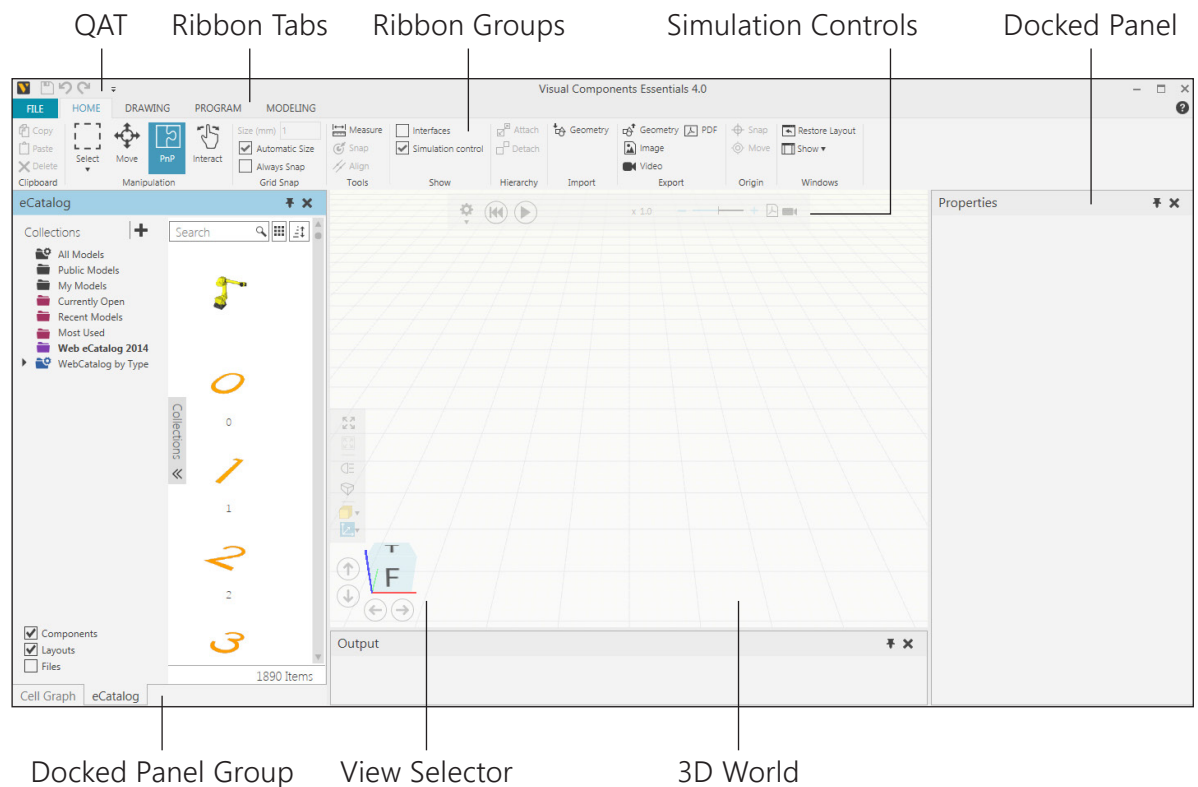


# Workspace

The workspace of Essentials has a set of tabs containing groups of related commands listed in a ribbon. Docking panels border a viewport for the 3D world, which is the environment used for working with components and running simulations.

The context of Essentials affects its workspace and the availability of commands and options. For example, clicking the Home tab displays a view for building layouts using components, and clicking the Modeling tab displays a view for modeling components. In some cases, commands become available based on your selection in the 3D world. In other cases, clicking a command displays a task pane with a set of options for executing the command.

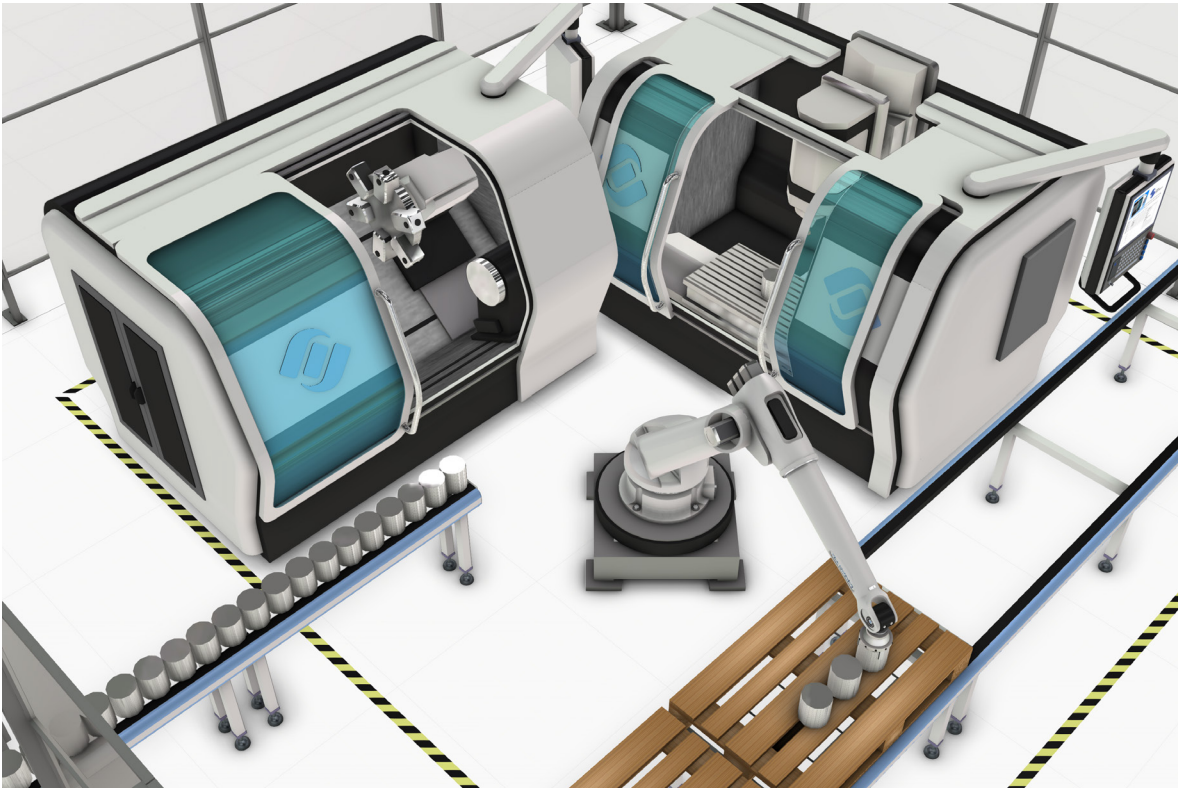
You can rearrange the workspace by docking and pinning panels, collapsing the ribbon, and adding and enabling commands in the Quick Access Toolbar or QAT, which is always either below or above the ribbon.





# Module 1 - Layouts and Simulations

In this module you learn about layouts, components, the 3D world and simulations.



*Exported image of a layout during a simulation*

A **layout** is a Visual Components file that is opened in the 3D world. Every layout contains at least one component and your settings. For example, edits you make to the size, color and location of a component in the 3D world are saved with a layout.

A **component** is an object that occupies space in the 3D world. The properties of a component are listed in the Properties panel.

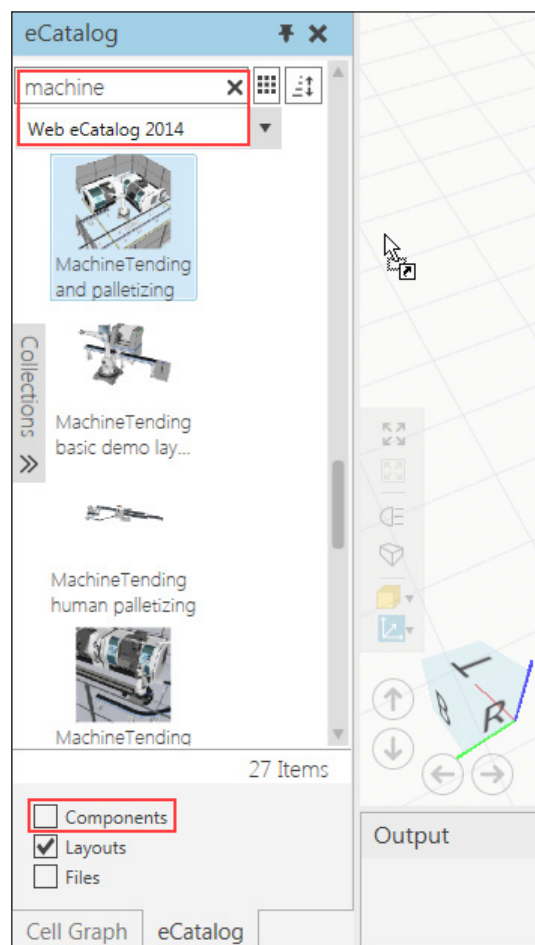
A **selection** is one or more components selected in the 3D world. Components in a selection can be moved, rotated, interacted with, and plugged into or remotely connected to other components. A selection can also be cut, copied, pasted and deleted.

A **simulation** is contained in the 3D world and provides a safe environment for testing and documenting results. You can control the speed, time, start and stoppage, and playback of a simulation.

## To open a layout

You can perform a drag-and-drop operation to open a layout in the 3D world.

1. In the eCatalog panel, Collections view, click **Web eCatalog 2014**, and then clear the **Components** check box.
2. Click the **Collections** vertical expander to collapse the Collections view.
3. In the Search box, type **machine**.
4. In the Display area, drag the **Machine Tending and palletizing** item into the 3D world.



A layout contains **links** or references to components and other supported files that are loaded when you open a layout. These files are known as **items** and they come from local folders and remote URIs that are known as **sources**. Links to items in sources are contained in **collections** that are listed in the eCatalog panel.

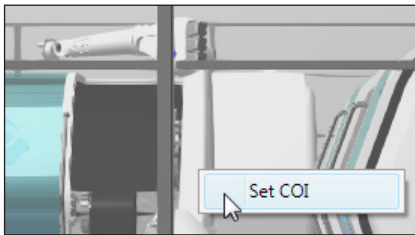
## To navigate the 3D world

Your view of the 3D world is controlled by a camera that you can move using mouse actions when the pointer is inside the viewport.

1. Point to the 3D world, and then do all of the following:
  - Press and hold the right mouse button (RMB), and then drag the pointer to rotate the camera.
  - Press and hold the left and right mouse buttons (LMB+RMB), and then drag the pointer to pan the camera.
  - Rotate the mouse wheel or press and hold SHIFT and the right mouse button (SHIFT+RMB), and then drag the pointer to zoom the camera.

The camera can be centered on any part of any component.

2. Either right-click a component, and then click **SetCOI** or hold down CTRL, and then right-click a component.



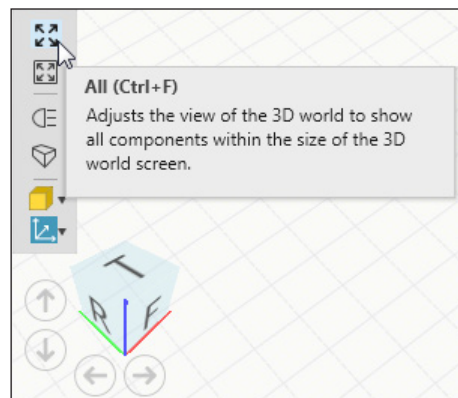
The **View Selector** at the bottom-left corner of the 3D world shows the camera point of view and positive direction of global coordinate axes (X-axis is red, Y-axis is green, Z-axis is blue).

3. On the View Selector, do all of the following:
  - Click the horizontal arrows to rotate the camera left or right, thereby using standard front, back, left and right views.
  - Click the vertical arrows to rotate the camera up or down, thereby using standard top and down views.

The toolbar directly above the View Selector provides a list of options related to the 3D world and scene, for example the display of frames and render modes.

4. On the 3D world toolbar, click **View All** to display all components within the size of the viewport.

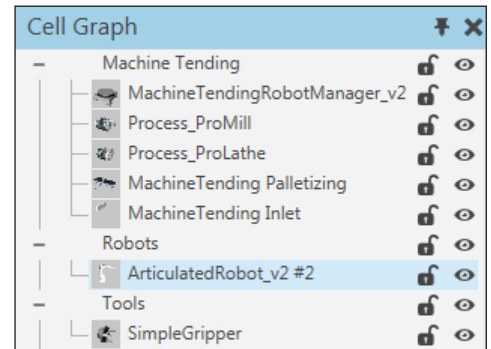
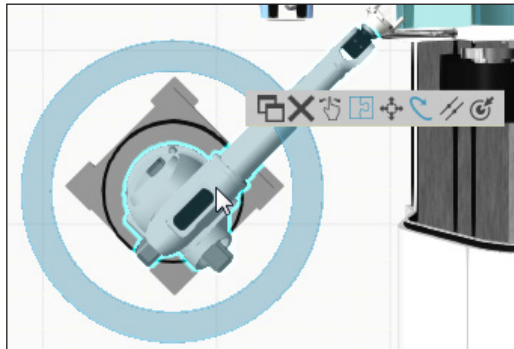
Unless you are given a specific instruction in this guide, you can edit your view of the 3D world at any moment.



## To select a component

A selection in the 3D world is highlighted blue and indicated in the Cell Graph panel, which provides an outline of all components in the 3D world. When you directly select a component, the 3D world displays an interactive ring to allow you to rotate the selection in place and a Mini toolbar containing quick commands.

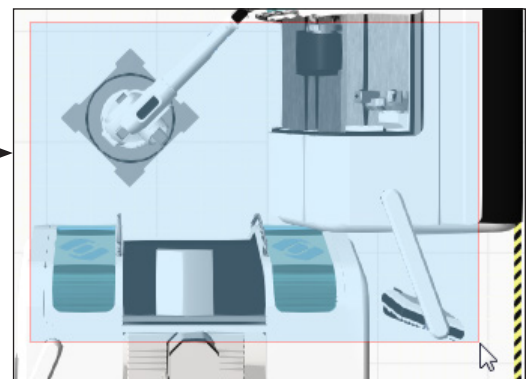
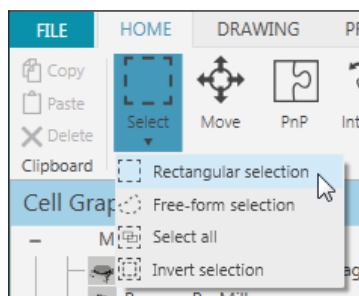
1. In the 3D world, click a component.



2. Press and hold CTRL, and then click another component to add that component to the current selection.
3. Press and hold CTRL, and then click a selected component to remove that component from the current selection.
4. Point to empty space in the 3D world to clear the selection.

You can use area selection to quickly select multiple components.

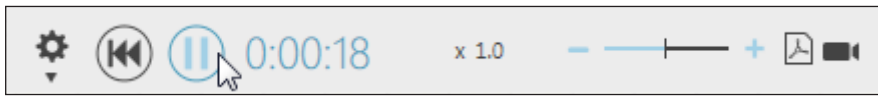
5. On the Home tab, in the Manipulation group, click the **Select** arrow, and then click **Rectangular selection**.
6. In the 3D world, drag the pointer to draw a selection area around components.



**TIP!** The Cell Graph panel can be used to select components that are difficult to see or invisible in the 3D world.

## To run and document a simulation

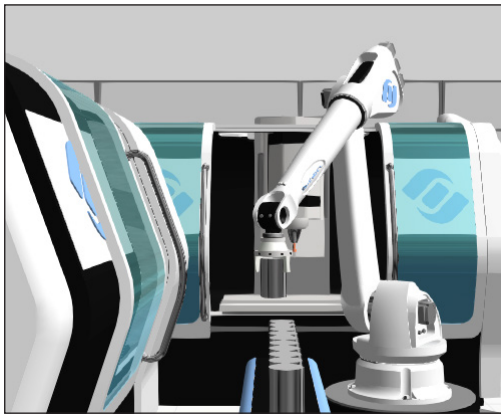
Simulation controls are located at the top-middle section of the 3D world.



### Running a simulation

You can start and stop a simulation at any moment in the 3D world.

1. On the Simulation controls, click **Run** to start and stop the simulation.



2. Click **Reset**.



When a simulation is reset **static components** return to their initial state at the start of simulation. That is why **dynamic components** created during a simulation are removed from the 3D world. For example, the joints of the robot in the layout reset to their initial values and the products and pallets disappear from the conveyors.

3. Run the simulation, and then do one of the following:
  - Drag the **Speed** slider left or right to speed up or slow down the simulation.
  - Click the **minus sign** or **plus sign** to step up or step down the speed of the simulation.

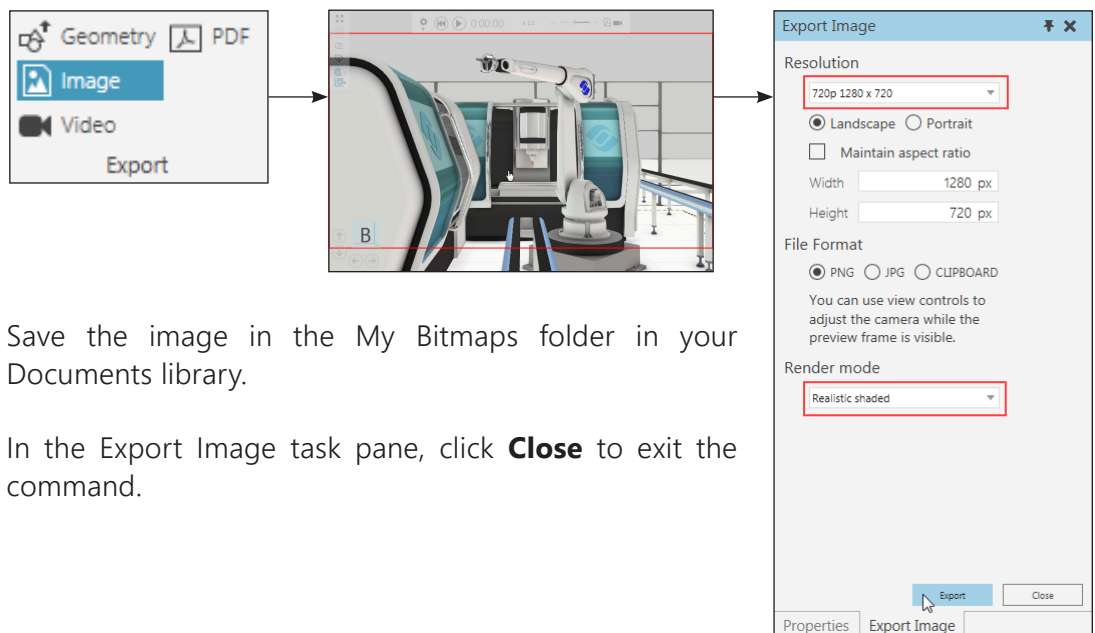
## Recording a simulation

You can document a simulation by printing and exporting images and recording 3D PDF and video files.



*A still image of a 3D PDF from a recorded simulation*

1. Stop the simulation.
2. On the Home tab, in the Export group, click **Image**, and then adjust your view of the 3D world according to the red boundary to frame the image.
3. In the Export Image task pane, set Resolution to **720p** and Render mode to **Realistic shaded**, and then click **Export**.



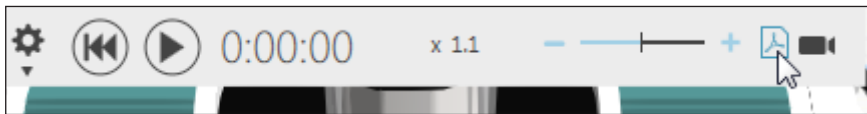
4. Save the image in the My Bitmaps folder in your Documents library.
5. In the Export Image task pane, click **Close** to exit the command.

### Path to My Bitmaps folder

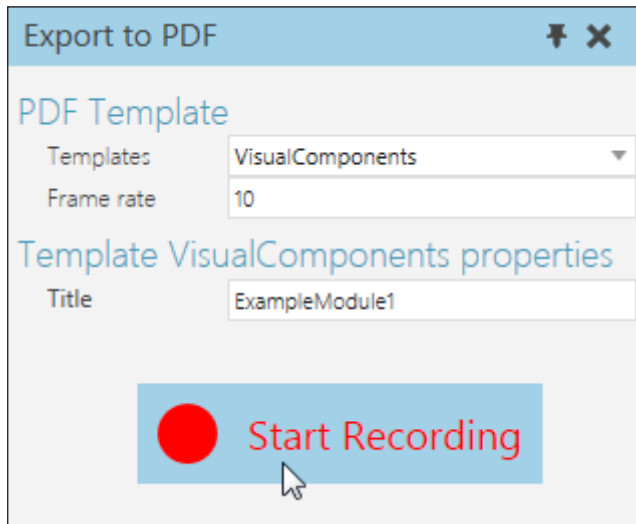
C:\Users\[Your User Name]\Documents\Visual Components\2014\My Bitmaps

Recording a 3D PDF or video can be executed at the start of or during a simulation. If the simulation does not have a defined runtime, you will need to manually stop recording.

6. Reset the simulation, and then on the Simulation controls, click **PDF**.



7. In the Export to PDF task pane, define a title and then click **Start Recording**.



8. Save the file in the My Videos folder in your Documents library.
9. In the Export to PDF task pane, click **Stop and Save** to end recording and generate the file, and then click **Close** to exit the command.

### Path to My Videos folder

C:\Users\[Your User Name]\Documents\Visual Components\2014\My Videos

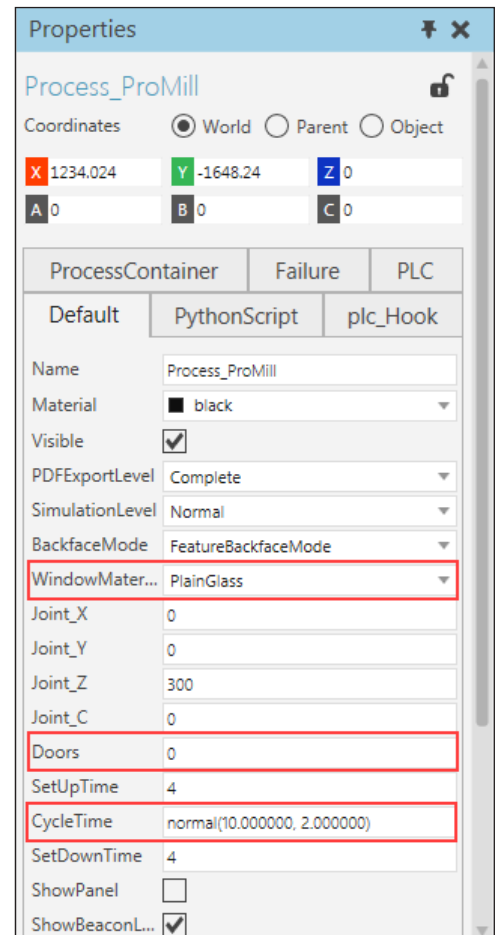
If you want, record a simulation as a video, but know that a video records any changes you make to the 3D world view.

## To edit the properties of a component

The properties of a selected component that you can edit are listed in the Properties panel. When multiple components are selected, common properties are listed and can be edited if they do not require unique values. For example, two static components cannot have the same name.

1. In the 3D world, select **Process\_ProMill**.
2. In the Properties panel, Default tab, do all of the following:
  - In the Doors box, type **0**.
  - Set WindowMaterial to **PlainGlass**.
  - In the CycleTime box, type **normal(10.0,2.0)** and then press ENTER.
  - Clear the **ShowPanel** check box.

Certain types of properties allow you to write expressions. When an expression is evaluated, the returned value is assigned to a property. To write an expression, refer to the Expressions Reference in the Help File.



*Set properties of a selected component*

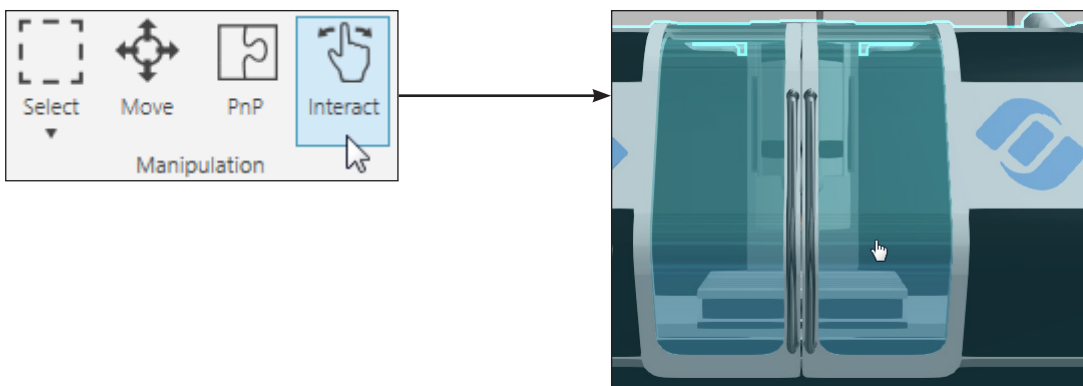


## To interact with a component

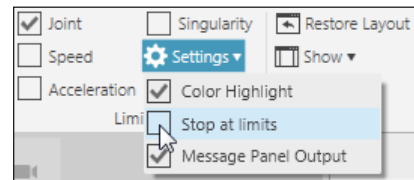
A component can have interactive parts, for example joints that can be moved and rotated in the 3D world. Generally, an interactive part has a limit and degrees of freedom (DOF) to constrain and define its range of motions.

Visual feedback for interactive parts is when the pointer changes to a hand icon.

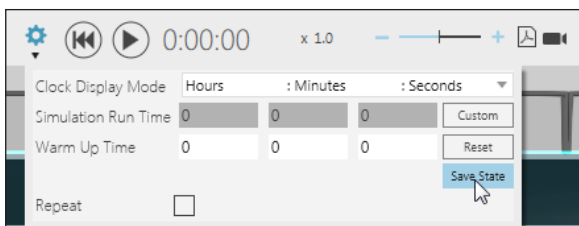
1. On the Simulation controls, click **Reset**.
2. On the Home tab, in the Manipulation group, click **Interact**, and then point to the doors of the selected component.
3. When the pointer changes to a hand icon, press and hold the left mouse button, and then drag the doors until they are closed and touching.



