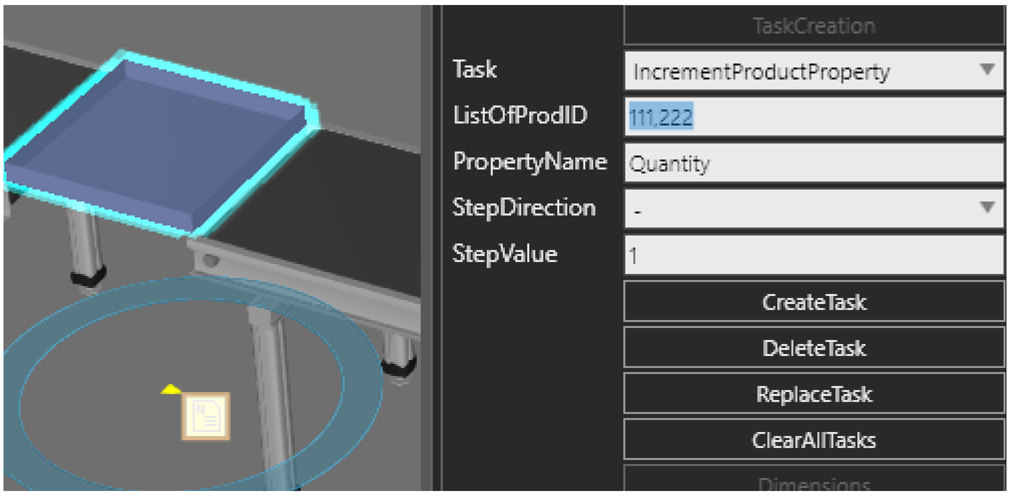
Increment Product Properties

Visual Components 4.1 | Version: December 3, 2018



Task: IncrementProductProperty
ListOfProdID: 111,222
PropertyName: Quantity
StepDirection: -
StepValue: 1

The Visual Components Works library allows you to simulate simple and complex processes. One aspect of a process might be to increment or decrement a property value. For example, you might want to track the quantity of a pallet or box as it moves through different stages in a production line. Because of its flexibility, the Works library allows you to create new tasks for your own solutions.

In this tutorial, you will learn how to:

- Create component properties and use them as task arguments.
- List a new task along with its arguments in the GUI.
- Use Python API to filter products by ProdID value, get a property value and increment or decrement its value.

This tutorial assumes you have completed the "Create a New Task in Works" tutorial and have a basic understanding of Visual Components Python API.

Support
support@visualcomponents.com

Forum
forum.visualcomponents.com

Learning Pathway

Before continuing with this tutorial, make sure you have a basic understanding of Python programming language and Visual Components Python API.

Here is a simple learning pathway for Python API.

Component Scripting

Introduction to using a Python Script behavior.

http://academy.visualcomponents.com/lessons/component-scripting/

Create a Distribution Property

Shows how to use a real distribution property and create one using GUI and Python. It also covers some info about where in Help to know about writing expressions.

http://academy.visualcomponents.com/lessons/create-a-distribution-property/

Control Servos

Introduction to using Python Script to control servo moving joints.

http://academy.visualcomponents.com/lessons/control-servos/

Component States

Introduction to how you can change the state of a component using Python Script and Statistics behavior.

http://academy.visualcomponents.com/lessons/component-states/

Conditions and Signals

Introduction to using triggers, conditions and signals in Python Script.

http://academy.visualcomponents.com/lessons/conditions-and-signals/

Event Handlers

Introduction to using events in Python Script.

http://academy.visualcomponents.com/lessons/event-handlers/

Create a Button Property

Shows you how to create a button and then use its OnChanged event in Python Script.

http://academy.visualcomponents.com/lessons/create-a-button-property/

Course - Python Robotics

You should know enough Python at this point in the pathway to take this course.

http://academy.visualcomponents.com/courses/python-robotics-programming-a-robot-with-python/

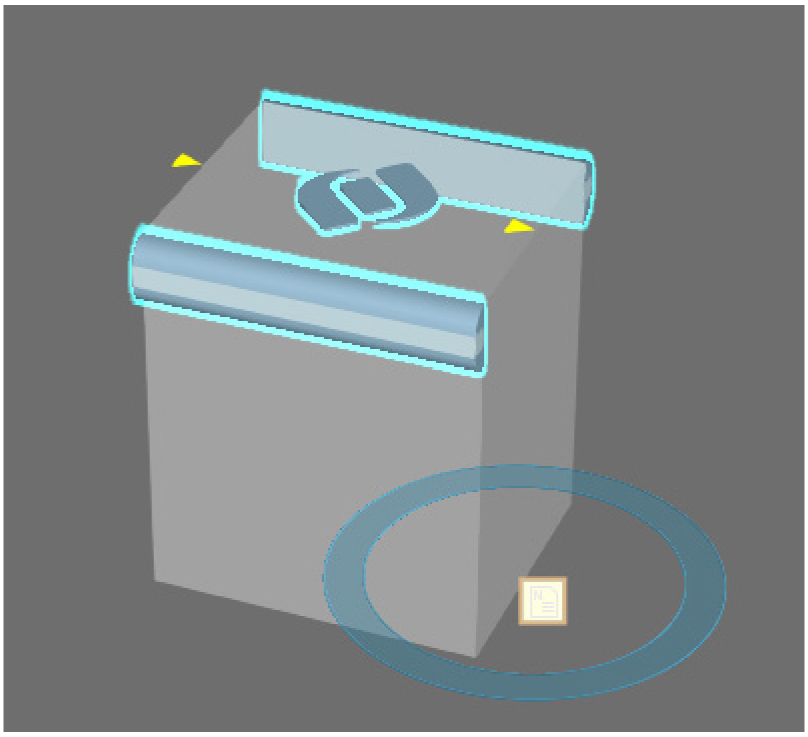
Add Advanced Feeder

An Advanced Feeder allows you to create up to three different components during a simulation. You have the option of creating components in batches or somewhat randomly using a distribution.