The doors will most likely be highlighted red to indicate a joint limit error. If you have access to the Program tab, you can enable the Stop at limits setting in the Limits group. Otherwise, you can ignore the error for now.



4. On the Simulation controls, click the **Setting** arrow, and then click **Save State** to save the current state of component's joints, and then click **Reset**.



**IMPORTANT!** The initial state of a component is automatically saved when you start a simulation.

5. Drag the doors of the selected component open.
6. On the Simulation controls, click **Run**, and then reset the simulation to verify the doors stay open.

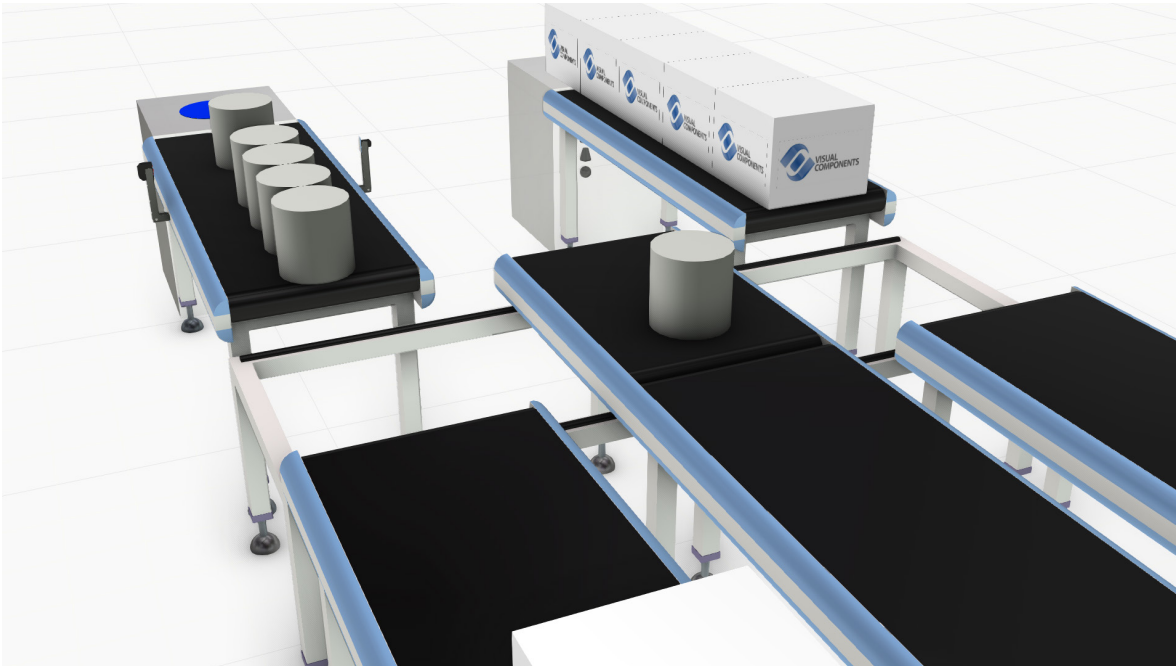
## Review

In this module you learned how to:

- Browse the eCatalog panel and open a supported file in the 3D world.
- Navigate and edit your view of the 3D world.
- Select components and modify a selection.
- Run and record simulations.
- Edit the properties of components listed in the Properties panel.
- Interact with and edit the state of components.

# Module 2 - Create and Save a Layout

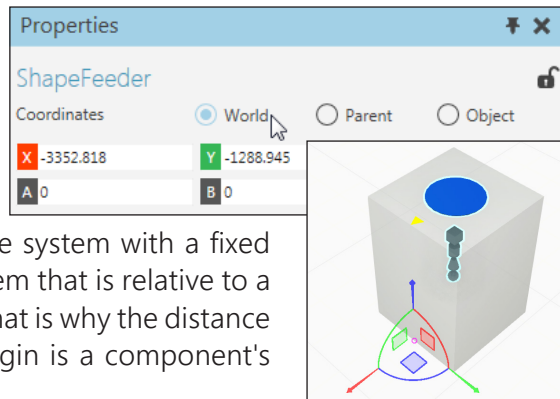
In this module you learn how to create a layout by adding, moving and connecting components in the 3D world.



Conveyor line of connected components simulating material flow

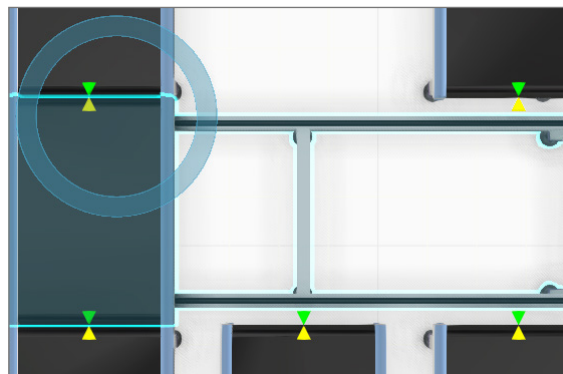
A **coordinate system** is used to reference the location and orientation of components in the 3D world. The origin of a coordinate system is a point or distance of zero on the XYZ axes.

Two of four available coordinate systems are World and Object. **World** is a global coordinate system with a fixed origin. **Object** is a component's coordinate system that is relative to a component's current position in the 3D world. That is why the distance of a component's origin from the 3D world origin is a component's location in the World coordinate system.



An **interface** supports Plug and Play and is a port that allows you to connect components in the 3D world. A **physical interface** is located at a Frame feature and indicated by an arrow. That type of interface allows you to snap components together. An **abstract interface** allows you to remotely connect components to send and receive data, for example signals.

Visual feedback for interfaces can be yellow or green. **Yellow** indicates an available interface and **green** indicates a connected interface.



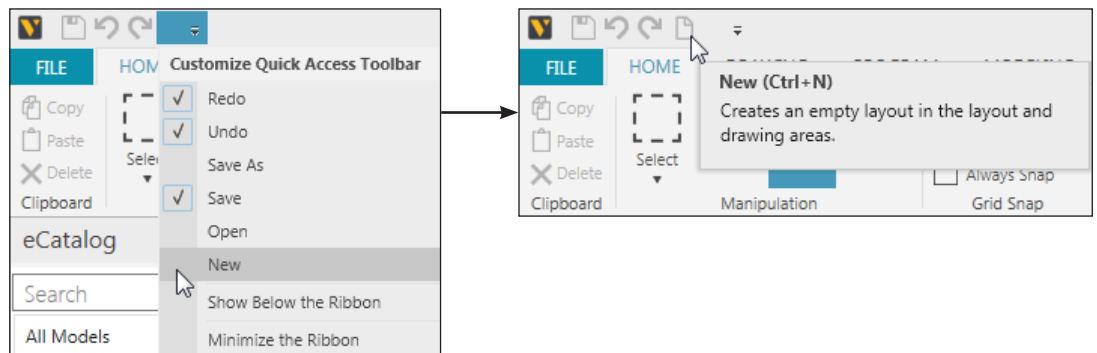
## To create a new empty layout

You can create a new layout to quickly clear the 3D world of components.

1. Either click the **File** tab, and then on the Navigation pane, click **New** or press CTRL+N.
2. In the prompt, click **Don't Save** to not save the current layout before opening a new layout.

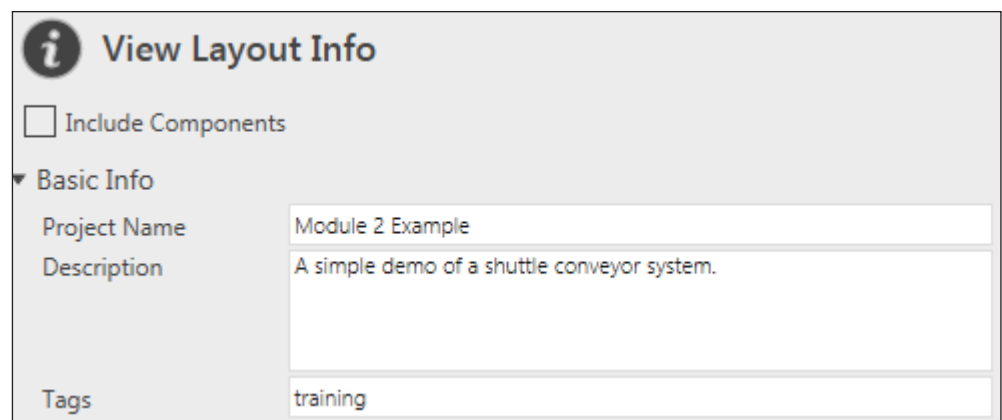
You may want to access File commands without changing the view of the workspace.

3. On the left corner of Essentials title bar, click the **Customize Quick Access Toolbar** arrow, and then click **New** to show the command on the QAT.



In some cases, you want to describe a layout or solution before you create it in the 3D world.

4. Click the **File** tab, and then on the Navigation pane, click **Info**.
5. In View Layout Info, do all of the following:
  - Set Project Name to **Module 2 Example**.
  - Set Description to **A simple demo of a shuttle conveyor system**.
  - Set Tags to **training**.



*Metadata for current layout in 3D world and saving options*

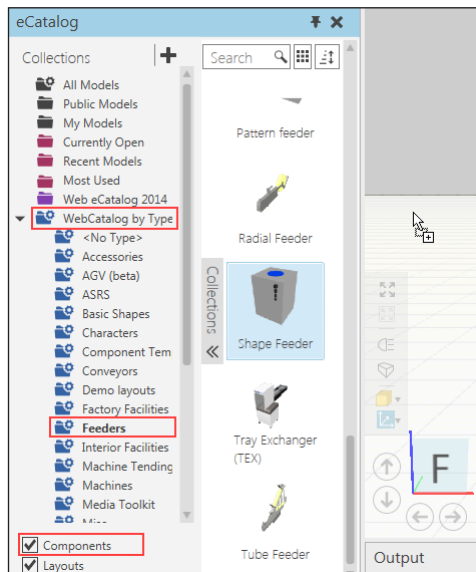
6. On the Navigation pane, click the **Go Back** arrow.

## To add components to the 3D world

Collections in the eCatalog panel can be used to quickly sort, filter and find components that you want to add to the 3D world.

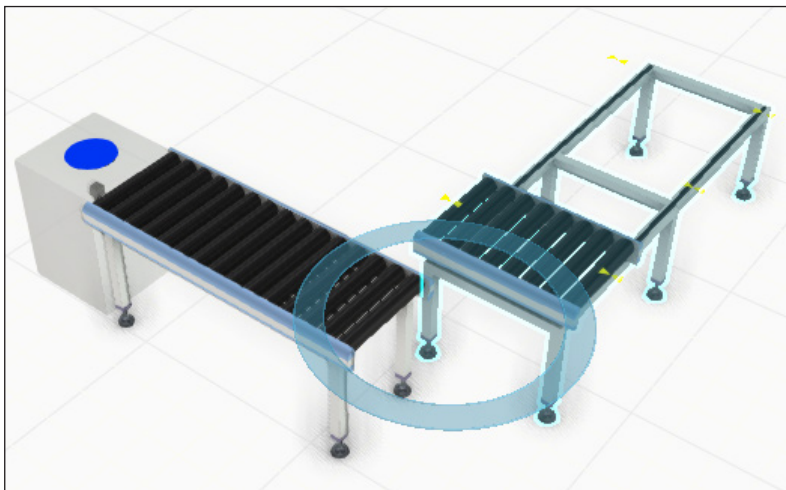
1. In the eCatalog panel, click the **Collections** vertical expander to show the Collections view.
2. Expand the **WebCatalog by Type** collection, and then click **Feeders**.
3. Select the **Components** check box to display items that are components.
4. In the Display area, drag a **Shape Feeder** item into the 3D world.

**TIP!** You can double-click an item to add it to the 3D world origin.



A component is automatically selected when it is added to the 3D world. The Plug and Play (PnP) command is activated to allow you to move and connect a selection to other components in the 3D world.

5. In WebCatalog by Type, click **Conveyors**.
6. In the Display area, double-click a **Conveyor** item, and then double-click a **Shuttle** item.

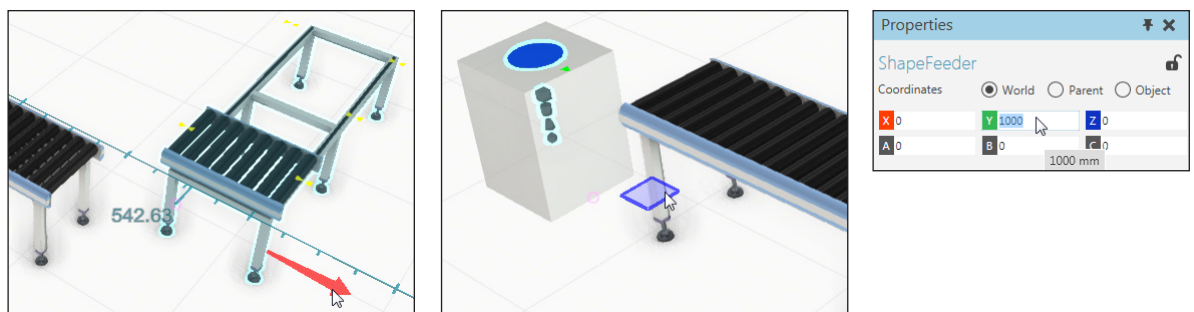


In some cases, an added component is automatically plugged into a selected component if they have compatible interfaces and can form a logical connection, which is what happened to the Shape Feeder and Conveyor. That is known as **Automatic Plug and Play**. In other cases, there may be too many solutions or no valid reason to automatically connect components, which is what happened to the Shuttle Conveyor.

## To move components

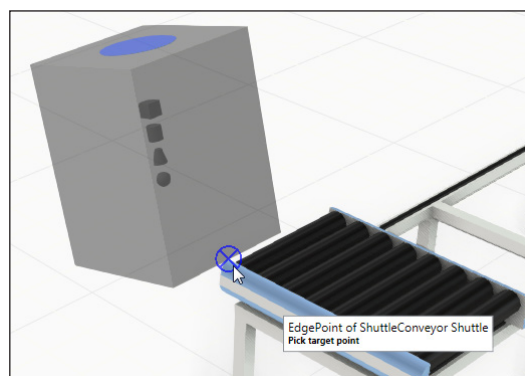
A selected component can be moved on a specific axis or plane and given specific XYZ coordinates.

1. On the Home tab, in the Manipulation group, click **Move**.
2. In the 3D world, click the **Shuttle Conveyor**, and then drag the **X-axis** arrow of the manipulator to move the Shuttle Conveyor away from the Conveyor.
3. Click the **Shape Feeder**, and then drag the **XY** plane of the manipulator to move the Shape Feeder away from the Conveyor.
4. In the Properties panel, click the **X-axis** button to reset its value to zero, and then in the **Y-axis** box, type **1000**.



In some cases, you may want to snap a component to a specific point or object in the 3D world. By default, you can preview the potential position and/or orientation of a selected component as well as undo and redo those actions.

5. On the Home tab, in the Tools group, click **Snap**.
6. Point to a edge or face on the Shuttle Conveyor to preview the result of the command, and then click a target to execute the command.



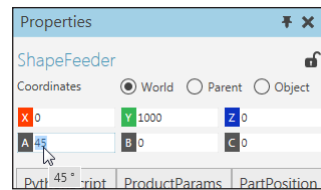
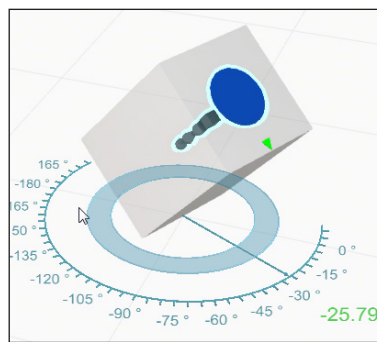
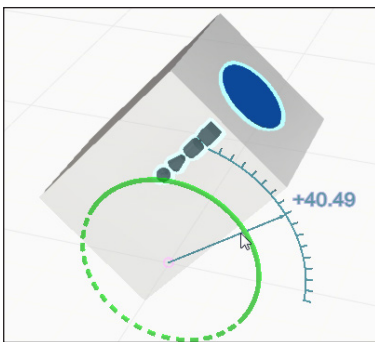
7. On the QAT, click **Undo** or press CTRL+Z to undo the new location of the Shape Feeder.

**TIP!** The torus of the manipulator is the origin of a selected component and can be dragged to move and snap a component based on the settings in the Grid Snap group.

## To rotate components

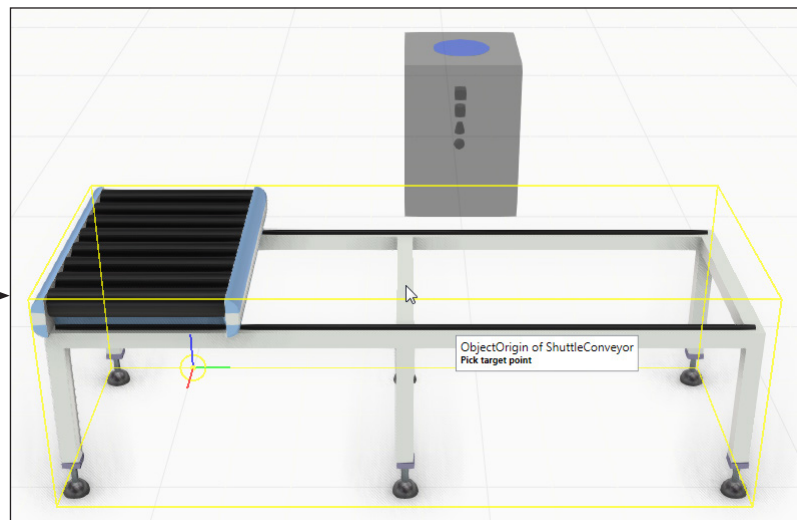
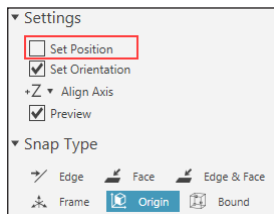
A selected component can be rotated around a specific axis, dragged around its Z-axis, and given specific degrees of rotation for the XYZ axes.

1. In the 3D world, drag the **Y-axis** ring of the manipulator to rotate the Shape Feeder toward the floor.
2. On the Home tab, in the Manipulation group, click **PnP**.
3. In the 3D world, drag the **XY** ring of the selection to rotate the Shape Feeder in place, that is around its Z-axis.
4. In the Properties panel, click the **B** button to reset the pitch value to zero, and then in the **A** box, type **45** to set the yaw value.



In some cases, you may want to snap a component to face a certain direction.

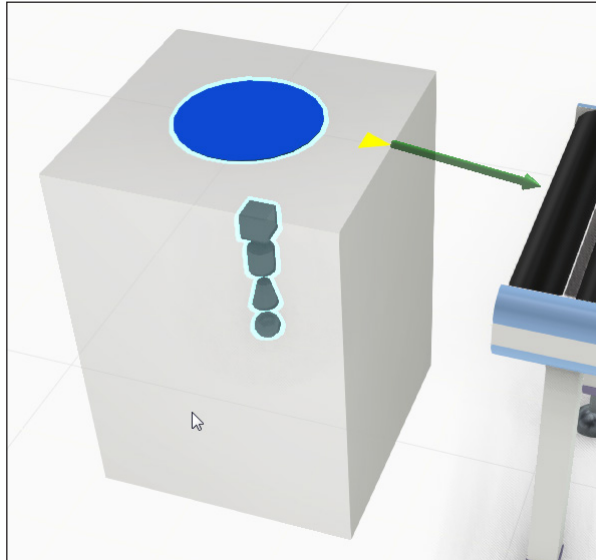
5. On the Home tab, in the Tools group, click **Snap**.
6. In the Component Snap task pane, under Settings, clear the **Set Position** check box, and then under Snap Type, click **Origin**.
7. Point to the Shuttle Conveyor and move the pointer left or right to orient the Shape Feeder to lay flat on the floor, and then click the target to execute the command, which should set the yaw value of the Shape Feeder to zero.



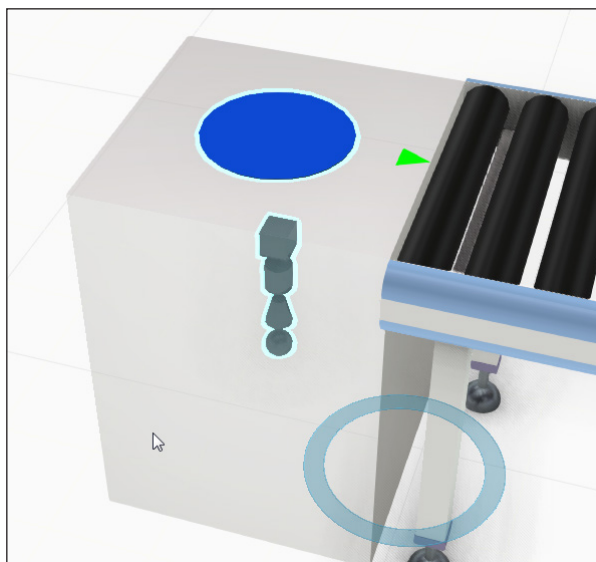
## To connect components

A component can be moved to another component to check for available physical connections. If an available connection exists, a green arrow will appear and point in the direction of the closest available connection.

1. In the 3D world, drag the **Shape Feeder** toward the Conveyor until a green arrow appears to show the direction to an available connection in the Conveyor.



2. Continue to drag the **Shape Feeder** in the direction of the green arrow until the Shape Feeder snaps to the Conveyor.



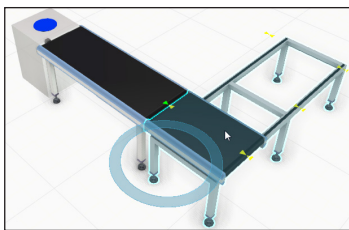
3. Click the **Shuttle Conveyor**, and then drag the **Shuttle Conveyor** to the other end of the Conveyor to connect those components.
4. Drag the **Shuttle Conveyor** away from the Conveyor until the arrow for the connected interface turns yellow to disconnect the components.



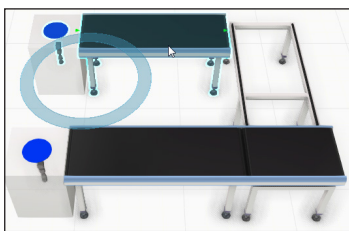
# Inheritance

Some components have similar properties whose values can be inherited when you establish connections between components. A blue arrow indicates a **parent-child relationship** between components in the 3D world.

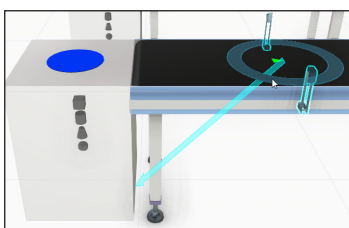
1. In the 3D world, select the **Conveyor**.
2. In the Properties panel, Default tab, do all of the following:
  - Set ConveyorType to **BeltConveyor**.
  - Verify the **AutomaticParameters** check box is selected.
3. In the 3D world, connect the **Shuttle Conveyor** to the Conveyor.



4. Select the **Shape Feeder** and **Conveyor**.
5. On the Home tab, in the Clipboard group, click **Copy**, and then click **Paste** to add a new set of components to the 3D world.
6. Connect the **selection of pasted components** to the other input connection of the Shuttle Conveyor as shown in the following image.



7. In the eCatalog panel, WebCatalog by Type collection, click **Processors**, and then add a **Conveyor Sensor** to the 3D world.
8. Drag the selected **Conveyor Sensor** to the path of the Conveyor until the Sensor snaps to the path of the Conveyor, thereby connecting and attaching the Sensor to the Conveyor.

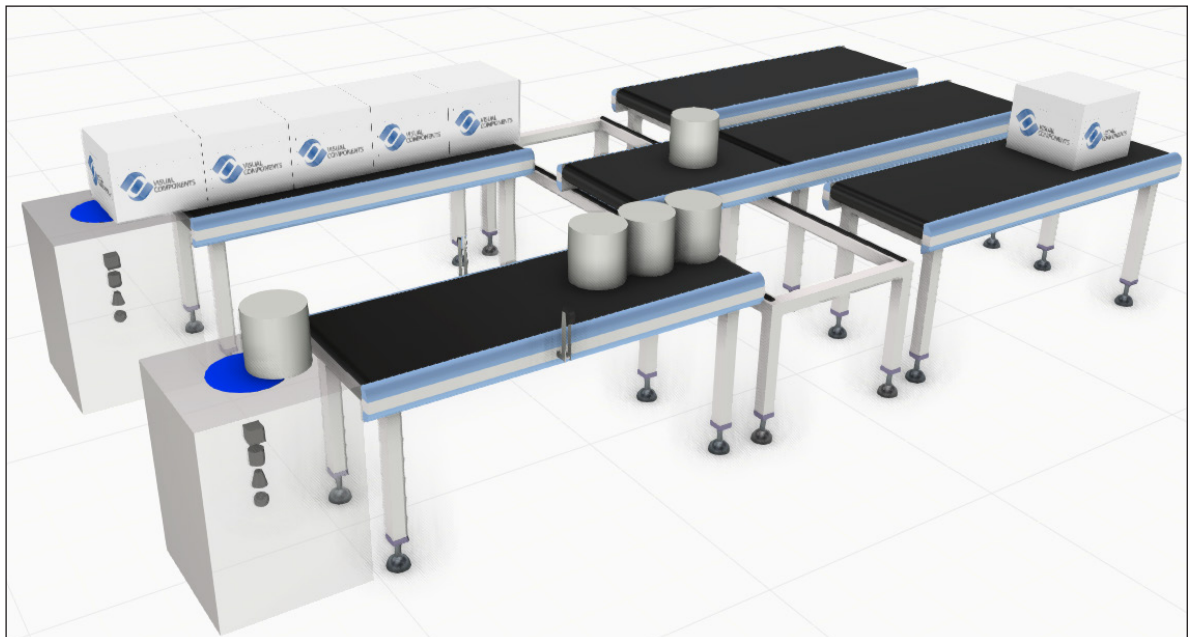


**TIP!** You can use the Hierarchy group to attach and detach components from one another in the 3D world.

## To create components during a simulation

**Feeders** are a type of component that allow you to create components during a simulation. A **template component** is the component to be created by a feeder. Some feeders provide you with a list of choices and properties for a template component. Other feeders require you to type the URL, VCID or name of a component or browse to the file path of a component.

1. In the 3D world, select the **Shape Feeder**.
2. In the Properties panel, Default tab, do all of the following:
  - Set Product to **Cylinder**.
  - In the CreationInterval box, type **normal(10.0, 2.0)** and then press ENTER.
3. In the Properties panel, ProductParams tab, set Material to **Aluminum**.
4. In the 3D world, select **Conveyor #2**, and then press CTRL+C.
5. Press CTRL+V three times to add three new conveyors in the 3D world.
6. Connect the **pasted conveyors** to the output connections of the Shuttle Conveyor.
7. Run the simulation to verify cylinders are being created by one Shape Feeder, and then reset the simulation.

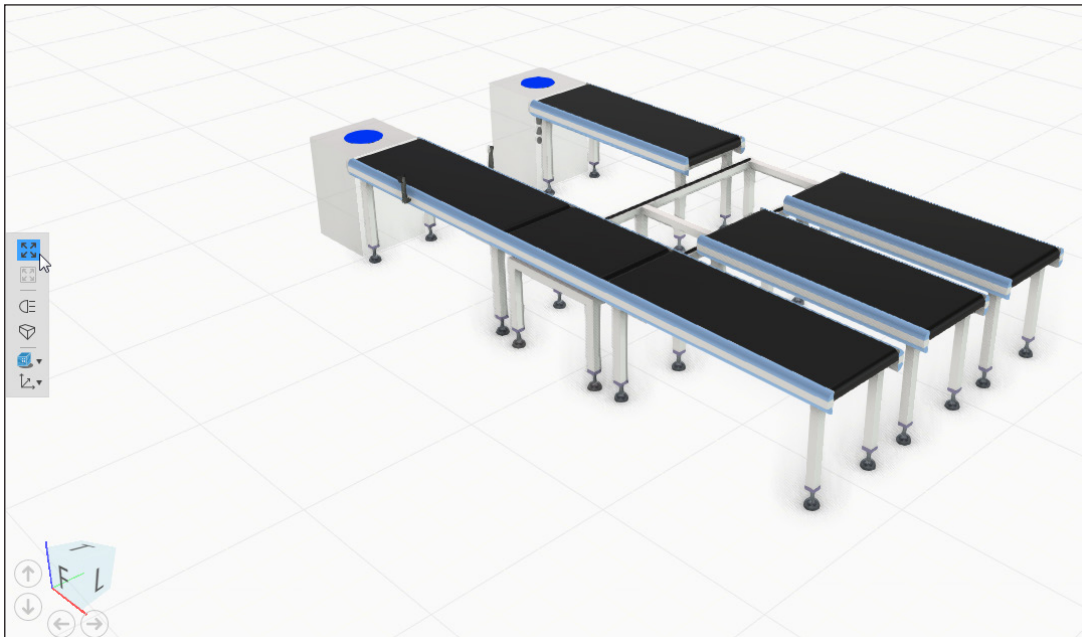


*Different components created during a simulation and moved in a flow*

## To save a layout

A layout is saved as a Visual Components file with a **vcmx** filename extension. Your configurations for components and the 3D world are saved in a layout. Any changes you make to a layout can be backed up to an existing file for that layout. Components can be packaged and saved with a layout. Otherwise, only links to components are saved with a layout.

1. In the 3D world, adjust your view to be isometric with the front, left and top views, and then on the 3D world toolbar, click **View All**.



2. On the QAT, click **Save** or press CTRL+S, and then save the file as **Module2.vcmx** in the My Models folder in your Documents library.

### Path to My Models folder

C:\Users\[Your User Name]\Documents\Visual Components\2014\My Models

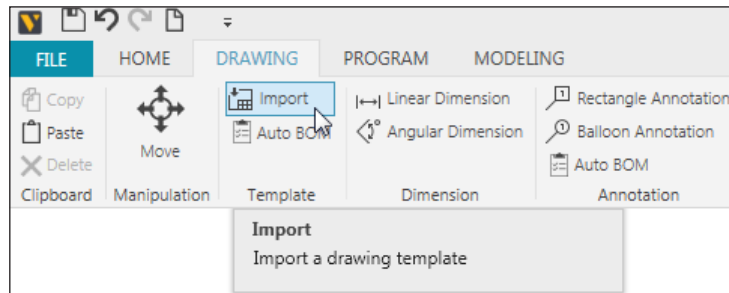
## To create a drawing

A **drawing** is a type of layout item that allows you to create scalable two-dimensional drawings of components in the 3D world. Drawings are contained on a 2D artboard called the **drawing world** which supports camera panning, zooming, filling and centering.

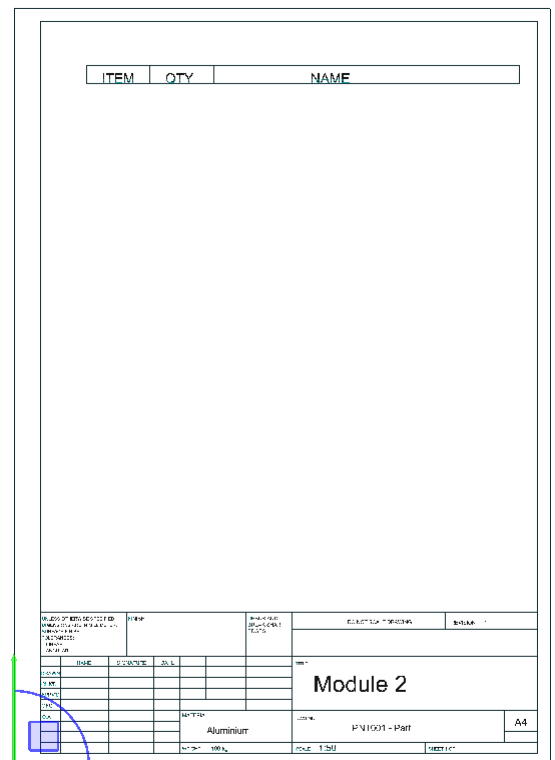
### Using a template

A template provides a print-ready sheet for adding and scaling new drawings and documenting

1. On the Drawing tab, in the Template group, click **Import**.



2. In the Template Import task pane, set Template to **Drawing Template A4**, and then click **Import**.
3. In the drawing world, click a point or line on the imported template to select the entire template.
4. In the Properties panel, Default tab, do the following:
  - In the Title box, type **Module 2** to update the name of the sheet.
  - In the Scale box, type **1:50** to define the scale size of any new drawing.

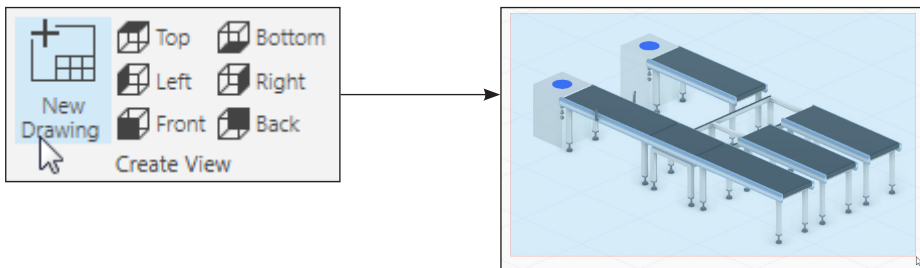


An imported drawing template

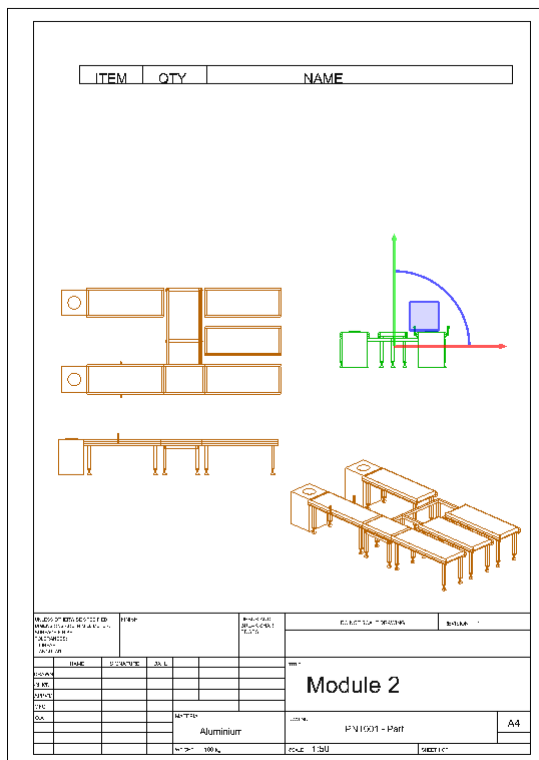
## Adding views

Every drawing is generated from a 3D world view and is an orthographic projection. You can manually set the view or use a standard view to automatically create new drawings.

1. On the Drawing tab, in the Create View group, click **New Drawing** to create your own view and drawing.
2. In the 3D world, drag the pointer to create a selection area encompassing all components in the current layout.



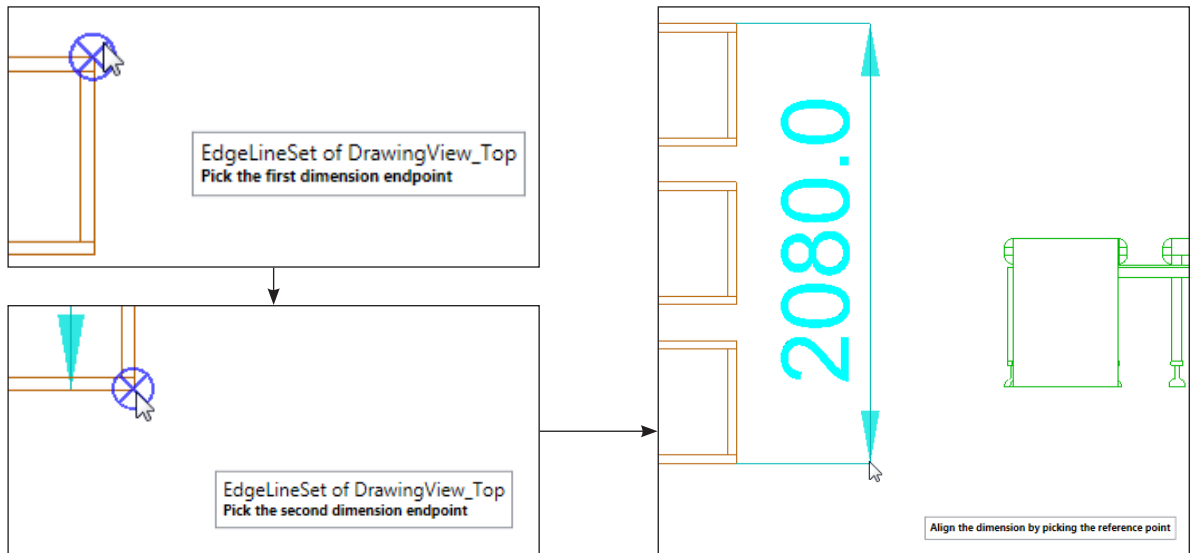
3. On the Drawing tab, in the Create View group, click **Top**, and then click **Front**, and then click **Right** to generate three drawings using standard views.
4. Position the four drawings to fit within the margins of the template in a logical order: Top, Right, Front and Isometric/Custom.



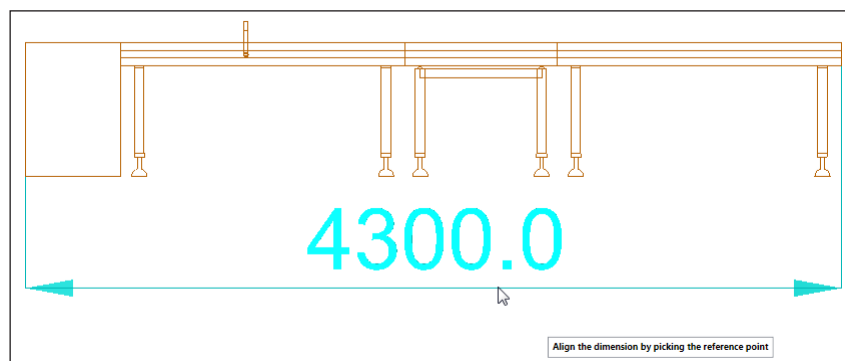
## Adding annotations and dimensions

Annotations and dimensions are other types of layout items that allow you to mark up drawings. An **annotation** is a text label anchored to a point, line or edge of a drawing. A **dimension** is a line with arrows representing the size or angle between the endpoints of the line.

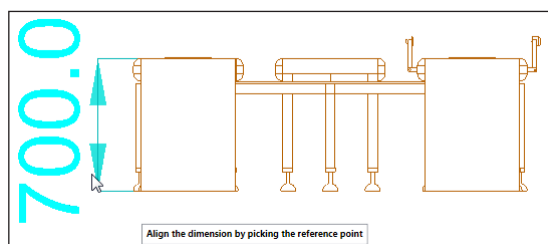
1. On the Drawing tab, in the Dimension group, click **Linear Dimension**.
2. Use the **top-side drawing** to add a dimension listing the width of the three output conveyors by selecting two endpoints and a reference point for placing the dimension.



3. Use the **front-side drawing** to add a dimension listing the total length of the conveyor line.



4. Use the **right-side drawing** to add a dimension listing the height of the conveyor line not including its attached sensor.



It is possible to auto-generate annotations for all or selected drawings in a template.

5. Click an empty space in the drawing world to exit the Linear Dimension command.
6. Select the **isometric drawing** you added to the template.
7. On the Drawing tab, in the Annotation group, click **Auto BOM** to auto-generate balloon annotations for the selected drawing and update the bill of materials table in the template.
8. Select and rearrange the added balloon annotations to fit within the margins of the template.

ITEM	QTY	NAME
1	1	Conveyor #2
2	1	Conveyor #5
3	1	Conveyor #4
4	1	Conveyor #3
5	1	ShuttleConveyor
6	1	Conveyor
7	1	ShapeFeeder
8	1	Sensor
9	1	ShapeFeeder #2

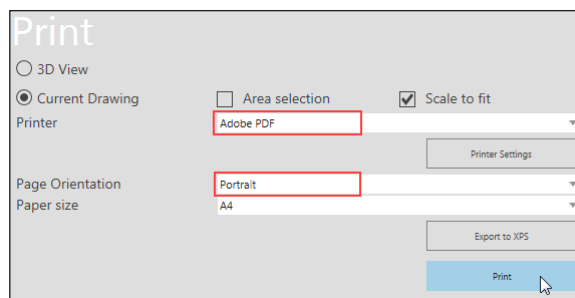
  

The image shows an isometric drawing of a conveyor system. It includes three orthographic views: a top view showing a total length of 4300.0, a side view showing a height of 2080.0, and a front view showing a height of 700.0. The isometric view is annotated with nine numbered callouts (1-9) that correspond to the items listed in the BOM table above. Callout 1 points to a sensor, 2 to a conveyor, 3 to a shuttle conveyor, 4 to a conveyor, 5 to a shape feeder, 6 to a conveyor, 7 to a shape feeder, 8 to a sensor, and 9 to a shape feeder.

## Printing and exporting a sheet

Layout items are automatically saved with the current layout of the 3D world. You can print and export drawings but cannot save them separate from their layout. Examples of drawings include floor plans, bill of materials and parts lists.

1. On the Drawing tab, in the Print group, click **Drawing**.
2. In the Print preview, do all of the following:
  - Set Printer to **Adobe PDF**.
  - Set Page Orientation to **Portrait**.
3. Click **Print**., and then save the file as **Module2Drawing.pdf** in the My Models folder in your Documents library.



4. On the Navigation pane, click the **Go Back** arrow, and then save the layout.



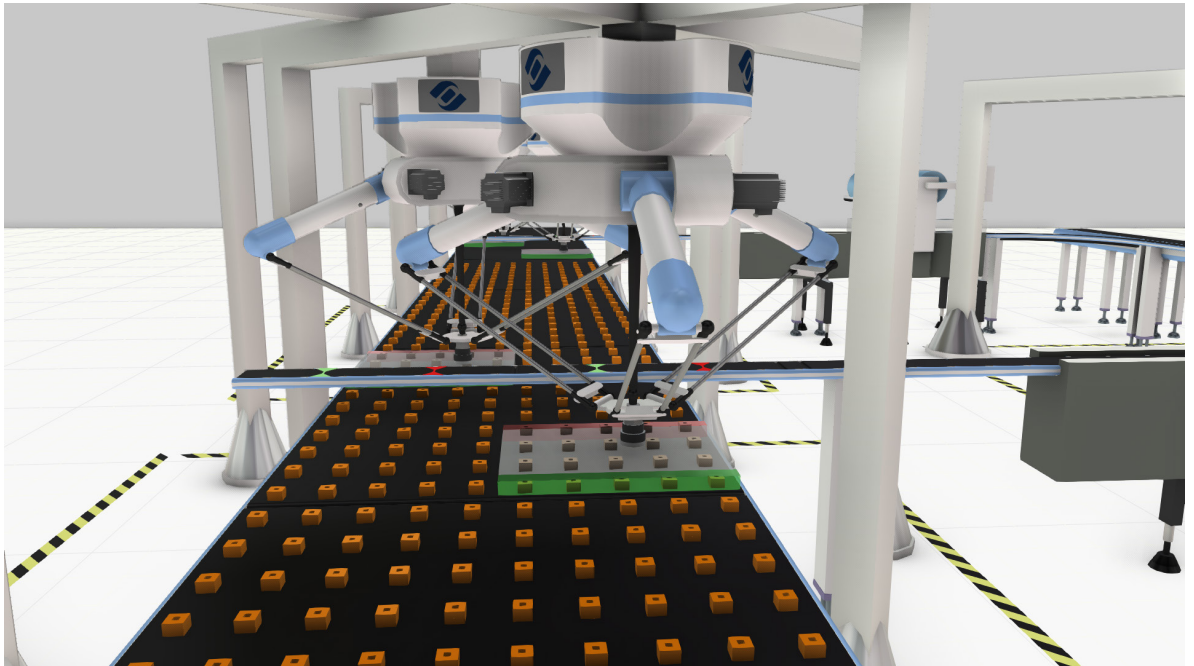
## Review

In this module you learned how to:

- Create, build and save a new layout.
- Add components from the eCatalog panel to the 3D world.
- Move and rotate components.
- Connect and attach components physically to one another.
- Create different types of components during a simulation.
- Create drawings, annotations and dimensions for a layout.
- Print a bill of materials for a layout.

## Module 3 - Create a Robot Program

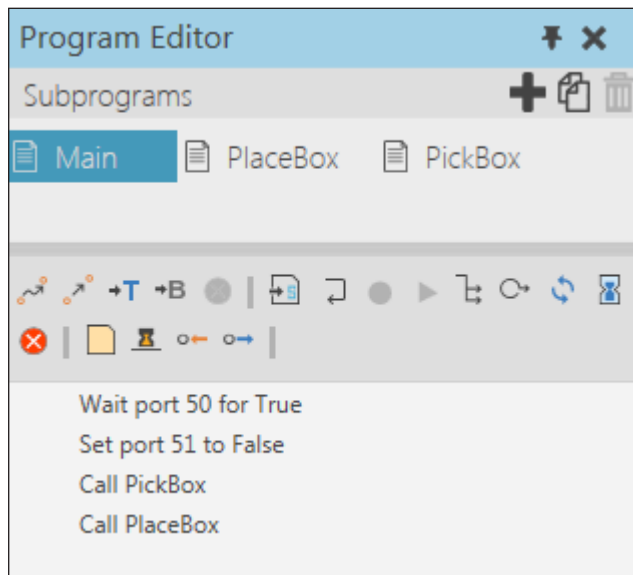
In this module you learn how to create robot programs and implement robot resources in a layout, which requires a premium license for Essentials.



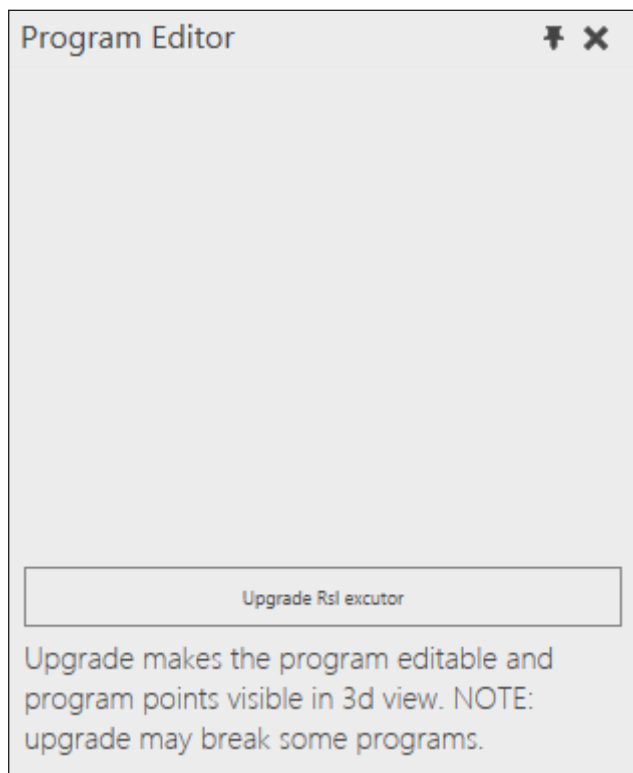
*Delta robots performing a pick and place operation*

# Concepts related to robot programming

The following is quick overview about robot programming in Essentials.



Robot program editor



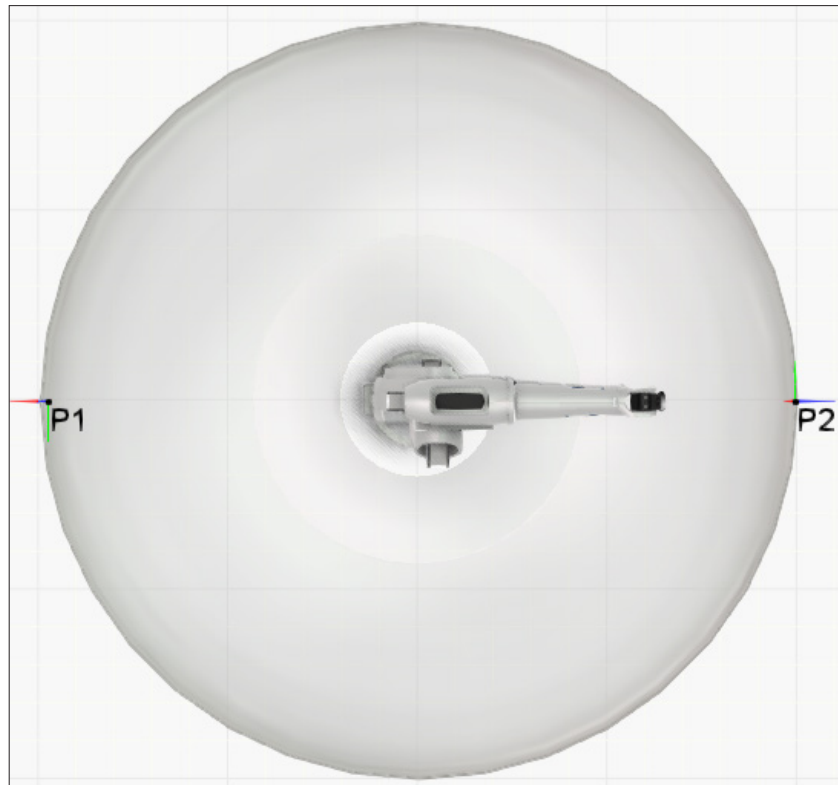
Upgrade option for a selected robot component

The Program tab on the ribbon displays a workspace for teaching robots. A **robot program** in Visual Components is written in robot sequence language or **RSL**. Routines in a program are sequences of tasks that a robot can perform during a simulation. Tasks are known as **statements** and are positioned on a grid that defines the order of execution.

A robot has a **RSL Program Executor** that is responsible for running a robot program during a simulation. Every program has a default **Main** routine while other sequences can be nested with one another and called as **subroutines**. A robot executor allows you to loop a robot program and call specific routines using connected components, for example a pedestal, resource manager or connected PLC controller.

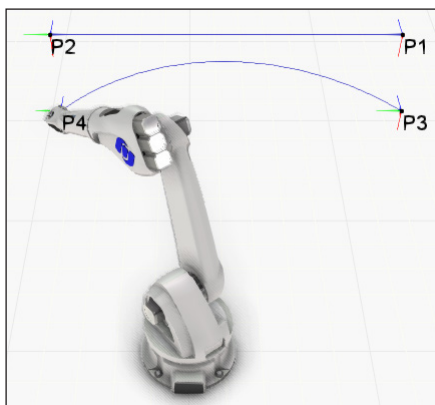
A robot program can be saved in an RSL file and opened by any robot component in the 3D world. In some cases, you may need to upgrade the executor of a robot component in order to teach and edit its program.

A **robot position** is a point in the 3D world for a robot to reach with the end of its arm. The **workspace** or area of reachability for a robot indicates where a robot can reach given its current location.



*The envelope of a robot shown in the 3D world*

The motion of a robot to reach a point can be **linear** or **point-to-point/joint** (interpolation). A **configuration** is the values of joints in a robot at a reached position.