1. Clear the 3D world.

2. In the eCatalog panel, Collections view, expand **Models by Manufacturer**, and then expand **Visual Components**.

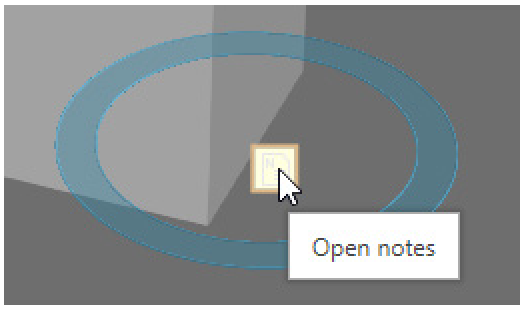
3. Click **Feeders**, and then add an **Advanced Feeder** to the 3D world.



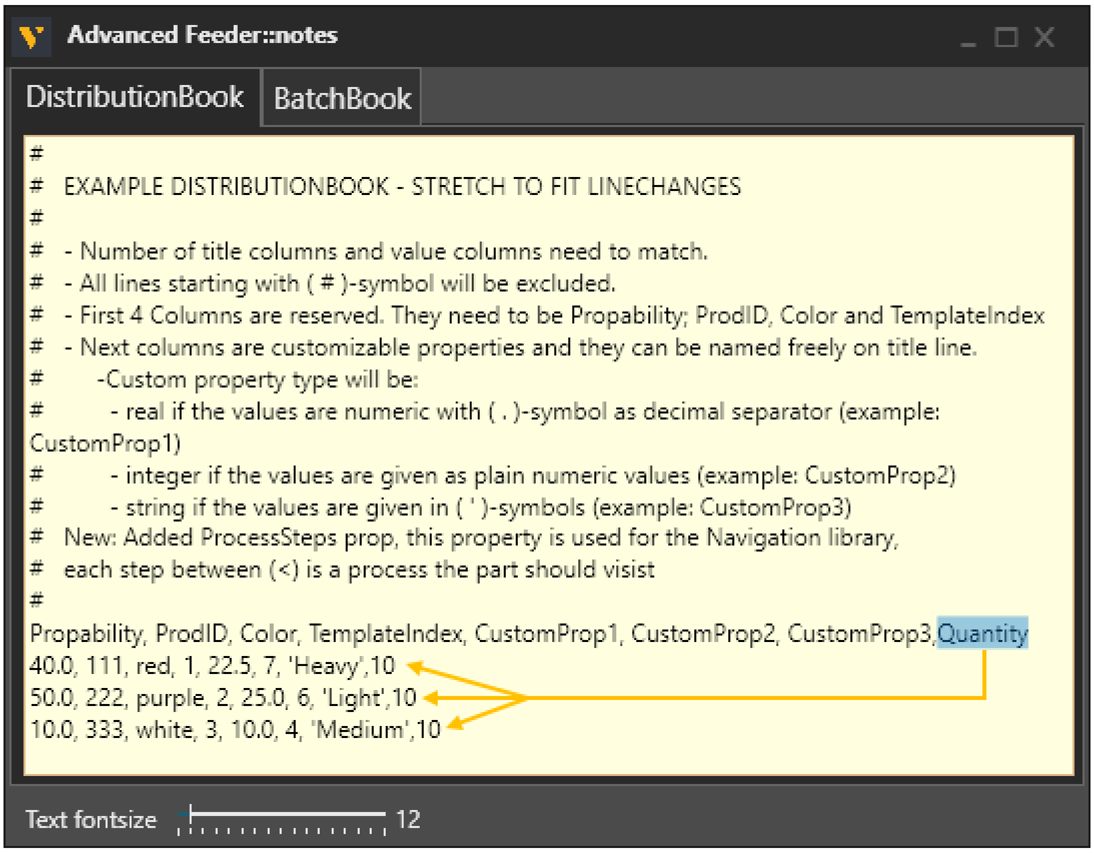
Add Property to Distribution Entries

By default, an Advanced Feeder creates components using a distribution of entries. Each entry defines a set of properties for a template component and the probability of creating that component.

1. In the 3D world, click the **Note** of the feeder, and then select the **DistributionBook** tab.



2. In the note editor, add a **Quantity** property, and then assign each entry a quantity of **10**.

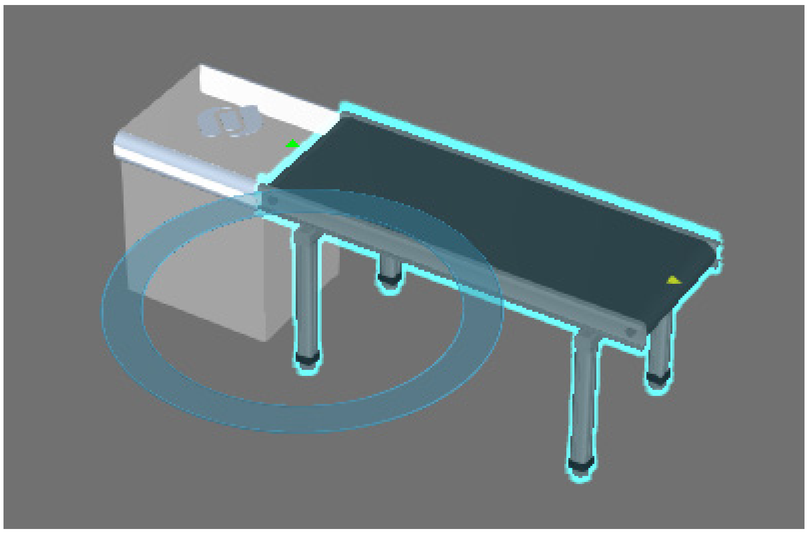


```
#
#  EXAMPLE DISTRIBUTIONBOOK - STRETCH TO FIT LINECHANGES
#
#  - Number of title columns and value columns need to match.
#  - All lines starting with ( # )-symbol will be excluded.
#  - First 4 Columns are reserved. They need to be Propability; ProdID, Color and TemplateIndex
#  - Next columns are customizable properties and they can be named freely on title line.
#       -Custom property type will be:
#         - real if the values are numeric with ( . )-symbol as decimal separator (example:
CustomProp1)
#         - integer if the values are given as plain numeric values (example: CustomProp2)
#         - string if the values are given in ( ' )-symbols (example: CustomProp3)
#  New: Added ProcessSteps prop, this property is used for the Navigation library,
#  each step between (<) is a process the part should visist
#
Propability, ProdID, Color, TemplateIndex, CustomProp1, CustomProp2, CustomProp3,Quantity
40.0, 111, red, 1, 22.5, 7, 'Heavy',10
50.0, 222, purple, 2, 25.0, 6, 'Light',10
10.0, 333, white, 3, 10.0, 4, 'Medium',10
```

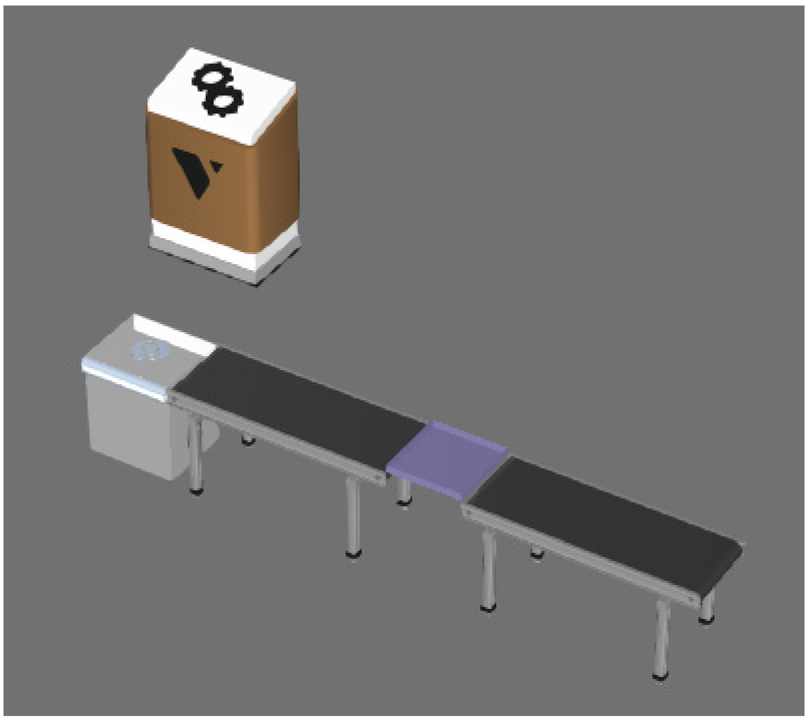
This means a component made by the feeder during a simulation will have a Quantity property of type integer with a value of 10.

Build Conveyor Line

1. In the eCatalog panel, under Visual Components, click **Conveyors (New)**.
2. Add and connect a **Conveyor** to the feeder in the 3D world.

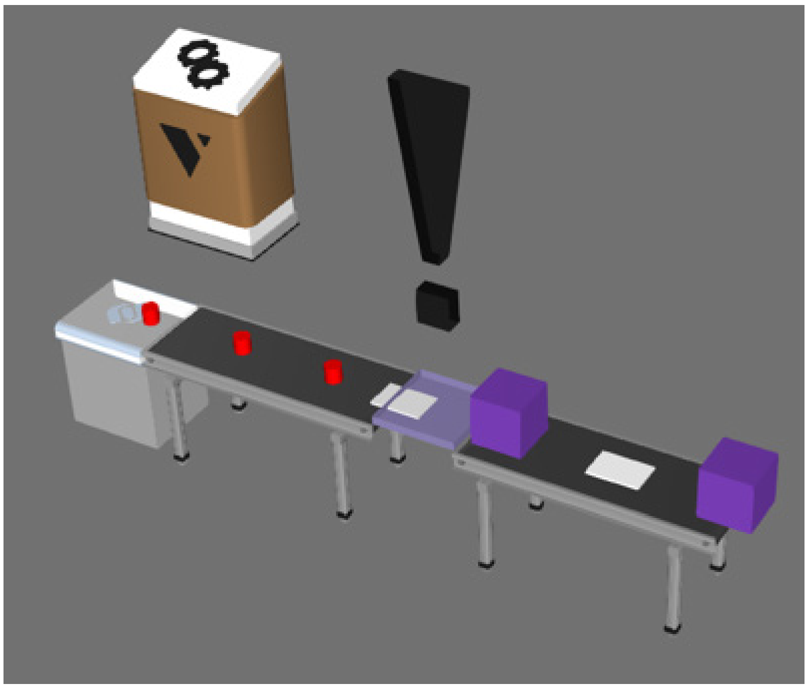


3. In the eCatalog panel, under Visual Components, click **Works**.
4. Add a **Works Task Controller** to the 3D world.
5. Add and connect a **Works Process** to the other end of the conveyor in the 3D world.
6. Copy and paste the **conveyor**, and then connect it to the other end of the Works Process component.