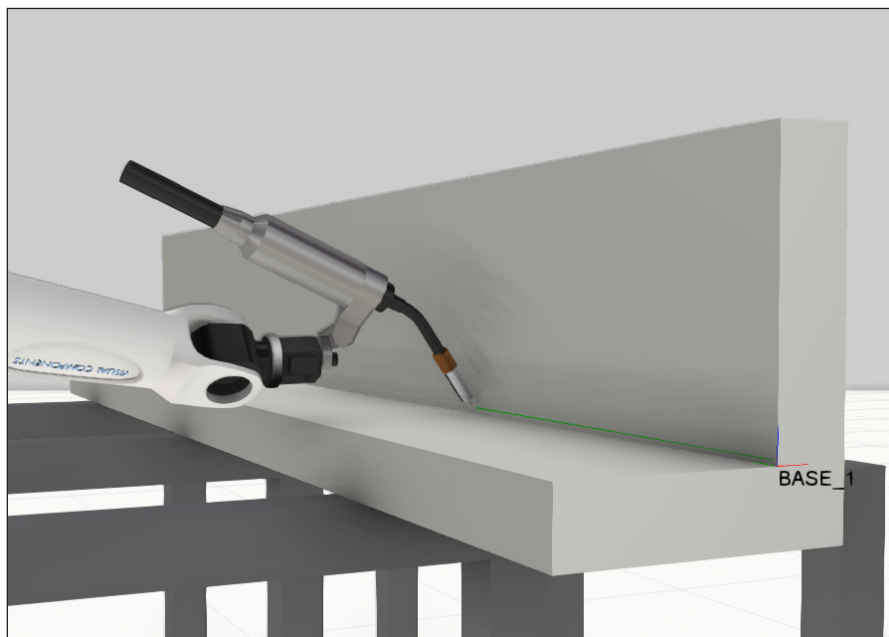
*Tracing robot motion paths*

Joints can be interacted with or given values to reach a position (forward kinematics). Predefined joint configurations can be used and Visual Components adheres to the naming convention used by robot manufacturers. Joint values can be automatically calculated based on the location and orientation of a **tool center point** (inverse kinematics).

A tool center point (TCP) is known as a **tool frame** and can be used to reference the position a robot moves to in the 3D world. For example, a motion statement by a robot could be completed when a referenced tool frame reaches a robot position/motion target.

A **base frame** can be referenced as a parent coordinate system to simplify coordinates for robot positions. That is, robot positions would refer to the base's coordinate system.

Robots can use signals to connect to other components can trigger certain actions. For example, you can signal **grasp** and **release** actions, **trace** the motion path to a robot position, **mount** and **dismount** tools, and start or stop a conveyor.



**Signal mapping** allows you to control the I/O of a robot and trigger events during a simulation. For example, you can synchronize movements, start and stop machines and conveyors, and execute specific routines in other robots.

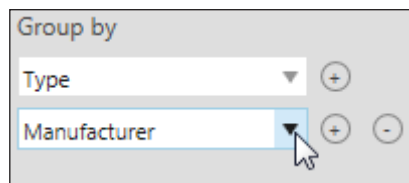
## To open and select a robot program

The Program tab allows you to select and teach robots in the 3D world. The **Jog** command and panel are used for configuring a robot, interacting with its joints and teaching positions. The **Program Editor** panel displays a robot program and commands for creating and editing routines and statements.

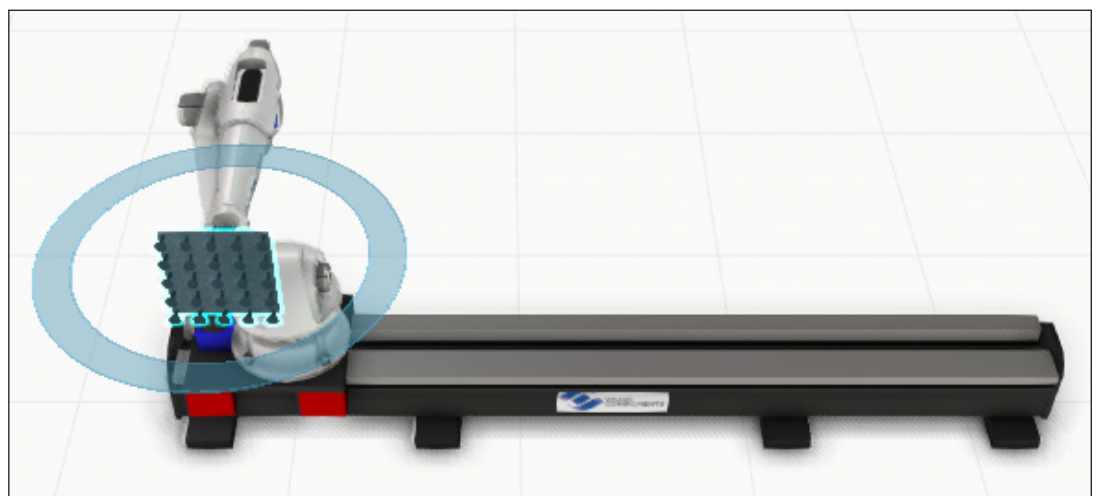
### Sorting by manufacturers

There are over one thousand robots in the Visual Components Web eCatalog, so it helps to sort items by type and manufacturer.

1. Create a new empty layout in the 3D world.
2. In the eCatalog panel, Collections view, expand **WebCatalog by Type**, and then select **Robots**.
3. Double-click **WebCatalog by Type** to access its editor, and then add another Group filter set to **Manufacturer**, and then click **Save**.



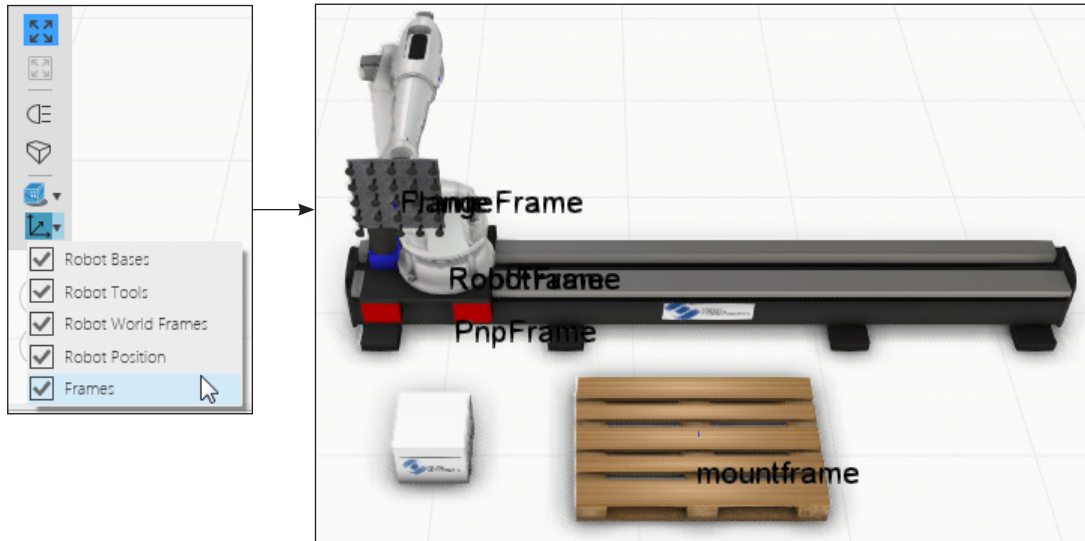
4. In WebCatalog by Type, expand **Robot Positioners**, and then select **Visual Components**, and then add a **Generic servo track** item to the 3D world origin.
5. Expand **Robots**, and then select **Visual Components**, and then add and connect a **Generic Articulated Robot v2** item to the platform of the track in the 3D world.
6. Expand **Tools**, and then select **Visual Components**, and then add and connect a **Parametric Suction Cup Gripper** item to the mount plate of the robot in the 3D world.



## Creating workcell

This module will keep things simple by showing you how to teach a pick and place operation to a robot mounted on a track. The operation will involve the robot picking a box with an end-of-arm tool (EOAT), and then placing the box on a pallet.

1. Add a **Euro Pallet** to the 3D world at XYZ location (1300, -1000, 0).
2. Add a **Visual Components Box** to the 3D world at XYZ location (0, -1000, 0).
3. On the 3D world toolbar, enable **Frame Types** and set each frame type to be visible in the 3D world.

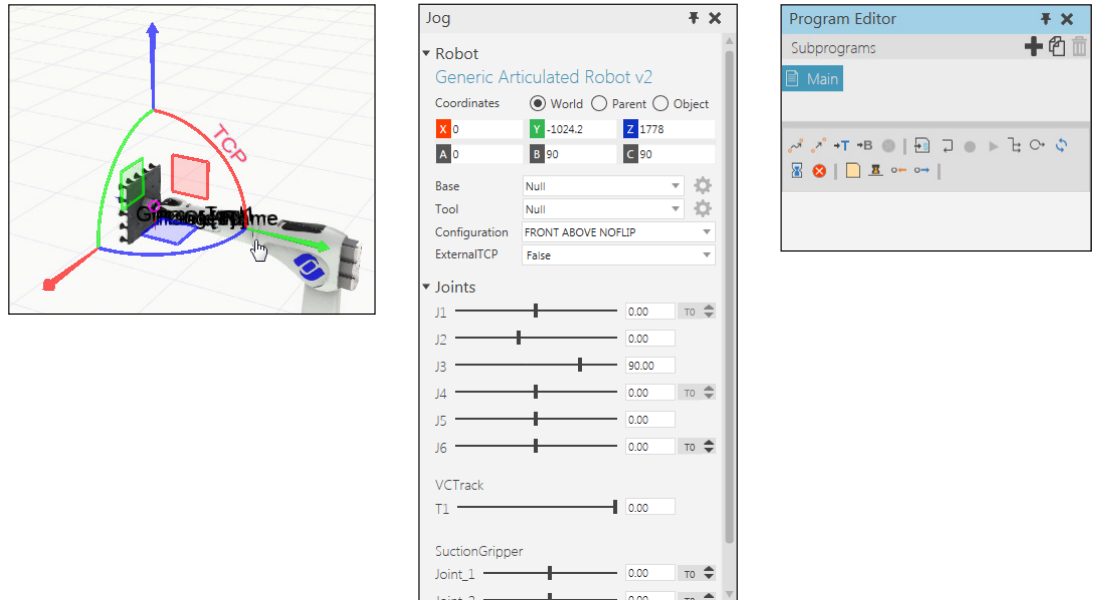


A **Frame feature** is a point of reference in the 3D world. Components and other types of objects use Frame features to determine points in a space, distances, rotations and paths. Frame features are named according to their use and can be selected, manipulated and snapped to in the 3D world.

## Selecting robot executor

The Jog command is used to select the robot executor of a component and interact with its joints. The whole component is not selected rather its robot executor and program.

1. On the Program tab, in the Manipulation group, click **Jog**.
2. In the 3D world, click the **robot**.



When a robot is selected its program is displayed in the Program Editor panel, a manipulator appears at the default tool center point location in the robot, and the Jog panel displays the configuration of the robot and properties for internal and external joints.

3. (Optional) If you are prompted in the Program Editor panel to upgrade the executor of a robot, click the **Upgrade Rsl executor** button.
4. Save the layout as **Module3.vcm** in the My Models folder in your Documents library.



## To interact with joints in a robot

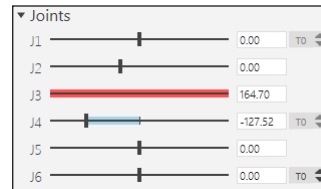
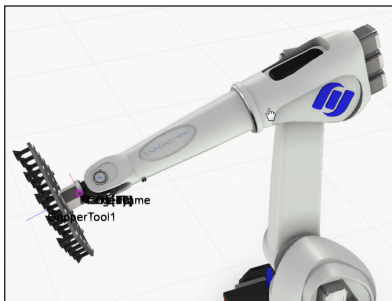
The joints of a robot can be moved and rotated based on their **constraints**. For example, a degree of freedom (DOF) defines the axis and type of motion for a joint, for example linear or rotary. Minimum and maximum values are **limits** for the range of motion of a joint.

### Manipulating local joints

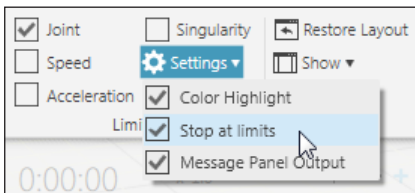
The joints of a robot are listed by name and value in the Jog panel. A slider is used to show the current value of each joint in its range of motion as well as min and max values reached during the execution of robot motions.

1. In the 3D world, point to a part of the robot until the pointer becomes a hand icon, and then drag the pointer to interact with that joint.

A joint will be highlighted red both in the 3D world and Jog panel if the joint exceeds its limits. It may be helpful to prevent a robot from exceeding joint limits during a simulation.



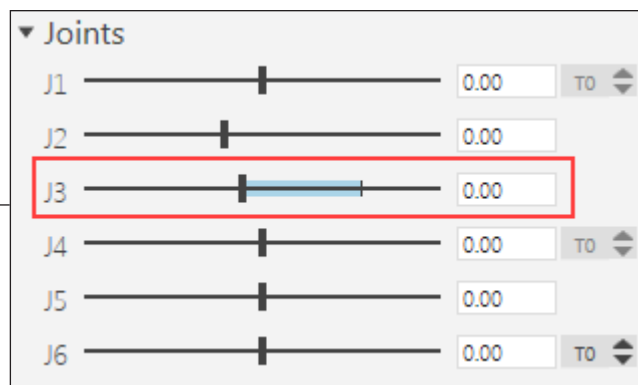
2. On the Program tab, in the Limits group, verify the Joint check box is selected, and then click **Settings**, and then select the **Stop at limits** check box.



3. Reset the simulation, and then in the Jog panel, Joints section, J3 box, type **0** and then press ENTER.



Joint zero position

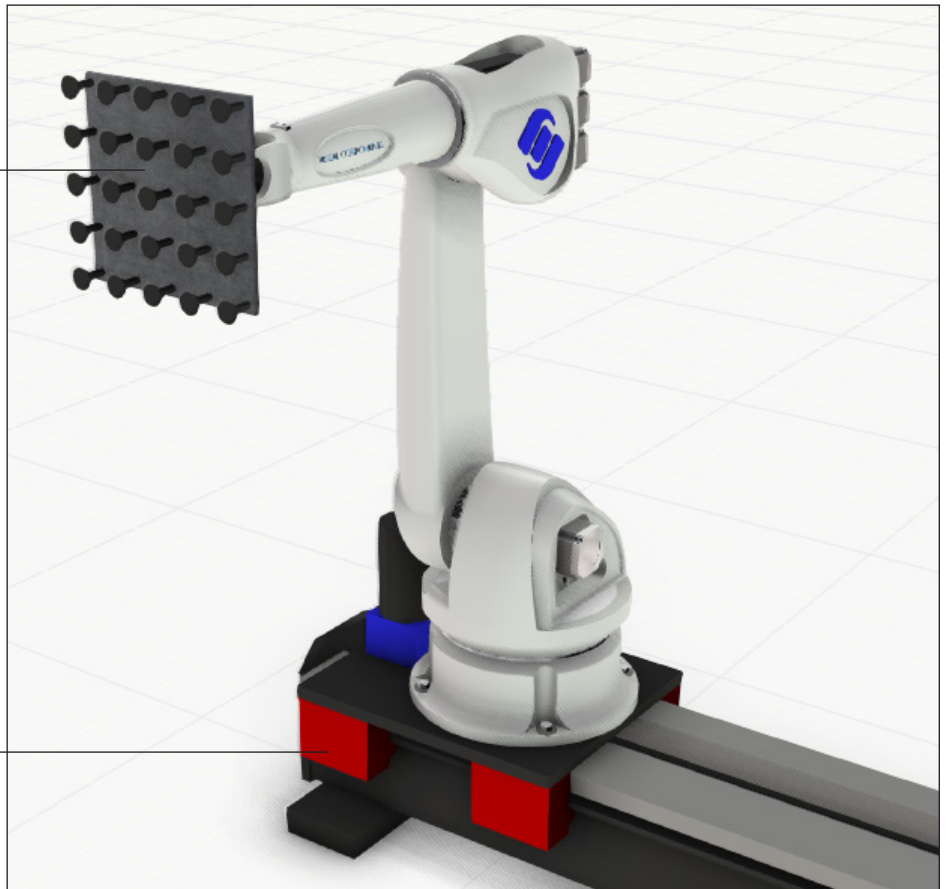


## Manipulating external joints

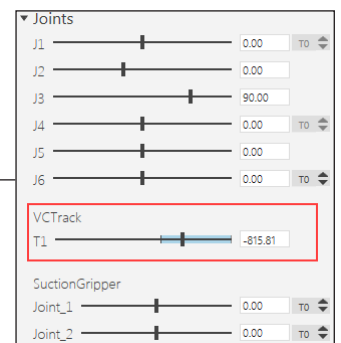
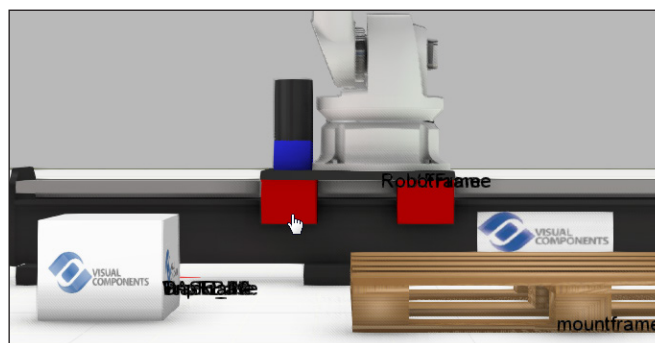
Joints can be imported and exported using interfaces to expand the reachability of a robot.

The Parametric Gripper has two joints that are not interactive.

The Generic Track has one joint that is interactive



1. In the 3D world, jog the platform of the track in the 3D world, which will also move the robot.

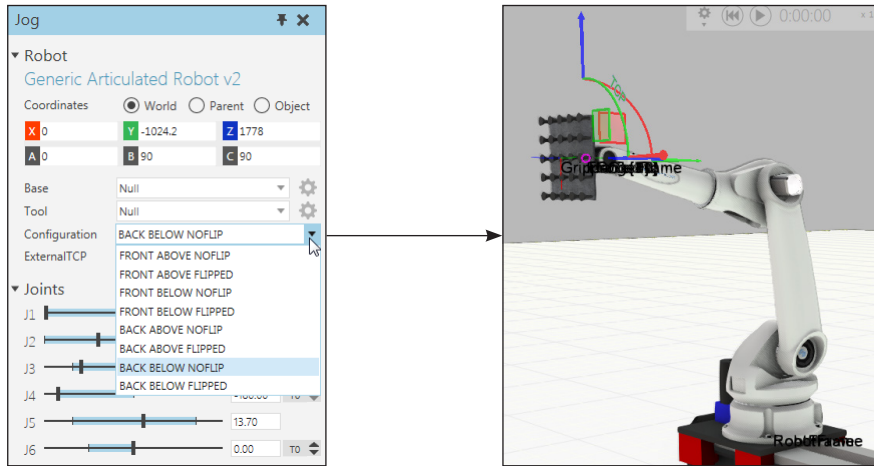


2. Reset the simulation.

## Using joint configurations

A robot can have a set number of configurations for its joints. The names of configurations in a robot adhere to the naming conventions used by real-world robot manufacturers.

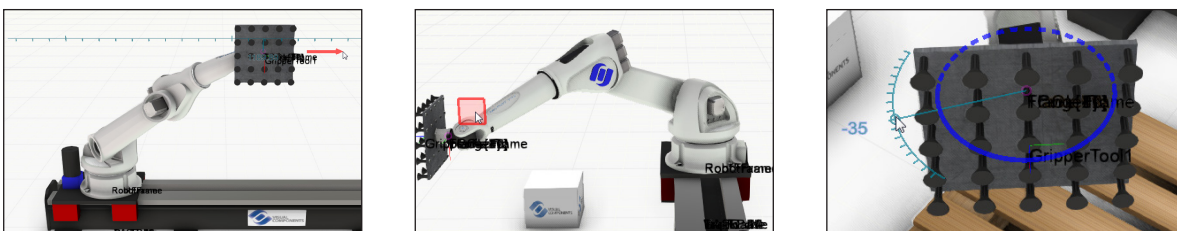
1. In the Jog panel, Robot section, Configuration list, select each configuration to verify joint values and the pose of the robot in the 3D world.



## Using a tool center point

If a robot supports inverse kinematics, the joint values and configuration of a robot will be automatically calculated relative to the position and orientation of an active tool frame.

1. In the Jog panel, Robot section, set Tool to **TOOL[1]**.
2. In the 3D world, drag the **X-axis** arrow of the manipulator in the positive direction as far as the robot can reach relative to its current position.
3. Drag the **YZ** plane of the manipulator toward the floor to jog the robot using a plane..
4. Drag the **Z-axis** ring of the manipulator to configure the robot based on rotations around the Z-axis.



**TIP!** When using the manipulator, drag the pointer along an axis line or ring to increment/step through each tick mark value.

5. Reset the simulation, and then refer to XYZABC values in the Jog panel. These values are the position and orientation of robot arm's end point relative to its base/robot coordinate system. For example, a base-tool configuration is the location from an active base frame to tool frame in robot. An external TCP would invert the relationship.

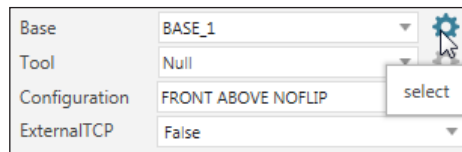
## To edit base and tool frames

Base and tool frames of a robot can be selected, moved and attached to other components in the 3D world.

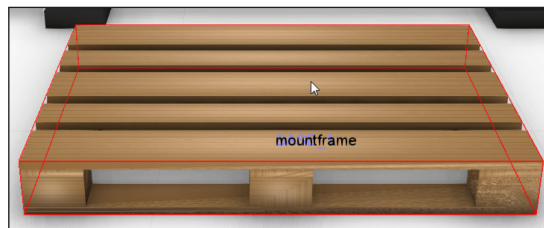
### Bases

In some cases, it is beneficial to attach a base frame to a workpiece component or specific area to localize and simplify robot positions.

1. In the Jog panel, set Base to **BASE\_1**, and then click the **Base** button to select the active base frame in the 3D world, list its properties in the Properties panel and activate the Move command.

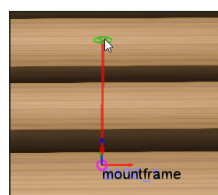


2. In the Properties panel, click the **Node** button, and then click the **pallet** in the 3D world to attach the base frame to that pallet.

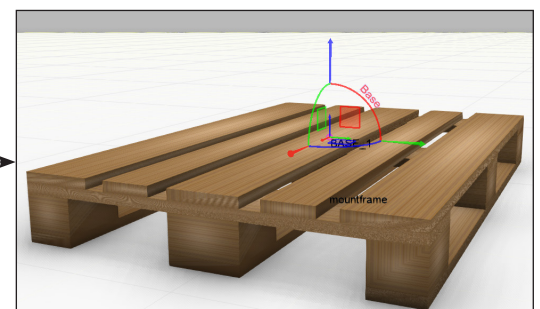
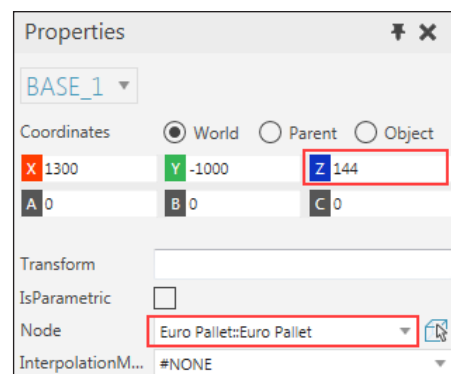


3. Do one of the following:

- In the 3D world, drag the **torus/pink doughnut** of the manipulator to the center of the top face of the top third board of the pallet.



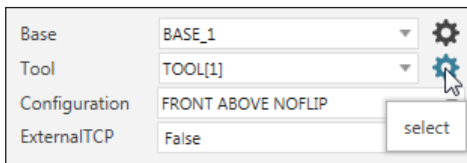
- In the Properties panel, set the Z-axis value to **144**.



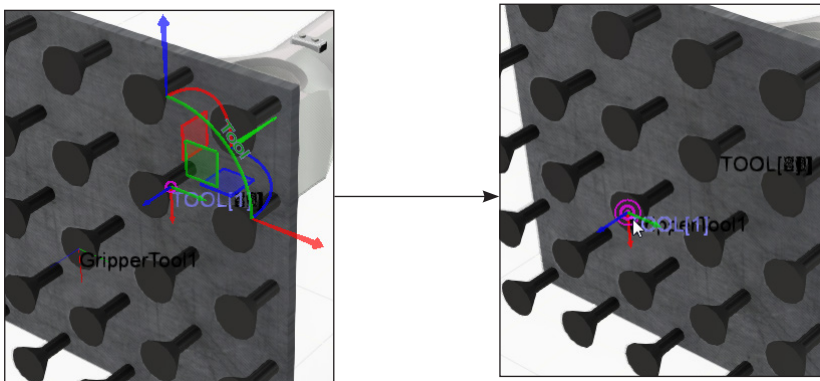
## Tool center points

In most cases, it is beneficial to set a tool frame in the robot to the same position and orientation of an imported tool frame. Why? There are a couple of reasons. First, when the component of an imported tool frame is disconnected from a robot, the robot no longer has a reference to that tool frame and any changes made to that tool frame while connected to a robot will be undone. By default, tool frames in a robot are mapped to signal actions, thereby making it easier to signal grasp and release and other types of actions at predefined tool frames.

1. In the Jog panel, set TOOL to **TOOL[1]**, and then click the **Tool** button to select the active tool frame in the 3D world, list its properties in the Properties panel and activate the Move command.



2. In the 3D world, drag the **torus** of the manipulator to the imported GripperTool1 frame.



The manipulator for a base or tool frame will always show two coordinate systems. The larger axes show the active coordinate system in the Properties panel, for example World. The smaller axes show the base/tool coordinate system (Object) and can be used to quickly verify the orientation of a tool center point.