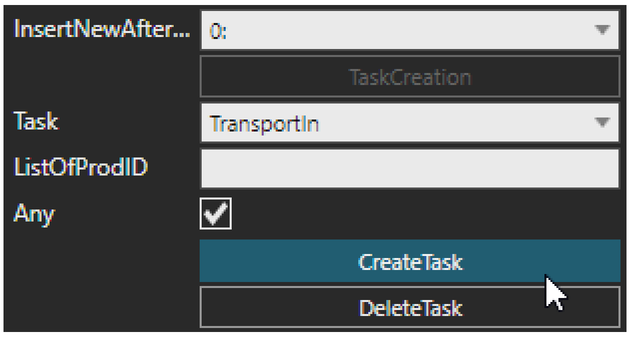


Transport In and Out Parts

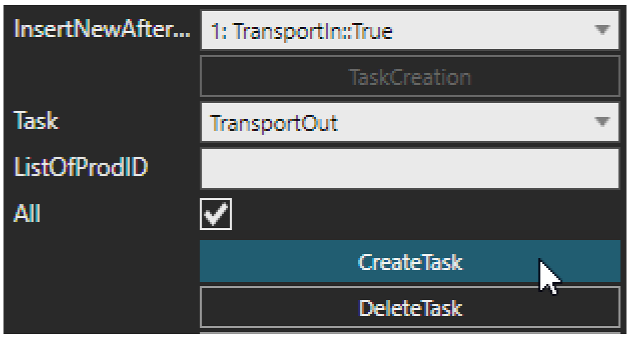
1. Run the simulation, verify parts flow in and out of the Works Process without stopping, and then reset the simulation.



2. In the 3D world, select the **Works Process**, and then in the Component Properties panel, add a **TransportIn** task, and then add a **TransportOut** task.



InsertNewAfter...	0:
	TaskCreation
Task	TransportIn
ListOfProdID	
Any	✓
	CreateTask
	DeleteTask



InsertNewAfter...	1: TransportIn::True
	TaskCreation
Task	TransportOut
ListOfProdID	
All	✓
	CreateTask
	DeleteTask

This means parts will move in and out of the container of the Works Process component. Remember that a lot of tasks only affect contained components.

Create the Task - First Iteration

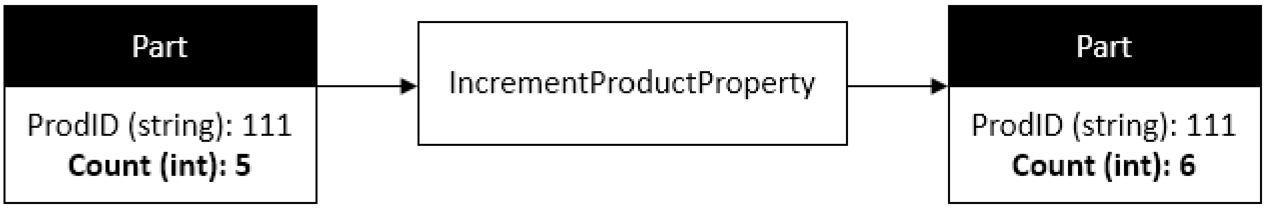
If you have access to the Modeling tab, you can create new types of tasks in a Works Process component. The Create a New Task in Works lesson in the Visual Components Academy covers the basics of adding new tasks.

Goal: Increment the value of an integer property in a product contained in a Works Process component

Specifications:

- Filter parts using a single ProdID value.
- Identify property by name.
- Option to increment or decrement the property value
- Option to define step size.

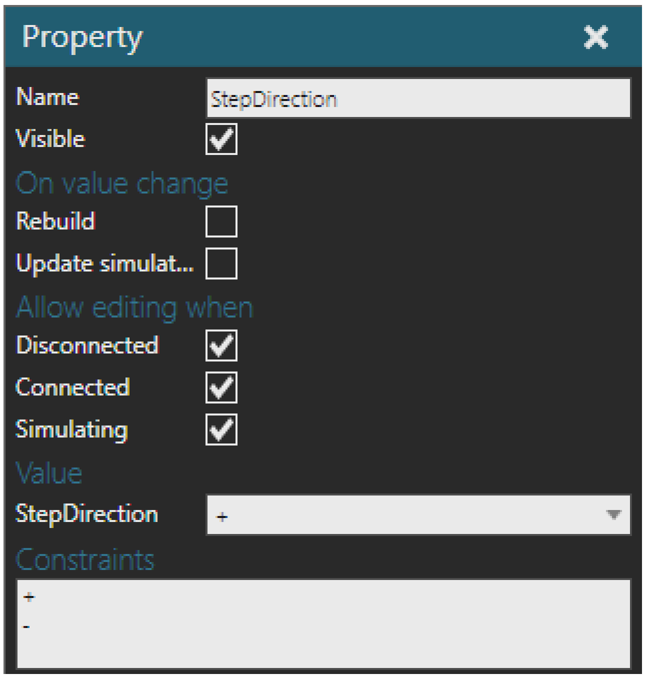
Diagram:



Make Task Arguments

A task argument is a component property listed as an option when creating a task.

1. In the 3D world, select the **Works Process** component.
2. Add a **String** property, name it **StepDirection**, clear the **Rebuild** checkbox, and then add two constraint values, "+" and "-" with the PLUS sign being the default value.

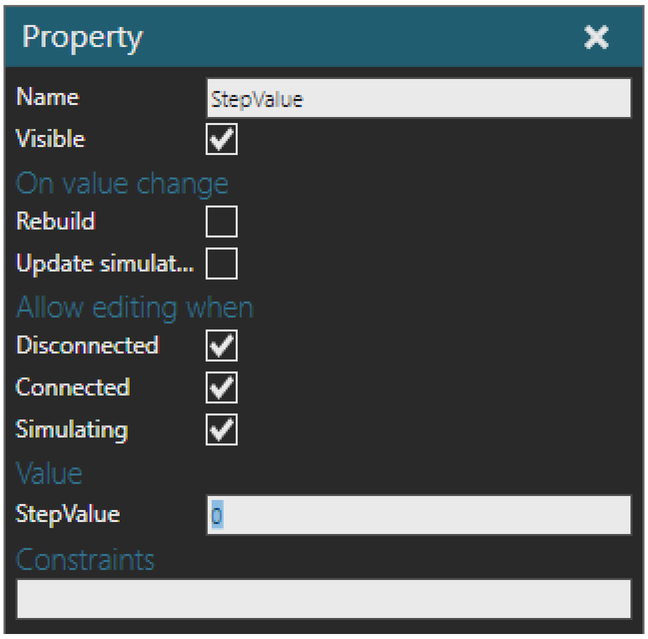


Property

Name	StepDirection
Visible	✓
On value change	
Rebuild	☐
Update simulat...	☐
Allow editing when	
Disconnected	✓
Connected	✓
Simulating	✓
Value	
StepDirection	+
Constraints	

```
+
-
```

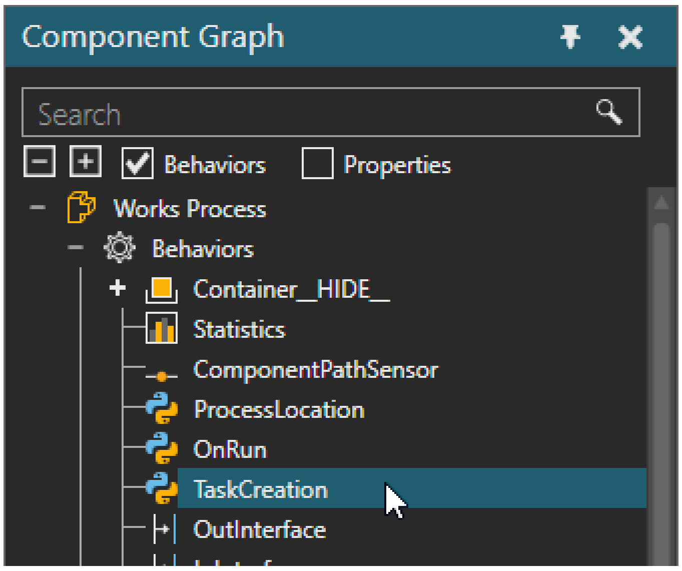
3. Add an **Integer** property, name it **StepValue**, and then clear the **Rebuild** checkbox. The default value should be zero.



Property

Name	StepValue
Visible	✓
On value change	
Rebuild	☐
Update simulat...	☐
Allow editing when	
Disconnected	✓
Connected	✓
Simulating	✓
Value	
StepValue	0
Constraints	