## To create a sequence of statements in a routine

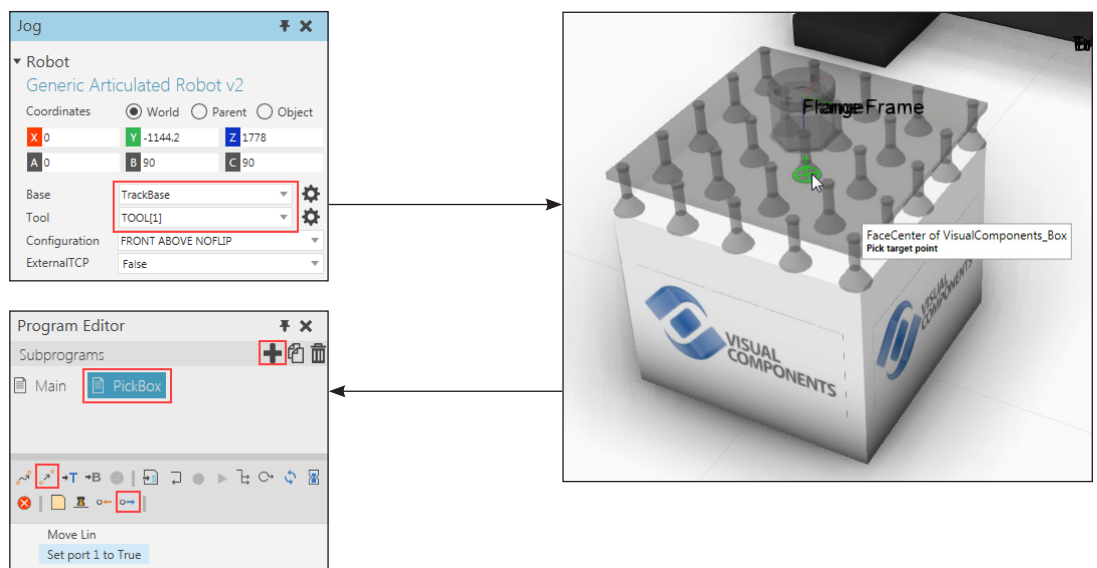
A sequence executes an ordered list of statements one by one. In some cases, the execution of a statement might be delayed and wait for an action triggered by the statement to complete, for example calling a subroutine or waiting for an input value. In other cases, the execution of a statement is immediate and continues onto to the next statement, even though one or more actions may have been scheduled by the statement, for example calling a remote routine in another robot or setting the value of an output signal.

A simple routine for picking a component is to teach a robot three motion statements (approach, pick and retract) and set the value of a port mapped to a tool frame to True in order to execute a grasp action. In Essentials, a motion statement is both a point and motion.

### Teaching pick position and grasp action

A **pick position** is the point where you want a robot to grasp a component.

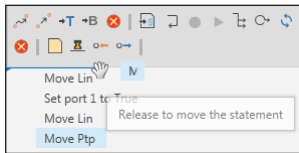
1. In the Program Editor panel, Subprograms pane, click the **plus sign** to add a new sequence.
2. Double-click **Sequence1** to edit its name, and then type **PickBox**.
3. In the Jog panel, Robot section, set Base to **TrackBase** and Tool to **TOOL[1]**.
4. On the Program tab, in the Tools group, click **Snap**, and then point to the center of the top face of the box in the 3D world in order preview a robot motion to that location.
5. In the 3D world, click the top face center of the box to snap the robot to that location.
6. In the Program Editor panel, Statements pane, click **Linear Motion Statement**, and then click **Set Binary Output Statement**.
7. In the Properties panel, with the added Set statement selected, set OutputPort to **1**, and then select the **OutputValue** check box to set the value of port 1 in the robot to True, which by default signals a grasp action to occur at the first tool frame in a robot.



## Teaching retract and approach positions

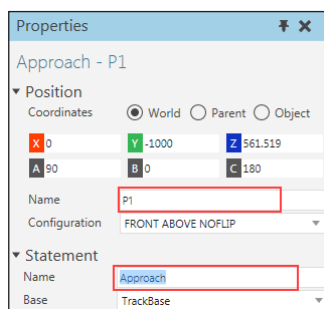
Retract and approach positions may have a similar location but different motion types. Generally, an **approach position** is a joint motion by a robot executed before a pick, and a **retract position** is a linear motion by a robot executed after a pick.

1. In the 3D world, drag the **Z-axis** arrow of the manipulator to approximately **560**, which is enough for the box to clear the height of the pallet.
2. In the Program Editor panel, Statements pane, click **Linear Motion Statement** to define a retract position, and then click **Joint Motion Statement** to define an approach position.
3. In the Statements list, drag the added **Move Ptp** statement to the top of the list.

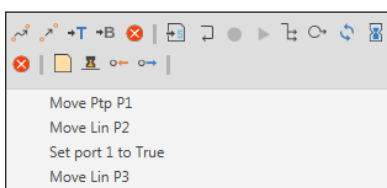


In all cases, you should clearly label statements and robot positions, especially if other people will use the layout and robot program. A motion statement has a *statement name* followed by a *position name* with a dash (-) separating each value. The position name of a motion statement will be displayed in the 3D world.

4. In the Properties panel, do all of the following:
  - In the Position section, set Name to **P1**.
  - In the Statement section, set Name to **Approach**.



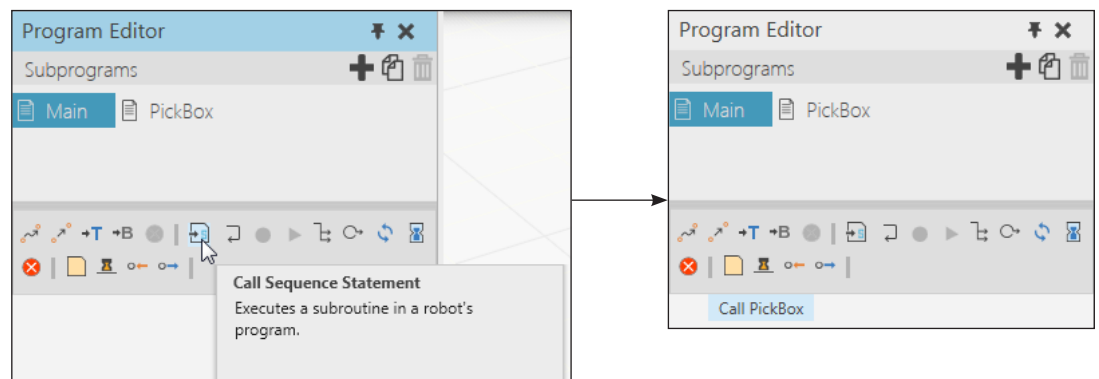
5. Set the second statement to have a name of **Pick - P2**.
6. Set the fourth statement to have a name of **Retract - P3**.



## Calling a subroutine

The Main sequence of a robot is the main loop of its program. Other sequences are subroutines and can be called in any sequence other than itself.

1. In the Program Editor panel, Subprograms pane, click **Main**.
2. In the Statements pane, click **Call Sequence Statement**.
3. In the Properties panel, set Routine to **PickBox**.



4. Run the simulation to verify the robot picks up the box, and then reset the simulation.



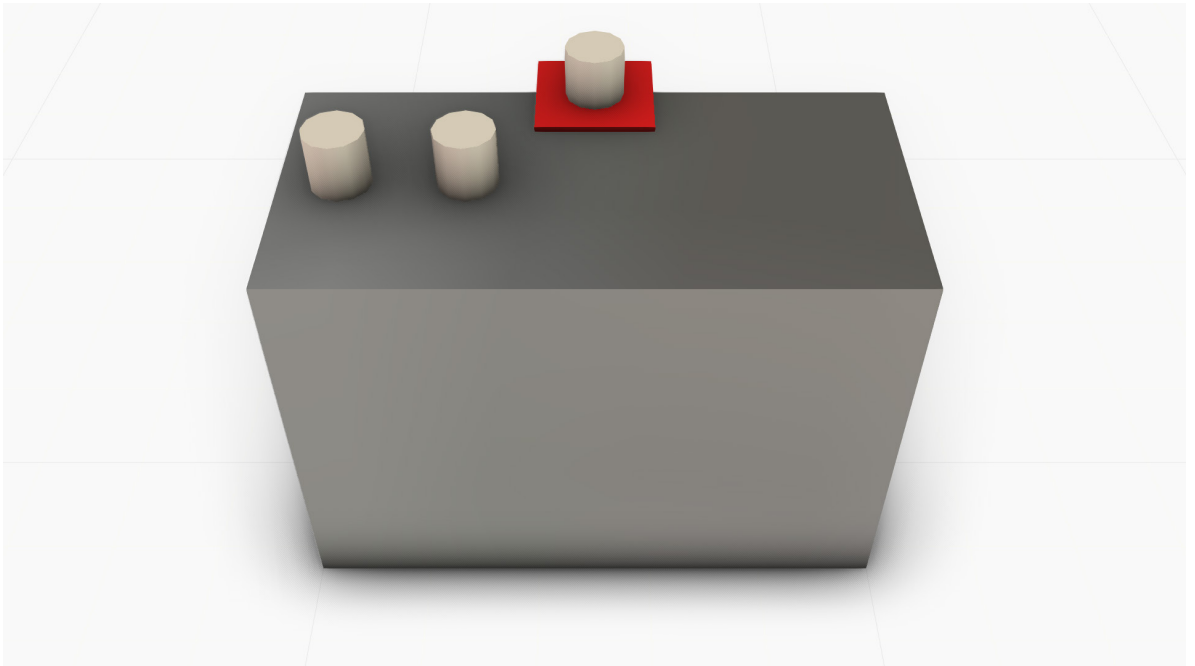
## Review

In this module you learned how to:

- Create and edit a robot program.
- Interact and assign joint configurations for a robot.
- Create and edit robot positions, base frames and tools frames.
- Interact with mounted tools and external joints in a robot.

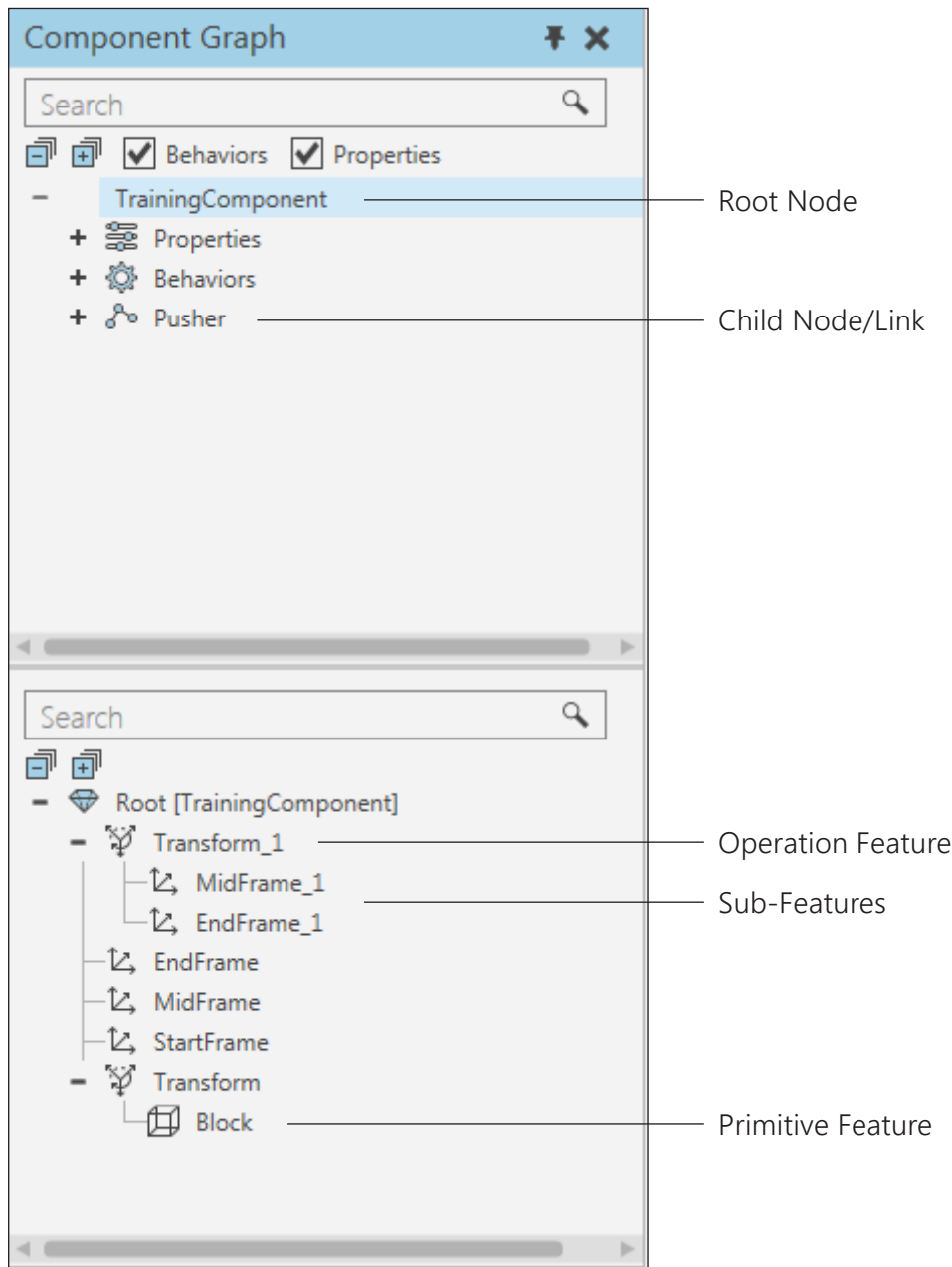
# Module 4 - Model a Component

In this module you learn how to create a component that can be used in a layout.



# Structure of components

A **component** is a container of data that is organized in a tree structure.

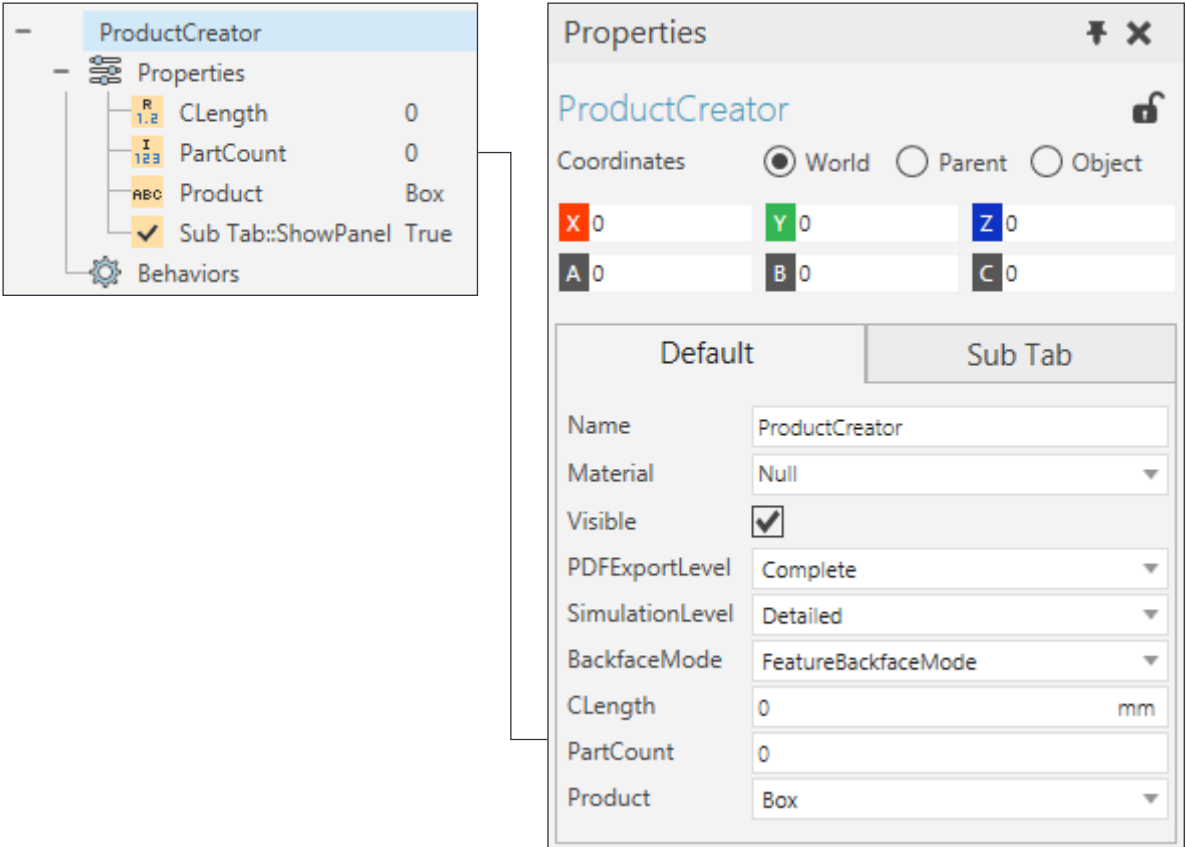


## Nodes

Every component has a **root node** that is a container of features, behaviors, properties, and the origin of the component. A component can have any number of child nodes that are known as links. Generally, these links are used to contain parts of a component that need to move without moving the entire component, for example interacting with the **links** and joints of a robot.

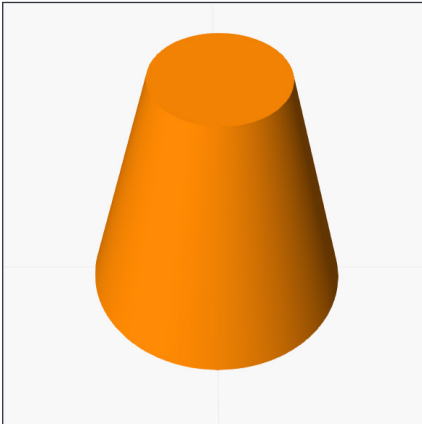
# Properties

A **property** is a variable that can be used to control the values of properties in nodes, features and behaviors. Component properties are contained in the root node, so they can be accessed and referenced in any node of a component, including expressions. Component properties can be shown or hidden from the Properties panel and listed in different tabs.

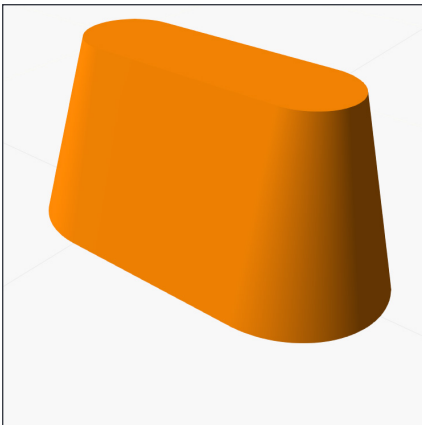


## Features

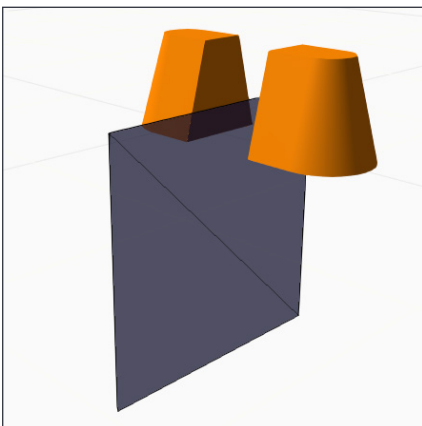
A **feature** is the visual representation of a shape. Primitives are a type of feature that represent basic shapes, for example blocks, cylinders and cones.



Some types of features, for example extrusions and transformations, perform **operations** that modify the shape of subfeatures (nested features).

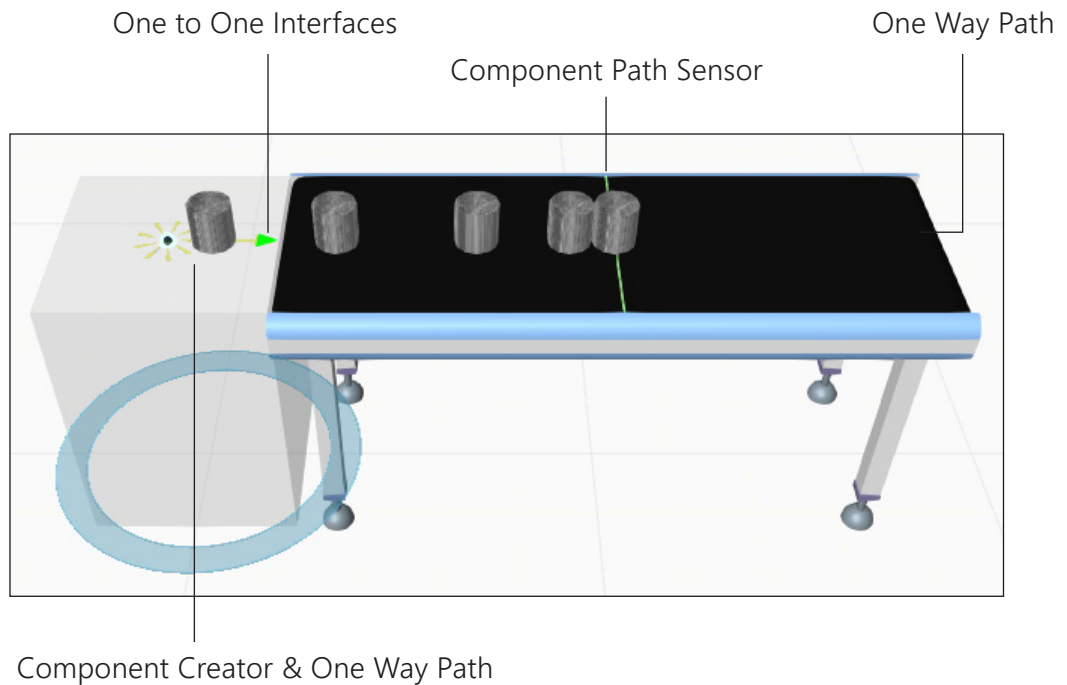


Other types of features contain data specifically related to the 3D world, for example frames and planes. Any type of feature can be collapsed into a **Geometry** feature to allow you to access geometry sets and additional operations.



## Behaviors

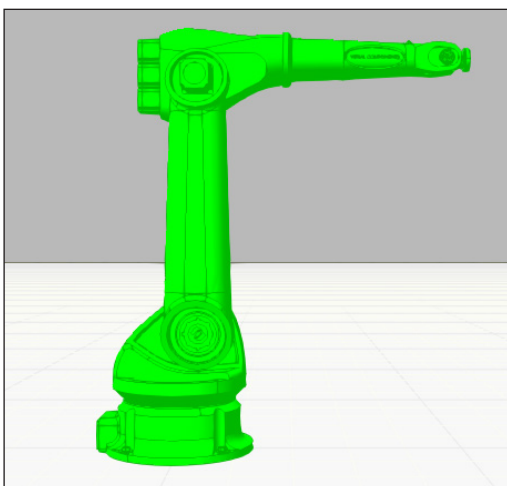
A **behavior** performs a specific task during a simulation. For example, a path can contain and move components in a specific direction. Behaviors can be connected to one another in order to work together. For example, sensors and signals can be connected to a path to detect incoming components.



Generally, behaviors are contained in a root node to make them easier to find, edit and connect with other behaviors. In some cases, a behavior may need to be contained in a specific node in order to work properly. For example, a Component Container at the end of a robot arm is needed to grab and hold onto components.

## Geometry sets

A **geometry set** is a collection of points, lines, edges and triangles. Generally, geometry sets or individual geometry in a set are selected and moved into different features for modeling purposes.



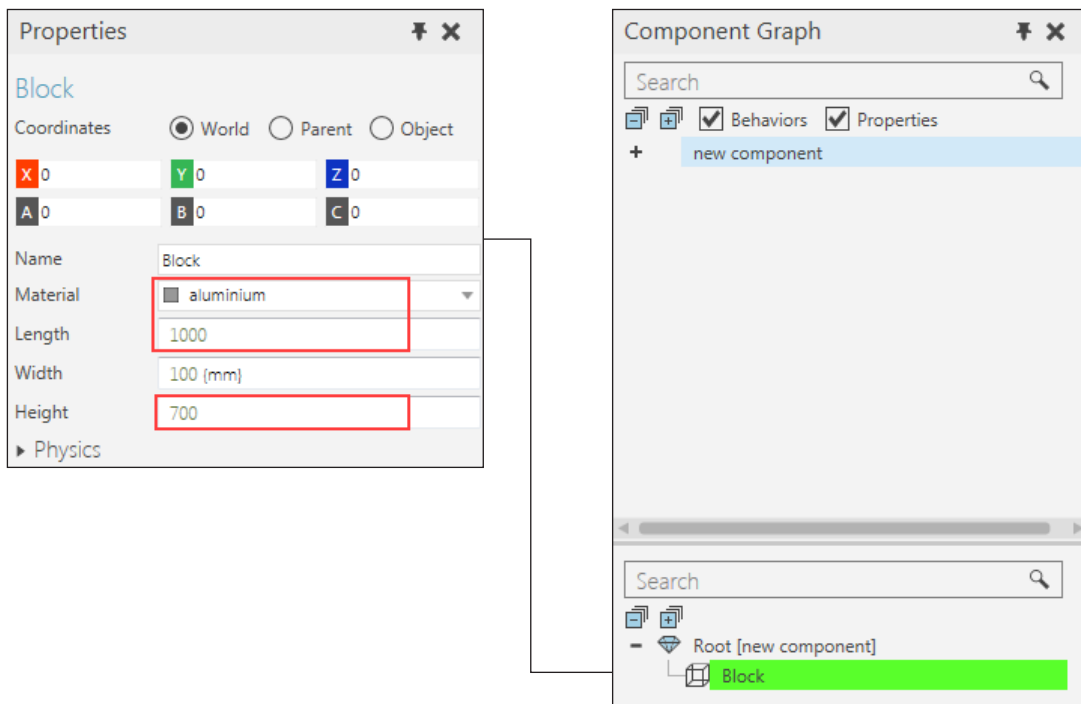
## To create a root node and feature

The Modeling tab displays a workspace for modeling new and existing components. The **Component Graph** panel shows the structure of a component. The **Component Node Tree** pane lists the nodes, properties and behaviors of a selected component. Properties are listed with the root node of a component, and behaviors are listed with their containing node. The **Node Feature Tree** pane lists the features of a selected node.

When you select an object in the Component Graph panel or 3D world, the Component Graph panel will automatically update to show where the current selection resides in a component. If a selected feature contains geometry, the feature will be highlighted **green** in the 3D world. If a selected feature performs an operation, any of its subfeatures containing geometry will be highlighted **olive** in the 3D world.

Every component contains at least one node and should have at least one feature. The preferred modeling environment for components is an empty layout. A new component is created at the 3D world origin and contains no geometry.

1. Create a new empty layout in the 3D world.
2. On the Modeling tab, in the Structure group, click **Create Component**.
3. In the Geometry group, click the **Geometry** arrow, and then in Primitive Geometry, click **Box** to create a Block feature in the root node of the new component.
4. In the Properties panel, set Length to **1000**, Height to **700** and Material to **Aluminum**.