4. In the Works Process, access the **TaskCreation** script.



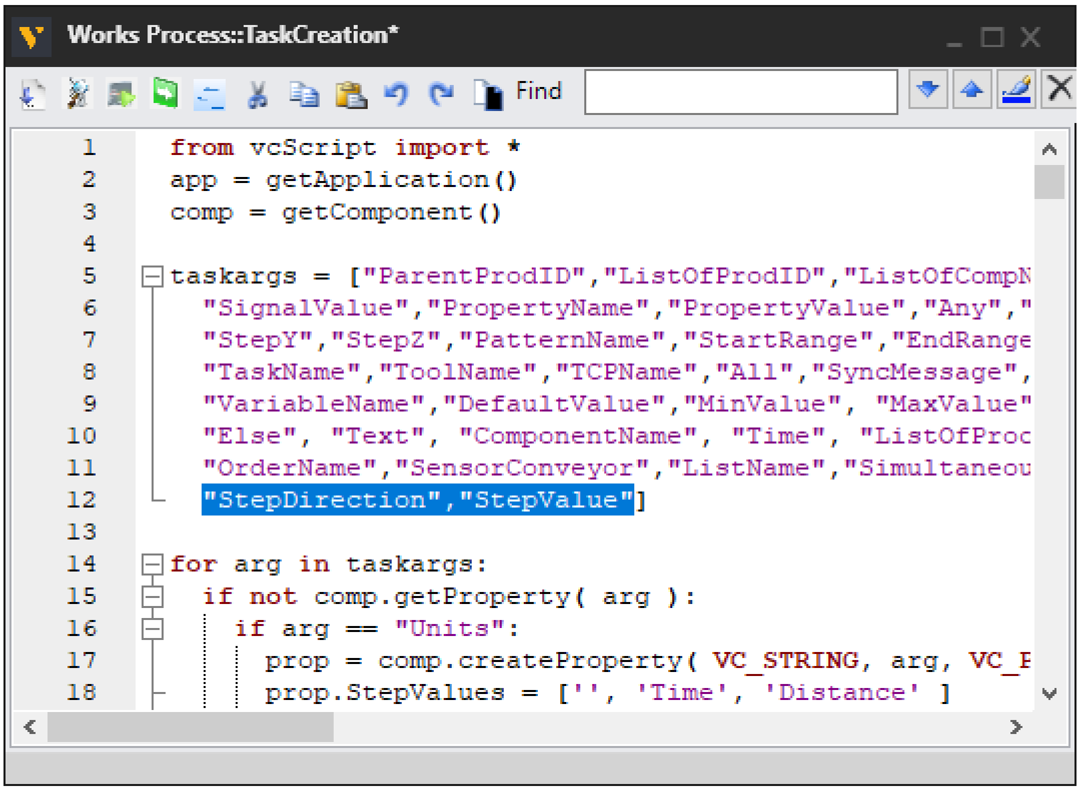
Component Graph

Search

Behaviors   Properties

- Works Process
  - Behaviors
    - Container__HIDE__
    - Statistics
    - ComponentPathSensor
    - ProcessLocation
    - OnRun
    - TaskCreation
    - OutInterface

5. In the script editor, go to line 5, and then add **StepDirection** and **StepValue** to the end of the **taskargs** list. This means these two arguments will be listed after other arguments when you select a task in the GUI.



Works Process::TaskCreation*

```
 1  from vcScript import *
 2  app = getApplication()
 3  comp = getComponent()
 4
 5  taskargs = ["ParentProdID","ListOfProdID","ListOfCompN
 6    "SignalValue","PropertyName","PropertyValue","Any","
 7    "StepY","StepZ","PatternName","StartRange","EndRange
 8    "TaskName","ToolName","TCPName","All","SyncMessage",
 9    "VariableName","DefaultValue","MinValue", "MaxValue"
10    "Else", "Text", "ComponentName", "Time", "ListOfProc
11    "OrderName","SensorConveyor","ListName","Simultaneou
12    "StepDirection","StepValue"]
13
14  for arg in taskargs:
15    if not comp.getProperty( arg ):
16      if arg == "Units":
17        prop = comp.createProperty( VC_STRING, arg, VC_E
18        prop.StepValues = ['', 'Time', 'Distance' ]
```

Note: In Python, a backslash "\" is sometimes used as a line break in a list of values.

Select Task Arguments

A task must define which arguments/component properties are associated with it. A Works Process component will automatically show/hide task arguments based on a selected task in the GUI.

1. In the script editor, find the **showhide()** function.
2. Add this **elif** statement.

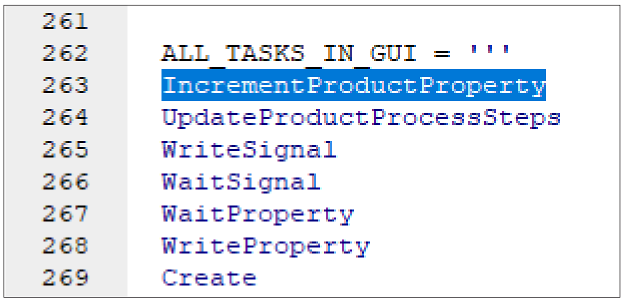
```
elif task == "IncrementPropertyProperty": sel = ["SingleProdID","PropertyName","StepDirection","StepValue"]
```

This means a task named IncrementProductProperty will have four arguments.

List Task as Option

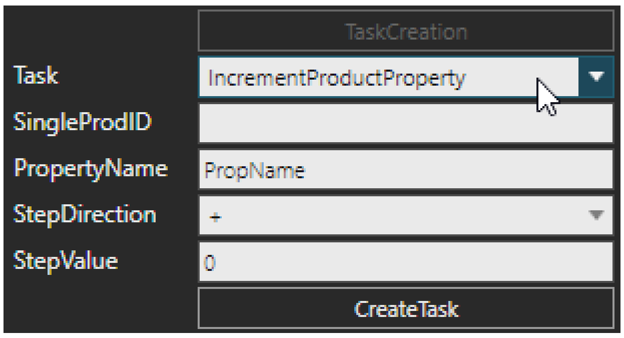
A task must be added to a dictionary that lists available tasks in a Works Process component.

1. In the script editor, find the **ALL_TASKS_IN_GUI** variable.
2. Add **IncrementProductProperty** as its own line in the string.



```

ALL_TASKS_IN_GUI = '''
IncrementProductProperty
UpdateProductProcessSteps
WriteSignal
WaitSignal
WaitProperty
WriteProperty
Create
```

3. Save the layout, and then test if IncrementProductProperty and its arguments are listed in the Component Properties panel.

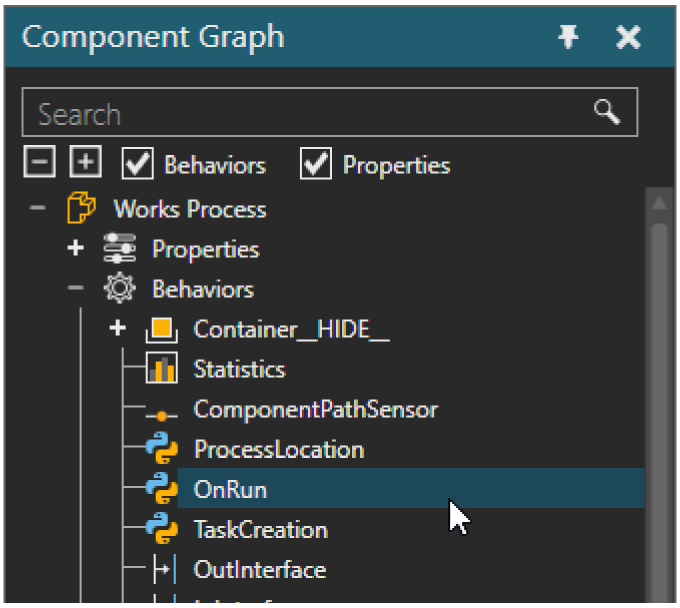


	TaskCreation
Task	IncrementProductProperty
SingleProdID	
PropertyName	PropName
StepDirection	+
StepValue	0
	CreateTask

Define Task Function

You need to define what the task will do when it is executed during a simulation.

1. In the Works Process, access the **OnRun** script.



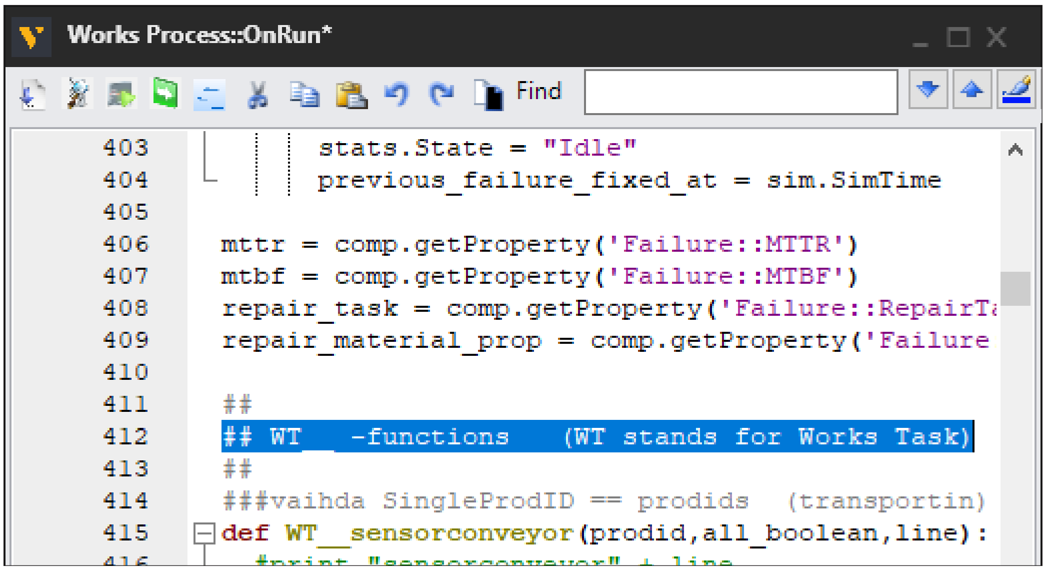
2. In the script editor, find the **call_TW function()** function.

3. Add this **elif** statement.

```
elif task == "IncrementProductProperty": WT__IncrementProductProperty(*data)
```

   This means a task named IncrementProductProperty will call a function in the script named WT__IncrementProductProperty and pass a set of arguments. Be aware that in this case the task arguments are passed in the order they appear in the GUI.

4. In the script editor, find **## WT__ -functions** which is a section for organizing the functions called by tasks.

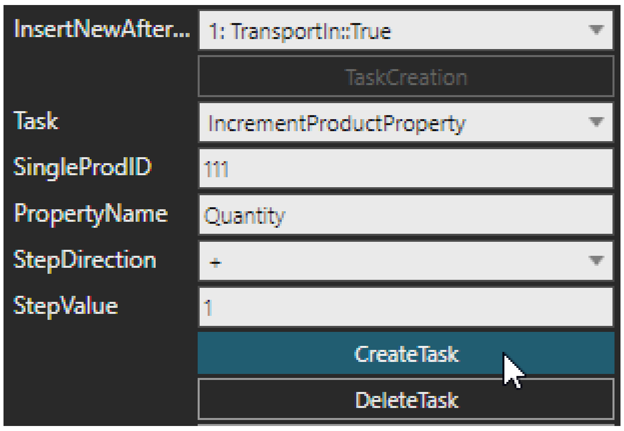


```
        stats.State = "Idle"
        previous_failure_fixed_at = sim.SimTime

  mttr = comp.getProperty('Failure::MTTR')
  mtbf = comp.getProperty('Failure::MTBF')
  repair_task = comp.getProperty('Failure::RepairTa
  repair_material_prop = comp.getProperty('Failure

  ##
  ## WT__-functions   (WT stands for Works Task)
  ##
  ###vaihda SingleProdID == prodids  (transportin)
def WT__sensorconveyor(prodid,all_boolean,line):
    #print "sensorconveyor" + line
```