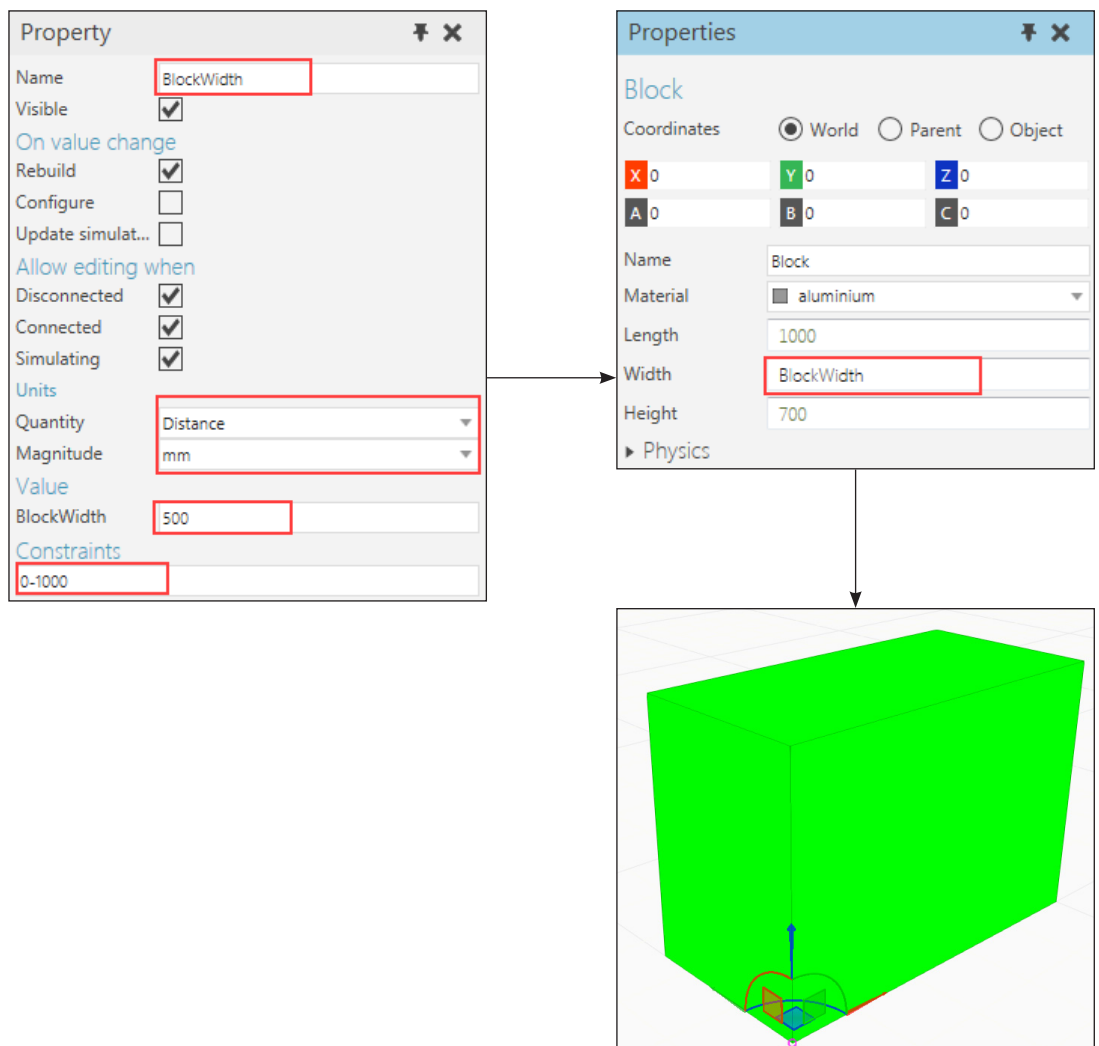
## To transform a feature

Properties allow you to make a component parametric and are visible in the Properties panel. Operation features can also be used to make a component parametric.

### Create and assign properties

A property has its own set of properties, which are listed in a Property task pane. Depending on the property type, you can define constraints. A range is min and max values separated by a dash (-) character. A set of step values are separated by a semi-colon (;) or each new line.

1. On the Modeling tab, in the Properties group, click the **Property** arrow, and then in Basic, click **Real** to create a real type property.
2. In the Property task pane, do all of the following:
  - Set Name to **BlockWidth**, Constraints to **0-1000**, and Value to **500**.
  - Set Quantity to **Distance**, and Magnitude to **mm**.
3. In the 3D world, click the **block**.
4. In the Properties panel, set Width to **BlockWidth**.

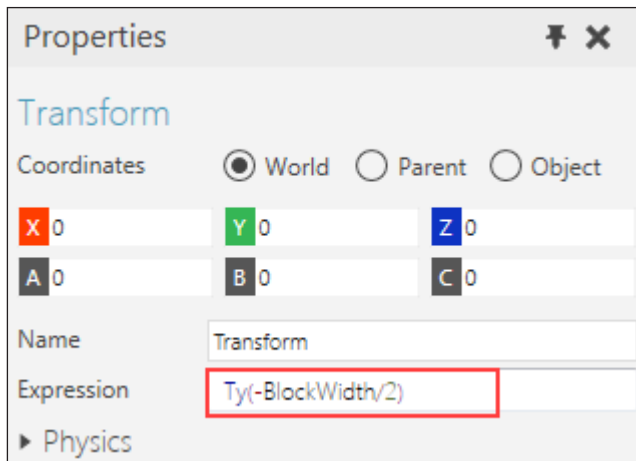




## Create and apply operation features

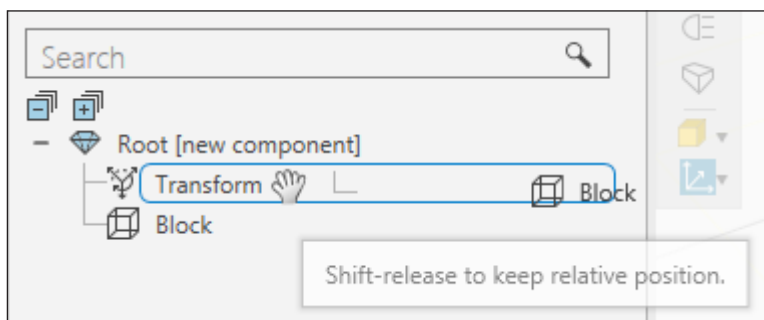
Features can be nested with one another by using a drag-and-drop action in the Node Feature Tree. Generally, features are nested in an operation feature, thereby the operation is applied to those nested features (subfeatures). For example, you can perform a linear transformation on one or more features.

1. On the Modeling tab, in the Geometry group, click the **Geometry** arrow, and then in Transform, click **Transform** to create a feature for writing a linear transformation expression.
2. In the Properties panel, set Expression to **Ty(-BlockWidth/2)** which will translate a subfeature along the Y-axis based on the negative value of BlockWidth divided by 2.



**NOTE!** Generally, expression functions perform translations (Tx, Ty, Tz), rotations (Rx, Ry, Rz) and scaling (Sx, Sy, Sz). For example, Tx(200).Rz(45).Tx(200) is an expression with three functions separated by a dot (.) character.

3. In the Component Graph panel, Node Feature Tree, press and hold SHIFT, and then drag **Block** into Transform in order to nest the feature and transform its relative location.



By default, features will retain their position when nested in another feature or placed in a different node. The SHIFT key is used to allow a feature to inherit the offset of its new parent or have its location transformed by an operation.

**TIP!** Nesting features can be undone and redone using the QAT or CTRL+Z and CTRL+Y.

## To save a component

A component has to be saved separate from a layout. That is, never save a component as a layout.

1. In the Metadata panel, do all of the following:
  - Set Name to **TrainingComponent**.
  - Set Type to **Conveyors**.
  - Set Author to your name.
  - Add a tag of **training**.

Metadata	
▼ Basic Info	
Name	TrainingComponent
Description	
Type	Conveyors
Tags	training
Max Payload	0 kg
Preview	Preview Image Update
Preview Au...	<input checked="" type="checkbox"/>
File	
VCID	6e2ac21e-971c-4c70-96f8-863802277103
Modified	
▼ Authoring	
Manufactu...	
Author	[Your Name]
Email	
Website	

2. On the Modeling tab, in the Structure group, click **Save Component**.
3. Save the file as **TrainingComponent.vcmx** in the My Models folder in your Documents library.

The component is now listed in the eCatalog panel and can be backed up to save your progress at any time.

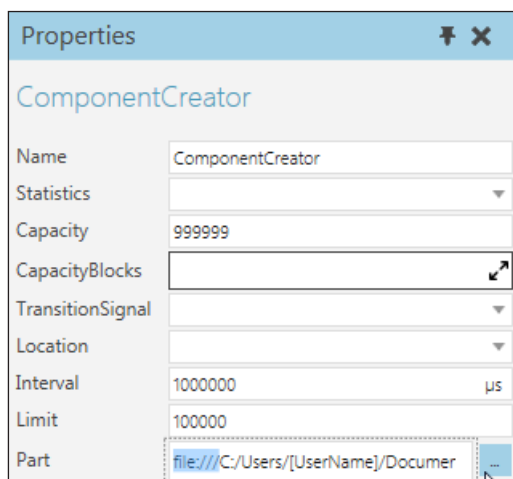
## To create and connect behaviors

Behaviors can be connected to one another to perform a series of related tasks. For example, behaviors that generate components during a simulation need to be connected to behaviors that can contain components.

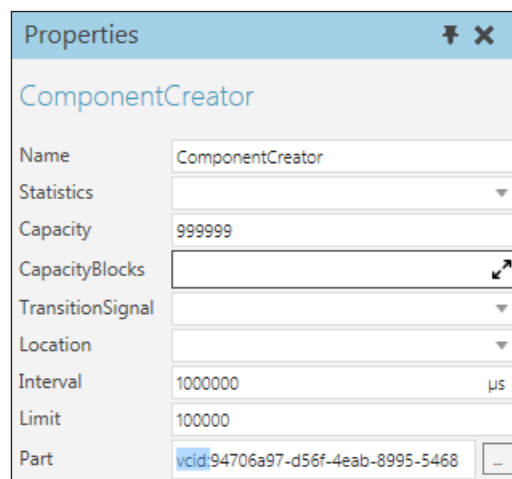
### Create components

Component Creator behaviors can be used to create multiple copies/clones of a template component during a simulation.

1. On the Modeling tab, in the Behavior group, click the **Behavior** arrow, and then in Material Flow, click **Component Creator**.
2. Do one of the following:
  - In the Properties panel, click the **Part** button. Next, in the Open dialog, browse to your **local copy** of the Visual Components Web eCatalog, and then double-click a **Cylinder** component to open and set that file as the template component. In the File Explorer, you may want to do a search of "Cylinder" to find the correct file. Since you are referencing a local file, Part will use a file URI scheme with a prefix of file:///.
  - Click the **Home** tab, and then in the eCatalog panel, select the **All Models** collection, and then in the Display area, search for a **cylinder**. Next, right-click the Cylinder item, and then click **View Metadata**, and then in View Metadata, copy the value of the VCID property, and then click **Close**. Finally, click the **Modeling** tab, and then in the Component Node Tree, click **ComponentCreator**, and then in the Properties panel, Part box, type **vcid:** and paste the VCID property value of Cylinder.



*Browsing to local file*

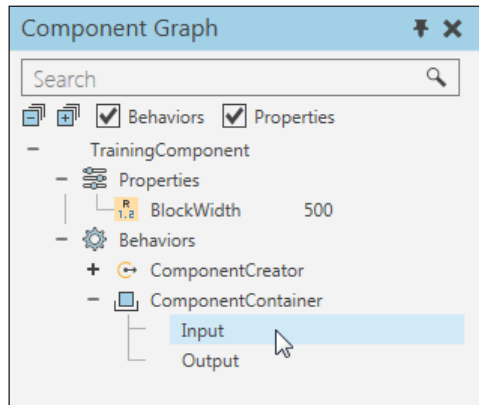


*Giving the VCID of a component*

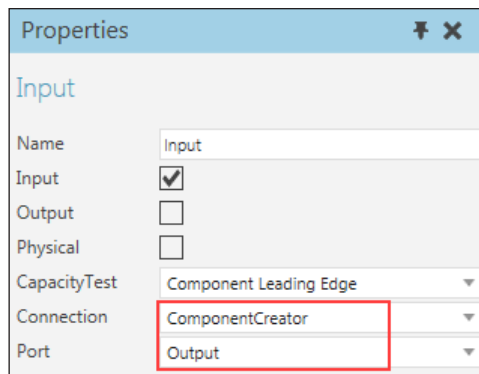
## Contain components

Behaviors that support material flow can be used to transfer components into and out of containers using ports. A **port** is a connector and is listed with its behavior in the Component Node Tree pane. In some cases, a behavior may be a static container and can only transfer in components, for example a Component Container.

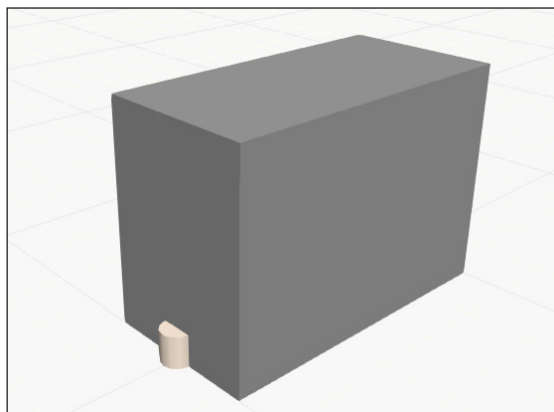
1. On the Modeling tab, in the Behavior group, click the **Behavior** arrow, and then in Material Flow, click **Container**.
2. In the Component Graph panel, Component Node Tree pane, expand **ComponentContainer**, and then click **Input**.



3. In the Properties panel, set Connection to **ComponentCreator** and Port to **Output**.



4. Run the simulation to verify a cylinder is created at the component's origin, and then reset the simulation.



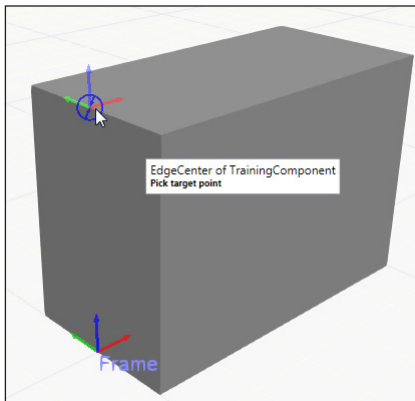
## To create locations for behaviors

Behaviors can reference the location of Frame features to locate where tasks are performed in the 3D world.

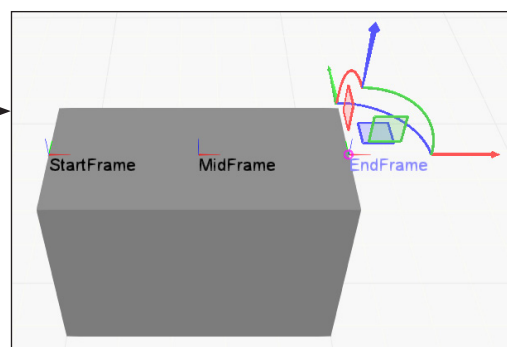
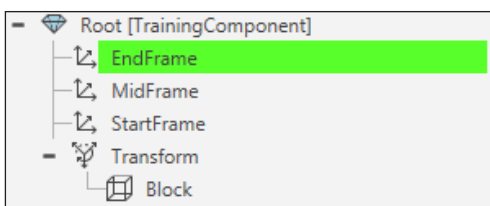
### Create points of reference

A **Frame feature** is a point of reference in the 3D world, which can be used by behaviors to define locations.

1. On the Modeling tab, in the Geometry group, click the Geometry arrow, and then in Other, click **Frame**.
2. In the Tools group, click **Snap**, and then snap the **added frame** to the midpoint of the top edge directly above the frame along the Z-axis.
3. In the Properties panel, set Name to **StartFrame**.



4. Create a new **Frame feature** that is labeled **MidFrame** and located at the center of the top face, a global XYZ location of (500, 0, 700).
5. Create a new **Frame feature** that is labeled **EndFrame** and located at the midpoint of the top edge directly across from StartFrame in the positive X-axis direction, a global location of (1000, 0, 700).



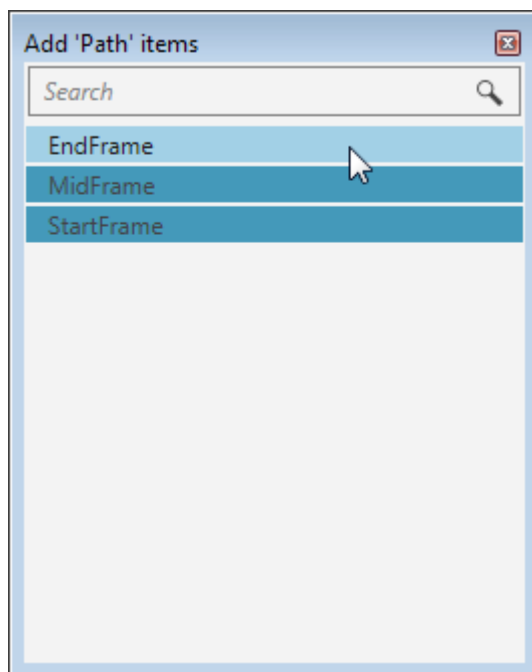
## Create and define paths

A **path** is a type of behavior that can contain and control the flow of components from waypoints defined by Frame features.

1. On the Modeling tab, in the Geometry group, click the Behaviors arrow, and then in Material Flow, click **One Way Path**.
2. In the Properties panel, click the **Path** expand button followed by its **Add** button.

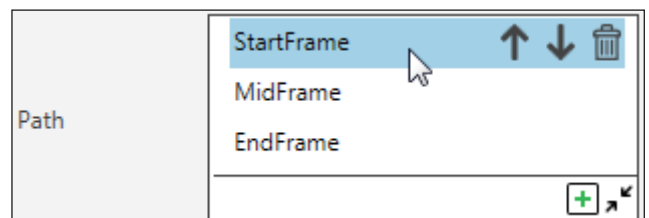


3. In Add 'Path' Items, click **StartFrame**, and then click **MidFrame**, and then click **EndFrame**, and then click **Exit**.



*An already added frame is highlighted dark blue*

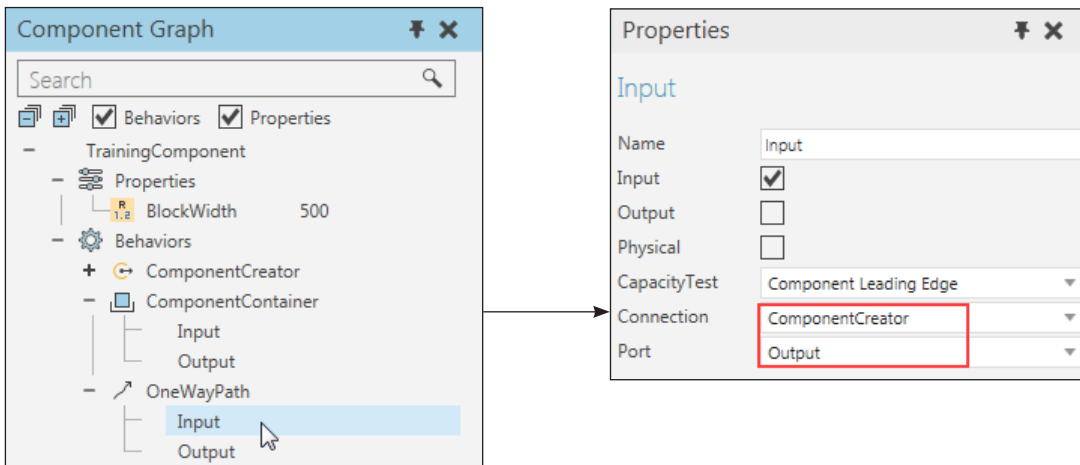
**IMPORTANT!** The order of Frame features defines the waypoints of a path. The first and last waypoint define the endpoints of the path and location of its ports.



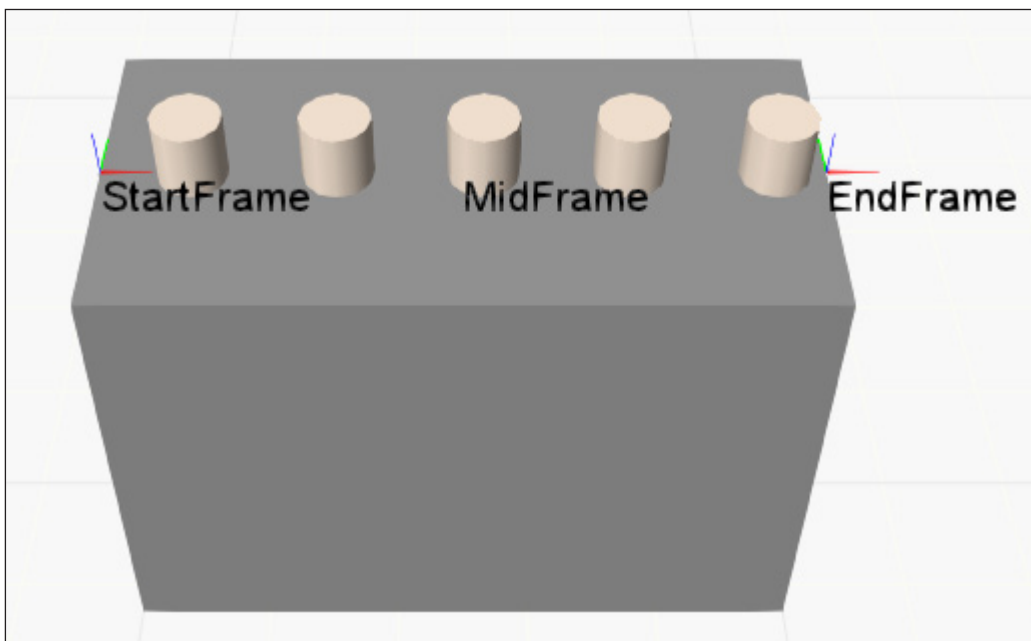
## Switch behavior port connections

The ports of behaviors supporting material flow can be switched to redefine a connection and transfer point for a flow.

1. In the Component Graph panel, expand **OneWayPath**, and then click **Input**.
2. In the Properties panel, set Connection to **ComponentCreator** and Port to **Output**, which will replace the Output connection ComponentCreator had with ComponentContainer.



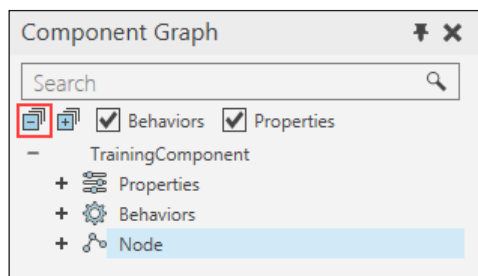
3. Run the simulation to verify components are created and move along the path, and then reset the simulation.



## To create new nodes

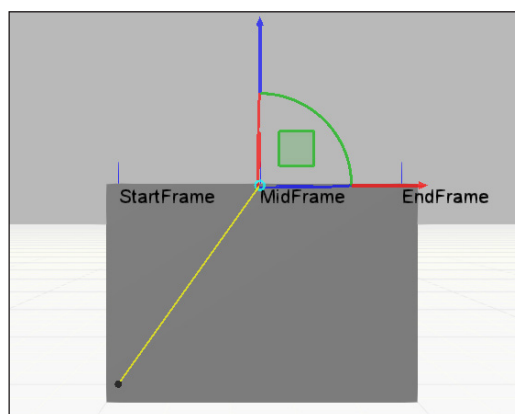
Nodes can be added to a component to contain geometry that moves or performs a specific task during a simulation as well as define joints. For example, the geometry in a robot is contained in several nodes and each node has a joint that are linked together to form a kinematic chain. This allows parts of a component to move independently of one another and be dependent on the movement of other parts.

1. On the Modeling tab, in the Structure group, click **Create Link**.
2. In the Component Graph panel, Component Node Tree pane, click **Collapse** to collapse everything in the tree.



A node can be given an offset from its parent node. Therefore, it is helpful when modeling components to visualize the node structure of a component in the 3D world.

3. In the Properties panel, set Name to **Pusher** and Offset to **Tx(500).Tz(700)** and then press ENTER to define an offset based in the parent node's coordinate system, which currently has the same origin as the World coordinate system.
4. On the Modeling tab, in the Show group, select the **Structure** check box.



*Yellow line is offset of node from parent*

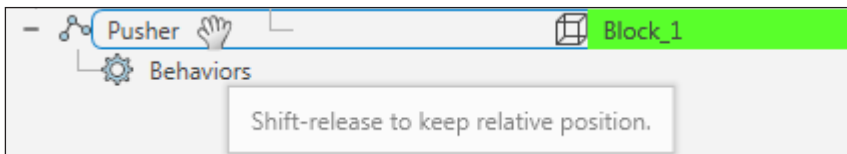
**NOTE!** The torus of the manipulator will be blue to indicate that you are manipulating the offset of a node. In some cases, the torus will be grayed out to indicate that you cannot manipulate the offset of a node. Generally, interaction with a node offset is disabled when you use properties to write an expression that makes a node offset parametric.



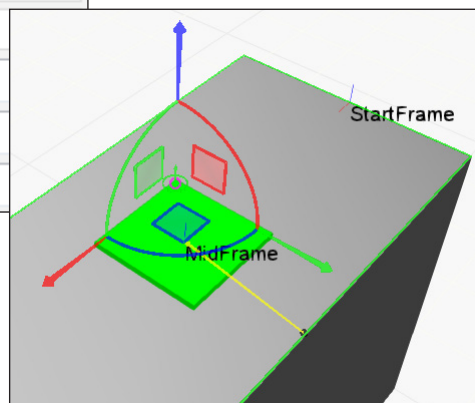
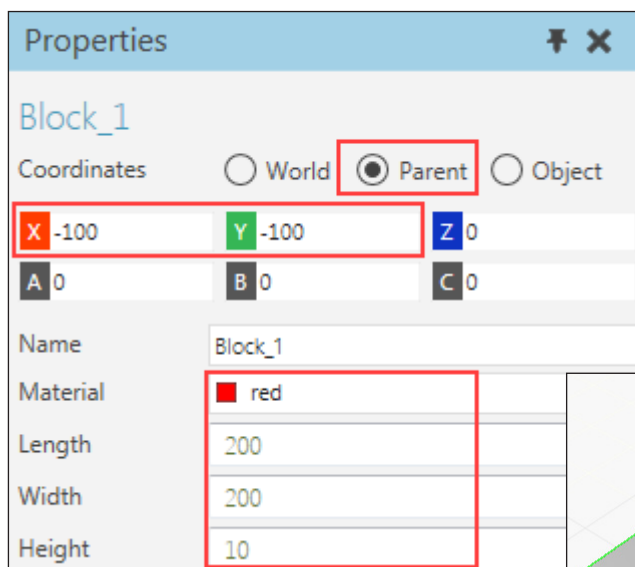
## Offset features

A feature can inherit the offset of a node, thereby affecting the location of the feature in the 3D world.

1. In the Component Graph panel, Component Tree Node pane, click the **root node**.
2. On the Modeling tab, in the Geometry group, click the **Geometry** arrow, and then in Primitive Geometry, click **Box** to create a new feature in the root node.
3. Press and hold SHIFT, and then in the Component Graph panel, Node Feature Tree pane, drag **Block\_1** into the Pusher node listed in the Component Node Tree pane to move that feature to a different node and inherit the offset of that node.



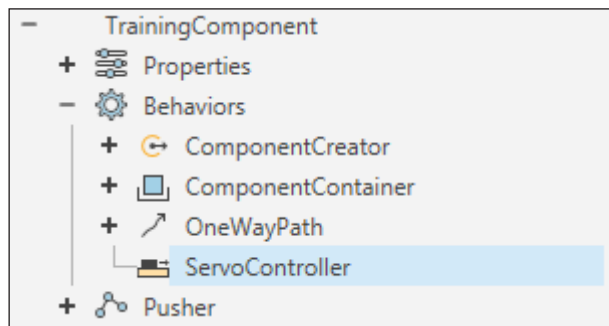
4. In the Properties panel, do all of the following:
  - In Coordinates, click **Parent** to switch from the World coordinate system to the parent (node) coordinate system of the block, and then set the X and Y axis boxes to **-100**.
  - Set Material to **red**.
  - Set Length and Width to **200**.
  - Set Height to **10**.



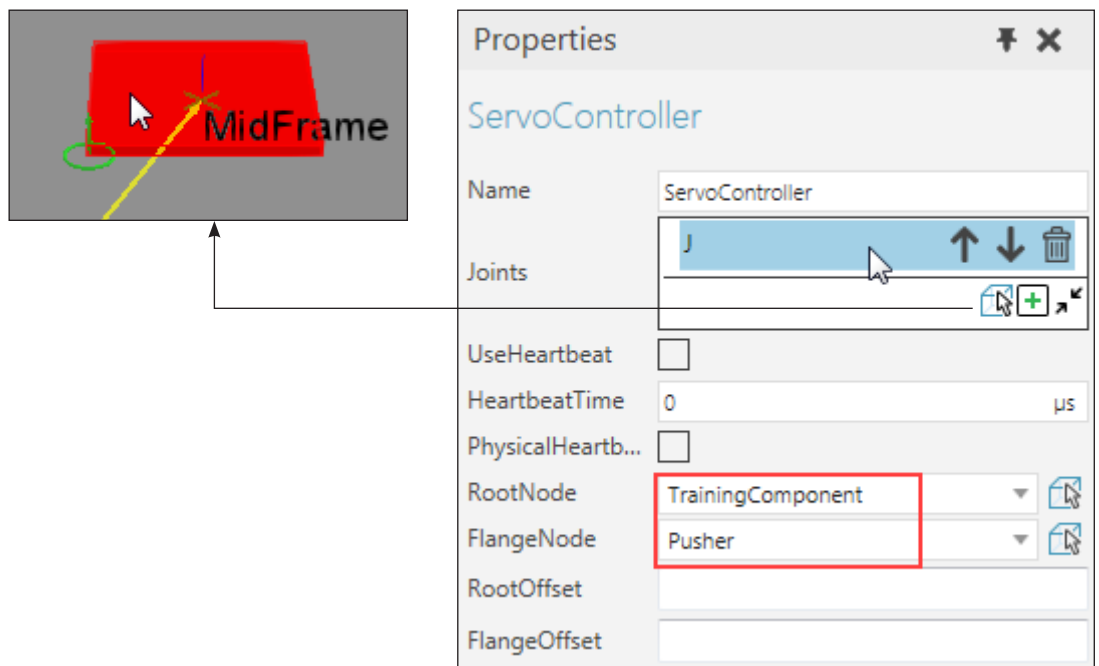
## To assign a joint

A joint provides mobility and the degree of freedom for a node. That is, a joint defines the mechanism of movement in a node. Joints can be driven by assigning them to one or more controllers type behaviors.

1. In Component Graph panel, Component Node Tree pane, click the **root node**.
2. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Robotics, click **Servo Controller**.



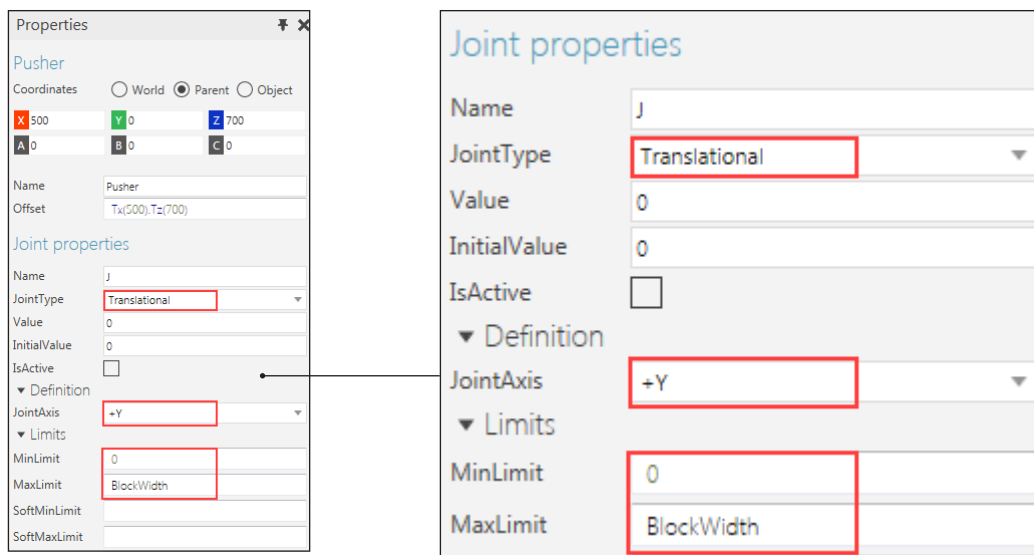
3. In the Properties panel, click the **Joints** expand button followed by its **Pick** button.
4. In the 3D world, click the **red block** of the Pusher node to select and add its joint to the Joints list, thereby assigning the joint to ServoController.
5. In the Properties panel, verify RootNode is set to **TrainingComponent** and FlangeNode is set to **Pusher**.



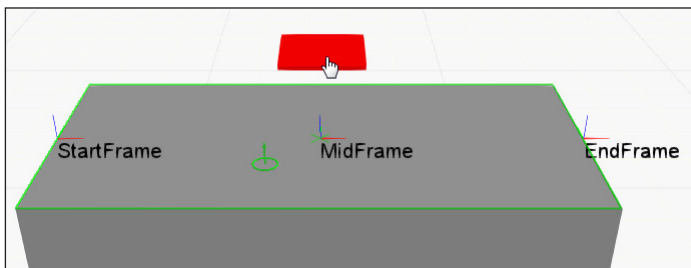
## To define a joint

The properties of a joint are listed with the properties of its node. A joint type can be defined using an expression. By default, the joint of a node is interactive based on its joint definition. Otherwise, you can create a Jog Info behavior to override joint interaction.

1. In the Component Graph panel, Component Node Tree pane, click **Pusher**.
2. In the Properties panel, Joint properties area, do all of the following:
  - Set JointType to **Translational**.
  - In the Definition section, set JointAxis to **+Y**.
  - In the Limits section, set MinLimit to **0** and MaxLimit to **BlockWidth**.



3. On the Modeling tab, in the Manipulation group, click **Interact**.
4. In the 3D world, jog the joint of the Pusher node by dragging its red block along its Y-axis.



5. On the simulation controls, click **Reset** to return the joint to its initial value of 0.

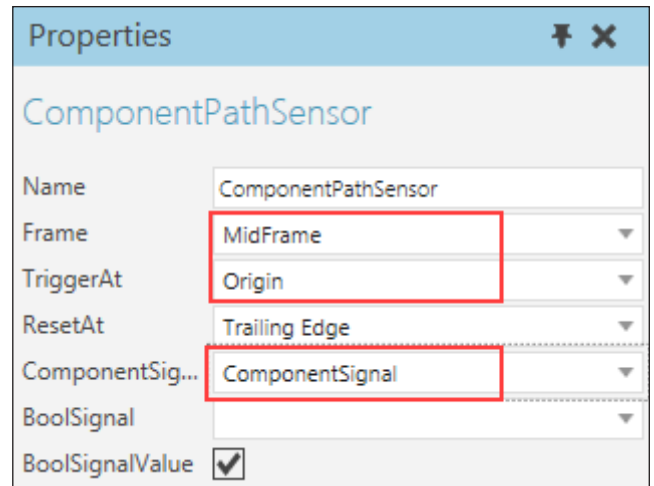
## To detect components and signal events

A **sensor** can be used to detect components moving along a path in the 3D world. Signals can be used to notify other behaviors if and what component triggered the sensor.

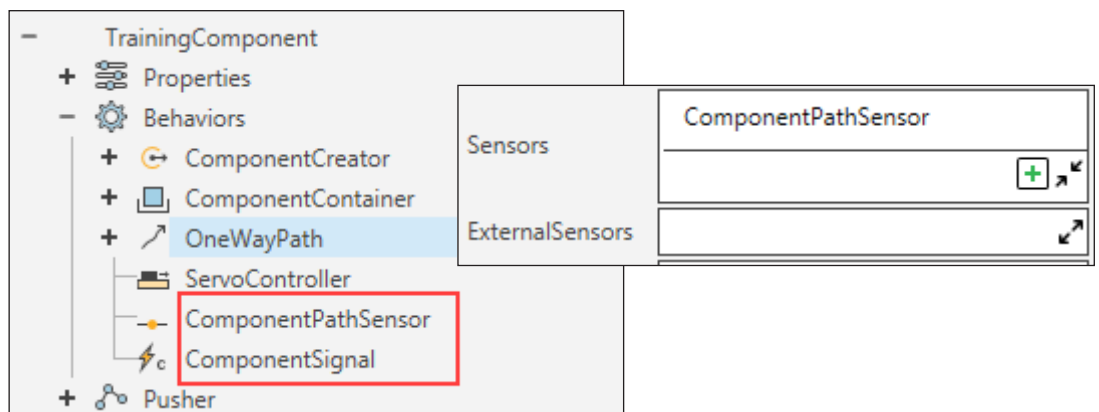
### Connect sensors, signals and paths

A sensor can be used by one or more paths to detect components.

1. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Sensors, click **Path** to create a new Component Path Sensor behavior in the root node.
2. In the Properties panel, set Frame to **MidFrame** and TriggerAt to **Origin** to position the sensor at the middle waypoint of the path and detect a component when its origin reaches the sensor's location.



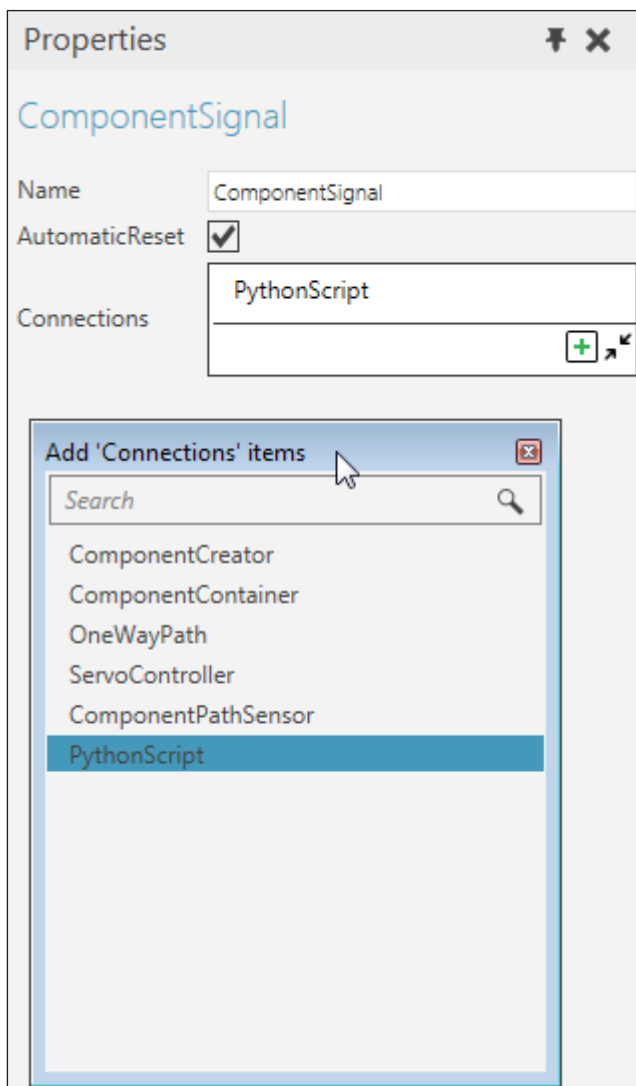
3. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Signal, click **Component** to create a new Component Signal behavior.
4. In the Component Graph panel, Component Node Tree pane, click **ComponentPathSensor**.
5. In the Properties panel, set ComponentSignal to **ComponentSignal**, which can send a signal with a component object value.
6. In the Component Graph panel, Component Node Tree pane, click **OneWayPath**.
7. In the Properties panel, click the **Sensors** expand button followed by its **Add** button.
8. In Add 'Sensors' Item, click **ComponentPathSensor** to add the sensor to the path, and then click **Exit**.



## Signal events

A **signal** can be used to notify other behaviors of its value and signal events during a simulation.

1. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Misc, click **Python Script**.
2. In the Component Graph panel, Component Node Tree pane, click **ComponentSignal**.
3. In the Properties panel, click the **Connections** expand button followed by its **Add** button.
4. In Add 'Connections' Items, click **Python Script**, and then click **Exit**.



When ComponentPathSensor is triggered by a component during a simulation, the Python Script can now be notified by the connected ComponentSignal. This is known as a signal event.

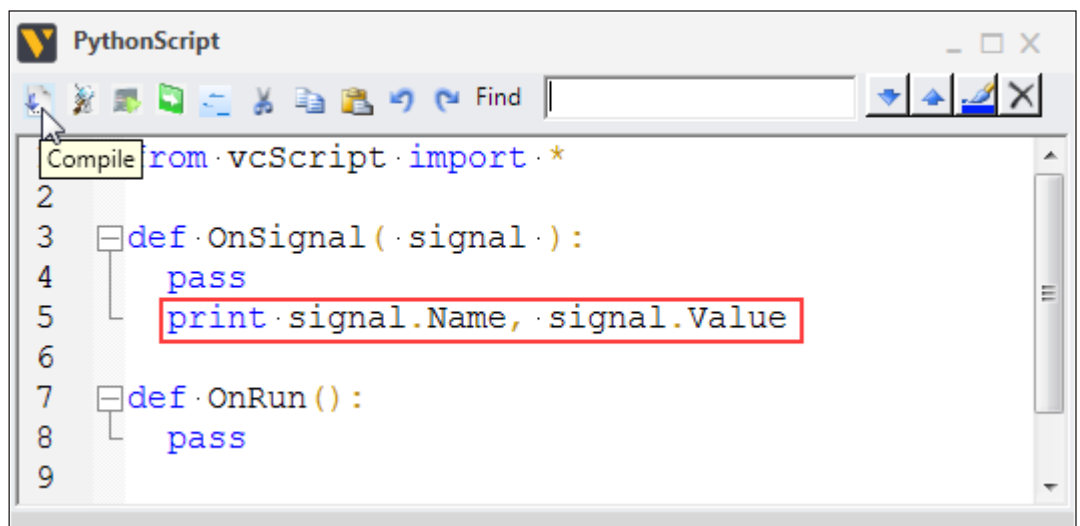
## Print event feedback

Python Script behaviors allow you to write a program using Python version 2.7.1 and Essentials API, which are Python libraries and modules for interacting with Essentials.

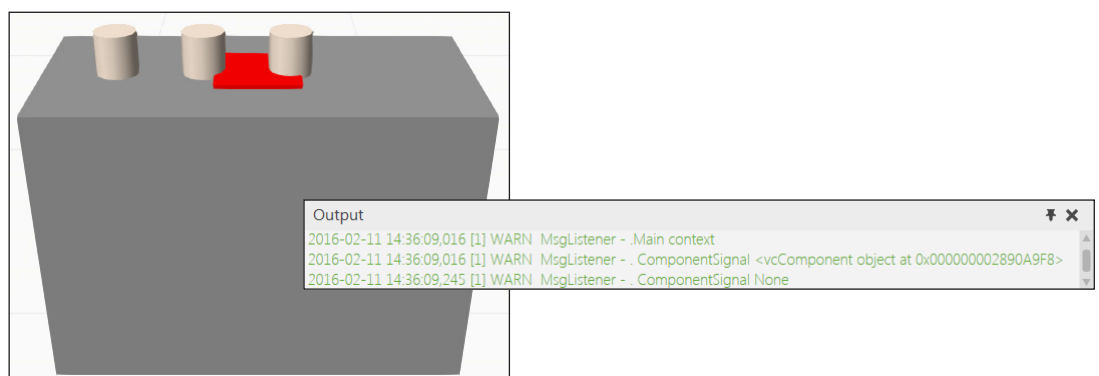
1. On the Component Graph panel, Component Node Tree pane, double-click **PythonScript** to access its editor.
2. In the editor, place the carrot/text cursor at the end of line 4, and then press ENTER to add a new line that is properly indented as an instruction of the OnSignal event.
3. On line 5, type the following code to print the name of a signal and its value whenever the OnSignal event occurs during a simulation.

```
print signal.Name, signal.Value
```

4. In the editor toolbar, click **Compile**, and then click **Exit**.



5. Run the simulation to verify feedback is printed in the Output panel, and then reset the simulation.



6. On the Modeling tab, in the Structure group, click Save **Component**.

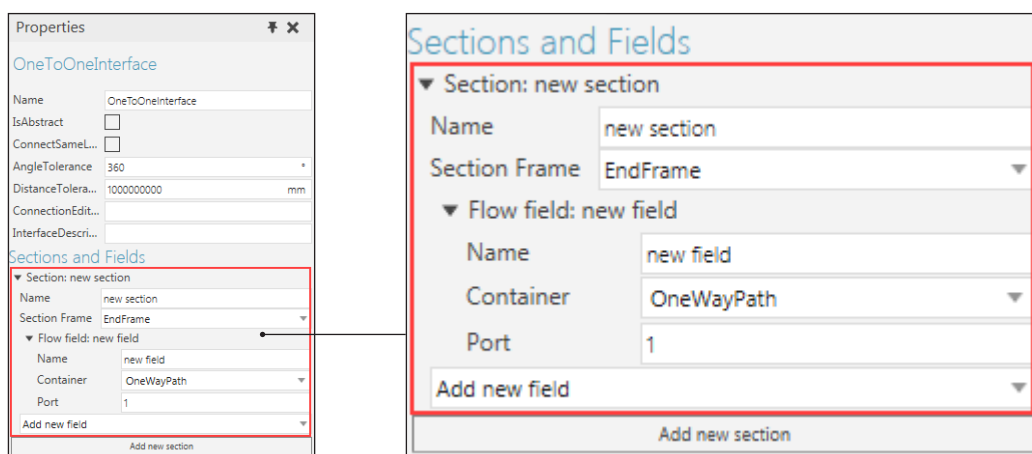
## To physically connect components

Interfaces allow you to connect a component physically and remotely with other components in the 3D world. That is, interfaces are a type of port/connector for transferring data from one component to another.

### Create interface sections and fields

An interface is similar to a power outlet that has input and output fields contained in a **section**, socket or plug. Each section can have multiple **fields** to support the exchange of information between connected devices.

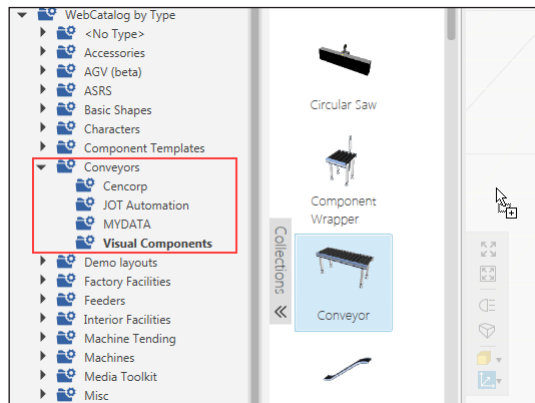
1. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Interface, click **One to One**.
2. In the Properties panel, Sections and Fields area, click **Add new section** to create a new section in the interface.
3. In the added section, do all of the following:
  - Set Section Frame to **EndFrame** to define the physical location of the section.
  - Set Add new field to **Flow** to add a new field in the section that supports material flow.
4. In the added flow field, do all of the following:
  - Set Container to **OneWayPath** to define which behavior will be referenced by the field in order to transfer components.
  - Set Port to **1** to reference the Output port of OneWayPath. That is, the interface can now be used to transfer components out of the path and into a connected interface.



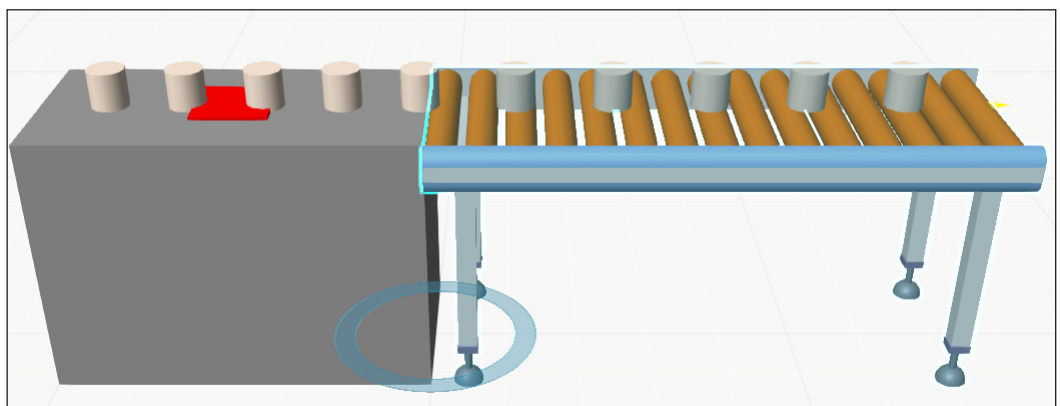
## Connect interfaces

Interfaces allow components to connect to one another by either plugging into one another at a physical location or wiring a connection between one or more sections.

1. Click the **Home** tab, and then in the eCatalog panel, Collections view, expand **WebCatalog by Type** followed by **Conveyors**, and then click **Visual Components**.
2. Drag a **Conveyor** item into the 3D world, and then connect the **conveyor** to TrainingComponent.



3. Run the simulation to verify cylinders move from TrainingComponent to the added conveyor, and then reset the simulation.



4. In the 3D world, click **TrainingComponent**.
5. Click the Modeling tab, and then in the Structure group, click **Save Component**.



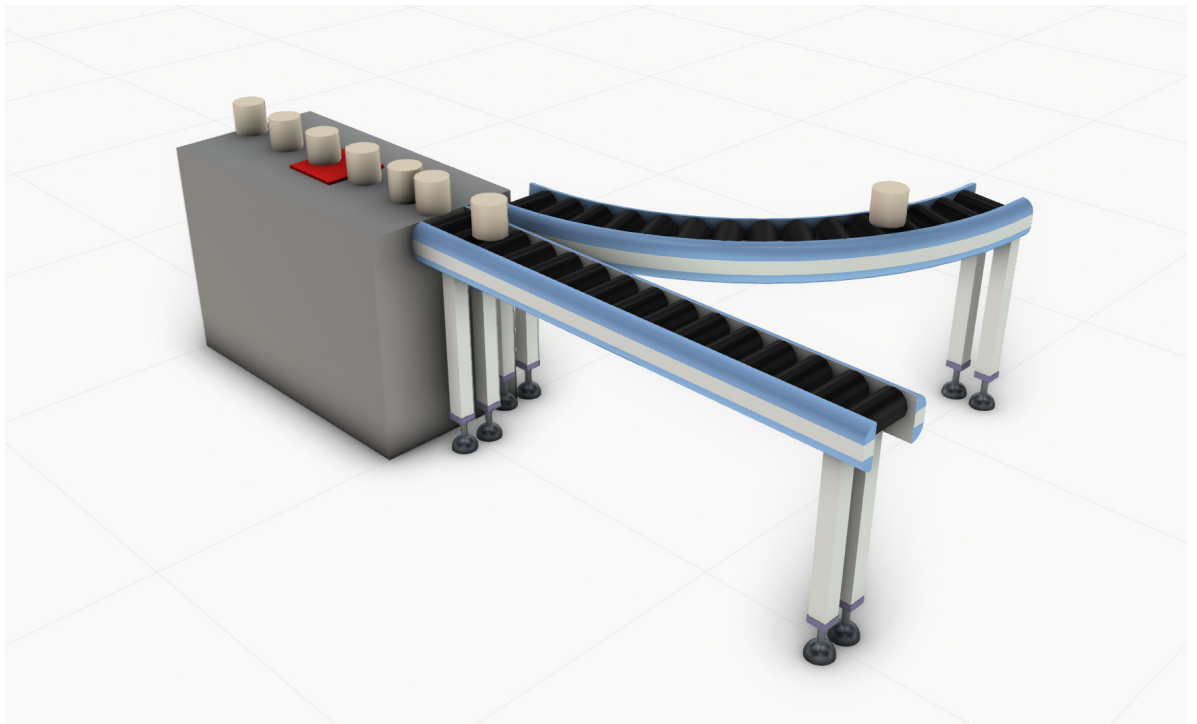
## Review

In this module you learned how to:

- Create a component and its nodes, features, behaviors and properties.
- Use features to perform operations on other features.
- Move geometry into different nodes.
- Define the joint and offset of nodes.
- Reference features to perform tasks with behaviors.
- Connect behaviors to perform related tasks and signal events in a simulation.
- Connect a component to other components in a layout.
- Save a component.

# Module 5 - Modeling with Python

In this module you learn how to use Python to add logic and functionality in a component.



# Python in Visual Components

The following is a quick overview of programming with Python in Essentials.

Visual Components uses **Python 2.7.1** and documentation for that version is available online at <https://docs.python.org/2.7>.

The **Help File** of Essentials contains a reference guide for Python libraries and classes created by Visual Components.

Python has its own **syntax** and **semantics** for writing instructions, which are normally interpreted (executed at runtime) instead of being compiled into an executable file.

In the context of Visual Components, Python is a **scripted language**:

- Some instructions can be executed when a Python Script behavior is compiled by a user and the simulation is not running.
- Other instructions are executed during simulation runtime.
- As a result, edits to a Python Script behavior must be performed when a simulation is not running.

Python is an **object-oriented language**:

- **Properties** are descriptive data of an object, for example the name of a component and the value of a signal.
- **Methods** are predefined functions for an object, or a task an object can perform that may return a value, for example a method for connecting behaviors.
- **Events** are predefined functions for an object that can listen and handle certain occurrences, for example when you run a simulation and a sensor is triggered by a component.

Python programs can be created as **add-ons** for Essentials.

- Store the Python files of an add-on in the My Commands folder in your Documents library.

## Path to My Commands folder

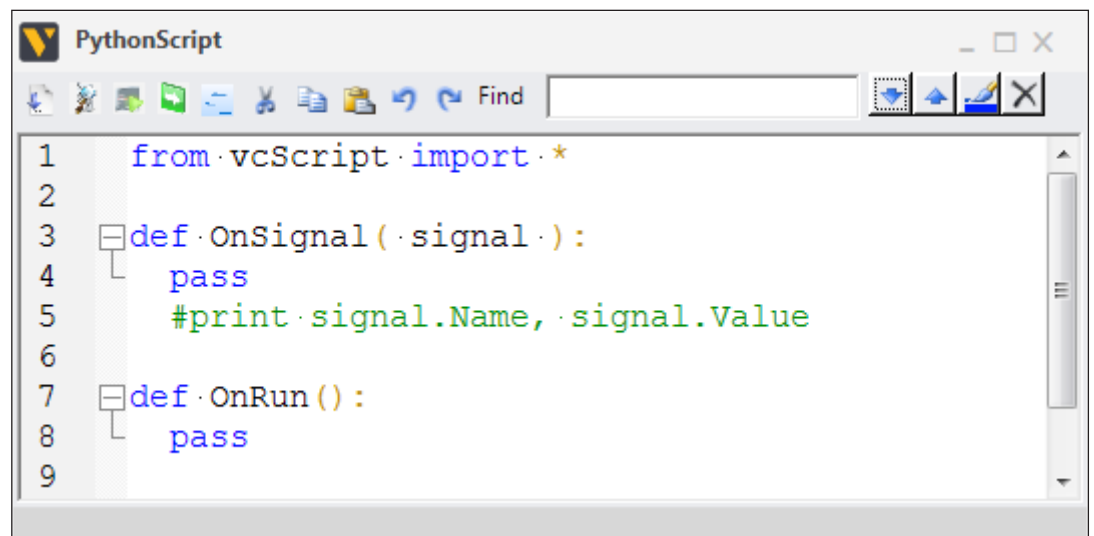
C:\Users\[Your User Name]\Documents\Visual Components\2014\My Commands

Other Python libraries, built-in functions, statements, and other supported tools can be used in Python Scripts.

## To trigger specific actions in a simulation

Simulation events can be used to execute instructions for behaviors in a component. For example, you can stop a path when a sensor is triggered by a component.

1. Create a new empty layout in the 3D world.
2. In the eCatalog panel, Collections view, click **My Models**, and then add **TrainingComponent** to the 3D world origin.
3. Click the **Modeling** tab, and then in the Component Graph panel, Component Node Tree pane, Search box, type **python**, and then double-click the found **PythonScript** to access its editor.
4. In the editor, comment the instruction in the OnSignal event that prints a signal name and value by adding a hash (#), which means any code that comes after the hash on the same line will not be executed.



```
PythonScript
1  from vcScript import *
2
3  def OnSignal ( signal ) :
4      pass
5      #print signal.Name, signal.Value
6
7  def OnRun ( ) :
8      pass
9
```

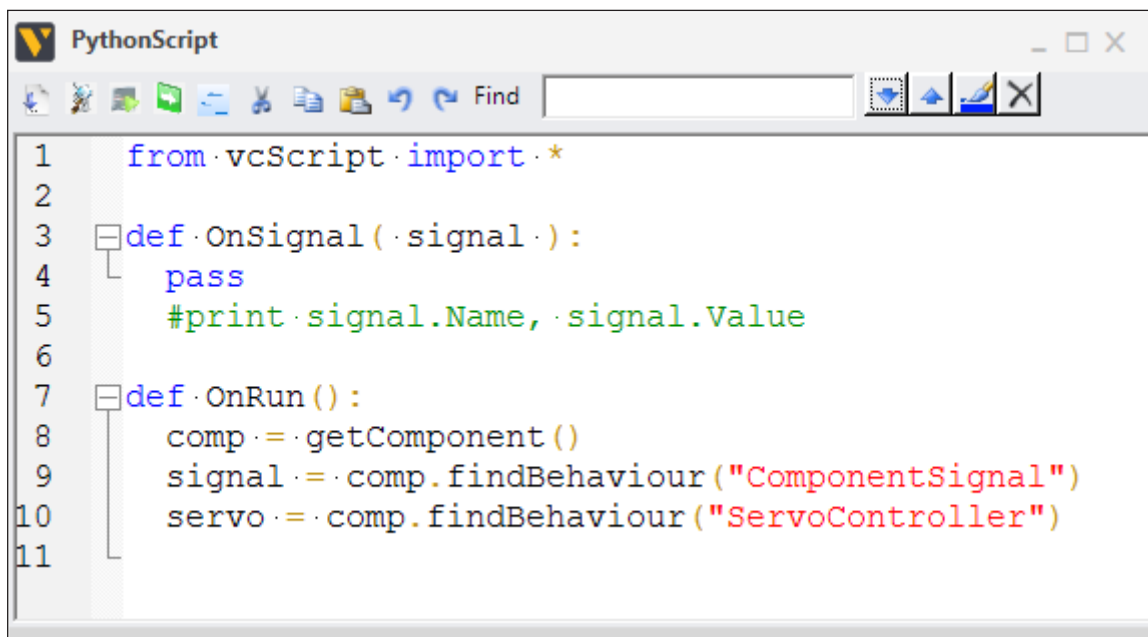
## Create variables using objects and methods

You can create variables (handles) to contain objects. A component is the sum of its parts and each part can be an object. That is, you can use methods with a component object to get specific parts of that component, for example a behavior.

1. In the editor, OnRun event, type the following code to create variables for TrainingComponent and its Component Signal and Servo Controller behaviors.

```
def OnRun():
    comp = GetComponent()
    signal = comp.findBehaviour("ComponentSignal")
    servo = comp.findBehaviour("ServoController")
```

**Explanation:** Every Python Script behavior is created with a snippet of code that imports everything from vcScript, including its methods. The GetComponent() method returns the component of the Python Script behavior. That is, the component containing that behavior. A component object is of type vcComponent and *inherits* the properties, methods and events from its root node, which is of type vcNode. Logically, a node contains behaviors, so a node object can be used to find existing or create new behaviors. This is why a component object can be used to find behaviors and other types of objects in the component.



The screenshot shows a window titled "PythonScript" with a toolbar and a code editor. The code editor contains the following Python code:

```
1  from vcScript import *
2
3  def OnSignal(.signal.):
4      pass
5      #print signal.Name, signal.Value
6
7  def OnRun():
8      comp = GetComponent()
9      signal = comp.findBehaviour("ComponentSignal")
10     servo = comp.findBehaviour("ServoController")
11
```

Python uses indentation to structure programs and form blocks of code. In the editor, a code block is formed when a continuing line of code is indented by two spaces. For example, the OnRun event is a code block currently consisting of three statements/instructions.

## Create test conditions for instructions

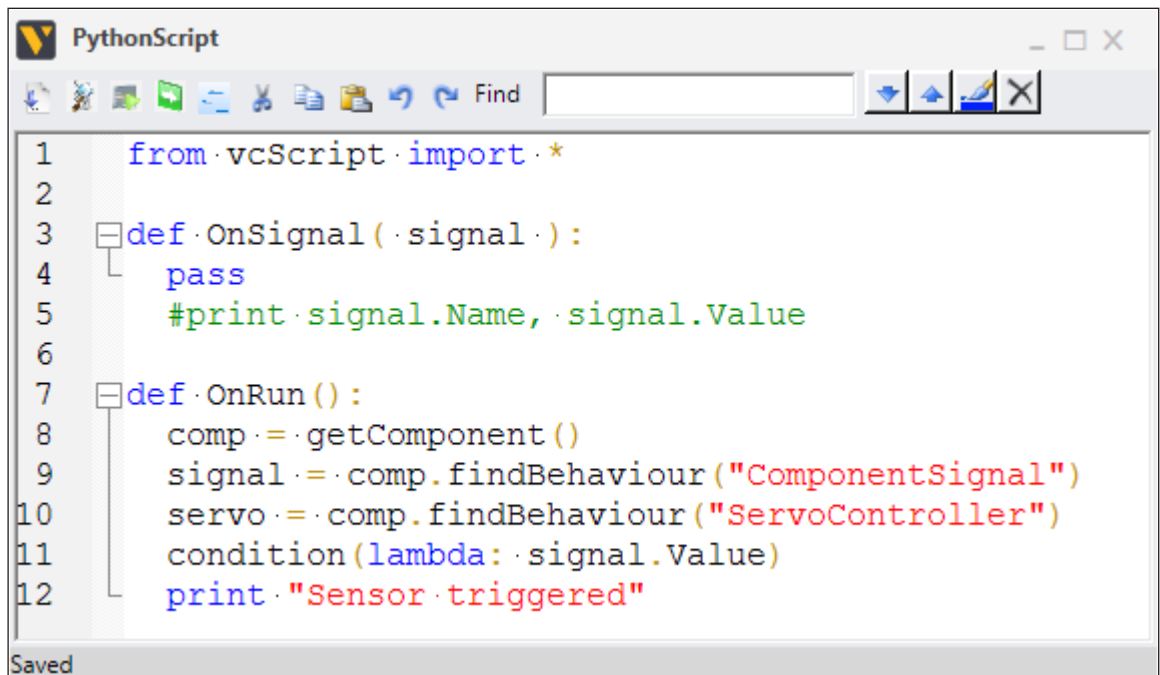
A **condition** allows you to wait for a trigger or expression to evaluate as True before executing other instructions.

1. In the OnRun event, use the condition() method in vcScript with a lambda operator to test the value of ComponentSignal, and then print feedback after the condition returns a True value.

```
def OnRun():
    comp = GetComponent()
    signal = comp.findBehaviour("ComponentSignal")
    servo = comp.findBehaviour("ServoController")
    condition(lambda: signal.Value)
    print "Sensor triggered"
```

2. Compile the code, and then run the simulation to verify feedback is printed in the Output panel when a cylinder triggers the path's sensor.

**Explanation:** A lambda operator is used to create a function that 1) tests an expression, and 2) returns a value. If the value of ComponentSignal is None, the lambda function returns a False value, which means the lambda will continue to execute until it returns a True value.



The screenshot shows a PythonScript window with the following code:

```
1 from vcScript import *
2
3 def OnSignal ( signal ):
4     pass
5     #print signal.Name, signal.Value
6
7 def OnRun ( ):
8     comp = GetComponent ( )
9     signal = comp.findBehaviour ( "ComponentSignal" )
10    servo = comp.findBehaviour ( "ServoController" )
11    condition ( lambda : signal.Value )
12    print "Sensor triggered"
```

The window title is "PythonScript" and it has a "Saved" status bar at the bottom.

## To turn off behaviors and move objects

Most behavior objects have an Enabled property that turns on/off tasks performed by a behavior. Some behaviors can perform specific tasks for their properties. For example, a Servo Controller can move assigned joints.

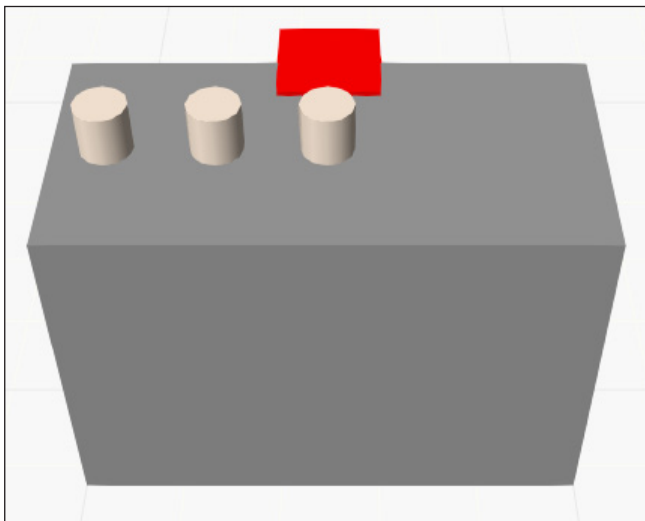
1. In the OnRun event, after the condition statement, create a variable to contain the OneWayPath of the component, and then set the Enabled property of the path to equal False.

```
...  
print "Sensor triggered"  
path = comp.findBehaviour("OneWayPath")  
path.Enabled = False
```

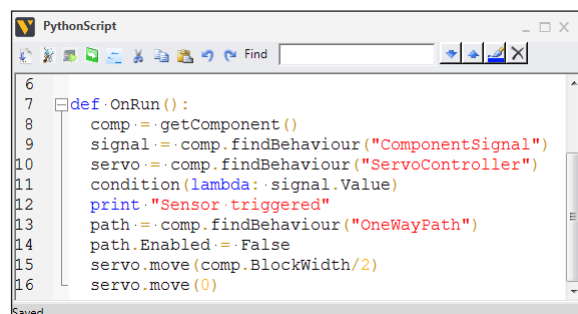
2. After the path is turned off, instruct the ServoController of the component to move its assigned joint to half the value of the BlockWidth property, and then move the joint back to zero value.

```
...  
path.Enabled = False  
servo.move(comp.BlockWidth/2)  
servo.move(0)
```

3. Compile the code, and then run the simulation to verify the path stops after its sensor is triggered and the Pusher node moves back and forth, and then reset the simulation.



**Explanation:** A component object can get the current value of its properties, which include the properties you create for a component using the Modeling tab. You can use the `getProperty()` method with a component object to return the property as a `vcProperty` object.

A screenshot of a PythonScript window. The code is as follows:

```
6  
7 def OnRun():  
8     comp = .GetComponent()  
9     signal = .comp.findBehaviour("ComponentSignal")  
10    servo = .comp.findBehaviour("ServoController")  
11    condition(lambda: signal.Value)  
12    print "Sensor triggered"  
13    path = .comp.findBehaviour("OneWayPath")  
14    path.Enabled = False  
15    servo.move(comp.BlockWidth/2)  
16    servo.move(0)
```

The window title is "PythonScript" and it has standard window controls. The code is line-numbered from 6 to 16.

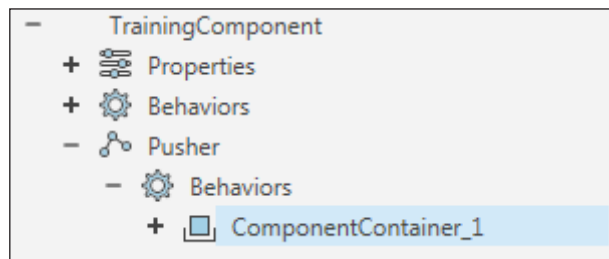
## To grab and move objects

Some behavior objects are containers for component objects. That allows you to grab and release components and relocate them during a simulation. This idea of containment is paramount to understanding component modeling.

1. On the Modeling tab, in the Component Graph panel, Component Node Tree pane, select the **Pusher** node.

**NOTE!** The editor of a Python Script will stay open unless you close the editor or Essentials.

2. On the Modeling tab, in the Behaviors group, click the **Behavior** arrow, and then in Material Flow, click **Container**.



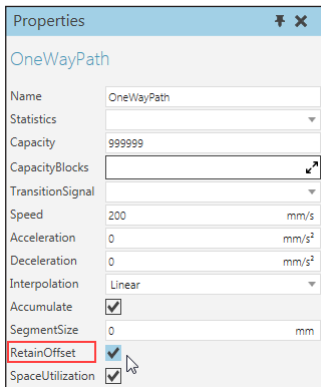
3. In the PythonScript editor, go to the OnRun event, and then immediately before the instructions for moving a joint assigned to ServoController, do all of the following:
  - Create two variables to contain ComponentContainer\_1 and the value of ComponentSignal.
  - Use the grab() method in vcContainer with ComponentContainer\_1 to grab a cylinder from the path in the root node to the container in the Pusher node.
4. In the OnRun event, immediately before the instruction for moving a joint assigned to ServoController to a zero value, do all of the following:
  - Grab a cylinder from the container in the Pusher node to the path in the root node.
  - Set the Enabled property of the path to equal a True value.

```
...
path.Enabled = False
push_container = comp.findBehaviour("ComponentContainer_1")
part = signal.Value
push_container.grab(part)
servo.move(comp.BlockWidth/2)
path.grab(part)
servo.move(0)
path.Enabled = True
```

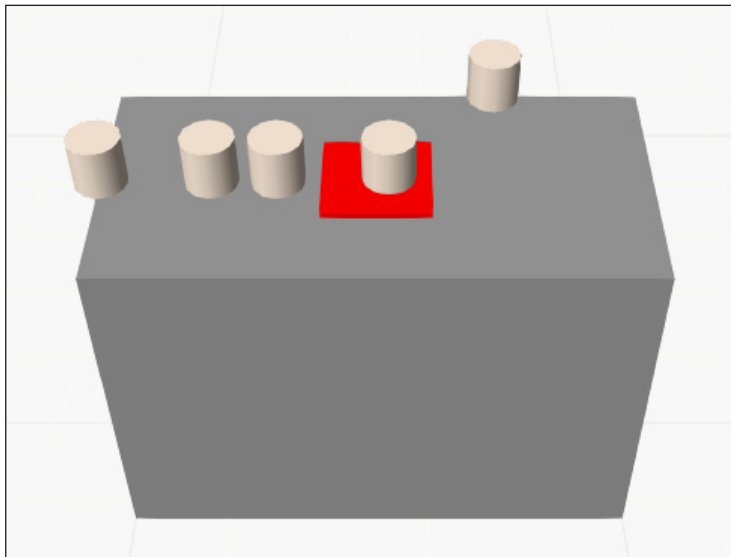
5. Compile the code.



6. Access the properties of the **OneWayPath** behavior in the root node.
7. In the Properties panel, select the **RetainOffset** check box.



8. Run the simulation to verify a cylinder is pushed to the side of the path and retains its position when moved forward on the path, and then reset the simulation.



**Explanation:** An instruction to move a joint assigned to a servo is not completed until the referenced joint reaches its given value with some exceptions. A component can retain its position while moving in the direction of the path.

```

PythonScript
7 def OnRun () :
8     comp := GetComponent ()
9     signal := comp.findBehaviour ("ComponentSignal")
10    servo := comp.findBehaviour ("ServoController")
11    condition (lambda: .signal.Value)
12    print "Sensor triggered"
13    path := comp.findBehaviour ("OneWayPath")
14    path.Enabled := False
15    push_container := comp.findBehaviour ("ComponentContainer_1")
16    part := signal.Value
17    push_container.grab (part)
18    servo.move (comp.BlockWidth/2)
19    path.grab (part)
20    servo.move (0)
21    path.Enabled := True

```

## To clean up your code

We recommend formatting your Python Scripts in a way that is easy to edit and parse through by yourself and others. Generally, it is good to keep variables and instructions separate, use whitespace to distinguish blocks of code, and precede an important instruction with a comment.

1. Access the **PythonScript** editor, and then clean up the code to resemble the following code example.

```
from vcScript import *

def OnSignal( signal ):
    pass
    #print signal.Name, signal.Value

def OnRun():
    comp = GetComponent()
    signal = comp.findBehaviour("ComponentSignal")
    servo = comp.findBehaviour("ServoController")
    path = comp.findBehaviour("OneWayPath")
    push_container = comp.findBehaviour("ComponentContainer_1")

    ##wait for sensor, get component at sensor
    condition(lambda: signal.Value)
    print "Sensor triggered"
    path.Enabled = False
    part = signal.Value

    ##grab part, move to new position on path
    push_container.grab(part)
    servo.move(comp.BlockWidth/2)
    path.grab(part)
    servo.move(0)
    path.Enabled = True
```

2. Save **TrainingComponent**.

# Project

Using the TrainingComponent you created in Module 5, route the cylinder grabbed by the Pusher node to a new path in the TrainingComponent.

The following steps are a how-to-guide for completing the project.

1. Select the **root node** of TrainingComponent.
2. Create a **Transform** feature with an Expression of **Ty(BlockWidth/2)**.
3. Make copies of **MidFrame** and **EndFrame**, and then nest those **copies** in Transform\_1 in way that the location of each copy is transformed by the operation.
4. Create a **One Way Path** behavior that has MidFrame\_1 and EndFrame\_1 as waypoints.
5. Create a **One to One Interface** located at EndFrame\_1 that flows components from the Output port of OneWayPath\_1.
6. Access the **PythonScript** editor.
7. Create a new variable to contain OneWayPath\_1, and then edit the code to move a cylinder at the sensor to a new path.

```
from vcScript import *

def OnSignal( signal ):
    pass
    #print signal.Name, signal.Value

def OnRun():
    comp = GetComponent()
    signal = comp.findBehaviour("ComponentSignal")
    servo = comp.findBehaviour("ServoController")
    path = comp.findBehaviour("OneWayPath")
    push_path = comp.findBehaviour("OneWayPath_1")
    push_container = comp.findBehaviour("ComponentContainer_1")

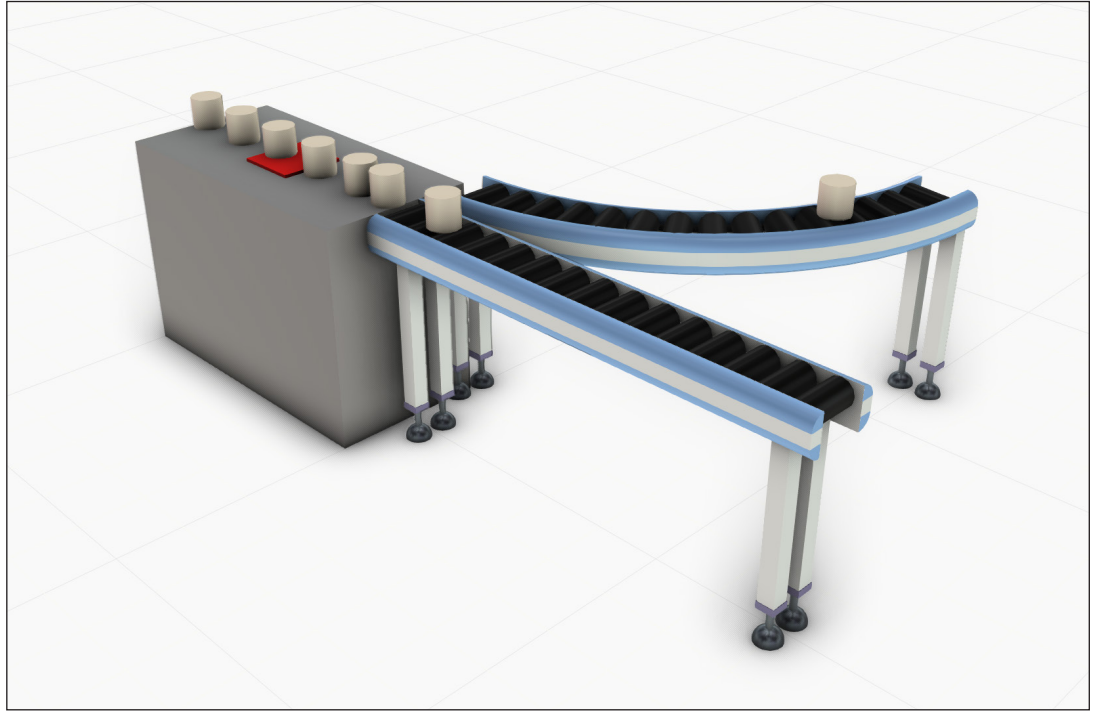
    ##wait for sensor, get component at sensor
    condition(lambda: signal.Value)
    print "Sensor triggered"
    path.Enabled = False
    part = signal.Value

    ##grab part, move to new position on path
    push_container.grab(part)
    servo.move(comp.BlockWidth/2)
    push_path.grab(part)
    servo.move(0)
    path.Enabled = True
```

8. Add and connect a **Conveyor** and **Curve Conveyor** to the TrainingComponent.

**TIP!** Resize and edit the conveyors before you plug them into the TrainingComponent.

9. Run the simulation to verify each conveyor receives a cylinder, and then reset the simulation.



10. Save **TrainingComponent**, and then save the layout as **Module5.vcmx** in the My Models folder in your Documents library.

## Review

In this module you learned about Python and how to:

- Create variables and assign values.
- Use objects with methods and properties.
- Create test conditions for instructions.
- Disable behaviors to create suitable conditions for instructions.
- Grab and move components into different containers.

If you would like to learn how to model components using imported geometry, access tutorials and other documents at <http://forum.visualcomponents.com>.

## Conclusion

Over the course of five modules you learned how to run simulations, build layouts, create robot programs, and model components. You now have a skillset in 3D modeling and simulation that can be applied in many sectors and fields of study. Furthermore, you have become proficient with Visual Components products that make it easy to create solutions and test possibilities. See for yourself what you can learn and develop further by using Essentials.

For more information, visit our website at [www.visualcomponents.com](http://www.visualcomponents.com).