5. Add the following code.

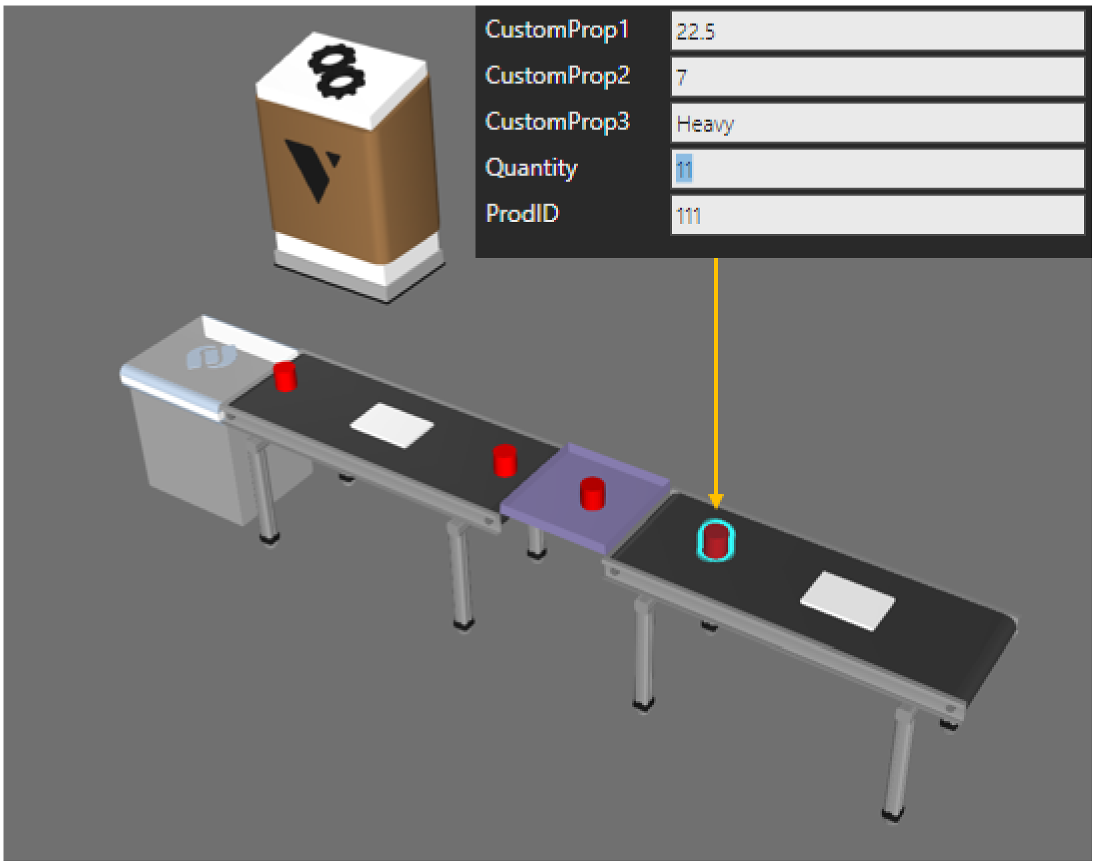
```
def WT__IncrementProductProperty(prodId,propName,direction,step):
  #get part in Works Process that matches the given ProdID
  parts = [x for x in cont1.Components if x.ProdID == prodId]

  #filter parts that have property matching the given name
  parts = [x for x in parts if x.getProperty(propName)]

  #by default, task args are passed to function as strings, so convert step
  step = int(step)

  #change value based on direction
  for c in parts:
    p = c.getProperty(propName)
    if direction == "+":
      p.Value += step
    else:
      p.Value -= step
```

6. Save the layout, and then insert an **IncrementProductProperty** task before the TransportOut task in the Works Process. Keep it simple: Increment the Quantity property of ProdID 111 by one.



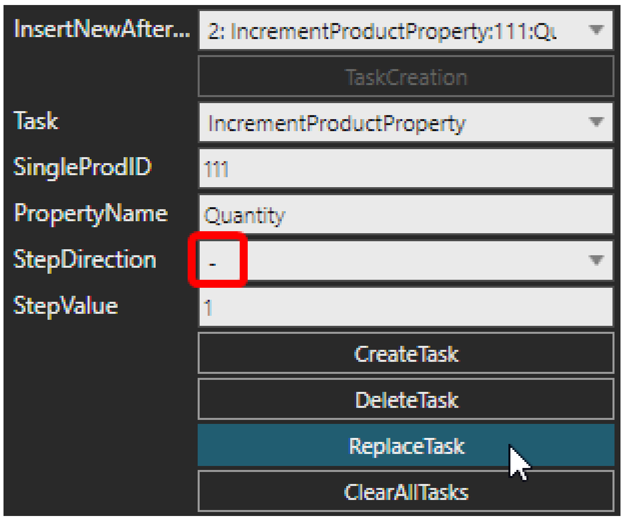
InsertNewAfter...	1: TransportIn::True
	TaskCreation
Task	IncrementProductProperty
SingleProdID	111
PropertyName	Quantity
StepDirection	+
StepValue	1

7. Run the simulation, verify the Quantity of a part with a ProdID of 111 is changed from 10 to 11, and then reset the simulation.



CustomProp1	22.5
CustomProp2	7
CustomProp3	Heavy
Quantity	11
ProdID	111

8. Update the IncrementProductProperty task to decrement the value by one.



InsertNewAfter...	2: IncrementProductProperty:111:Qu
	TaskCreation
Task	IncrementProductProperty
SingleProdID	111
PropertyName	Quantity
StepDirection	-
StepValue	1

9. Run the simulation, verify the Quantity of a part with a ProdID of 111 changes from 10 to 9, and then reset the simulation.

Extend the Task - Second Iteration

The first iteration of the task only affects one ProdID at a time. You can extend the task to support a list of ProdID values.

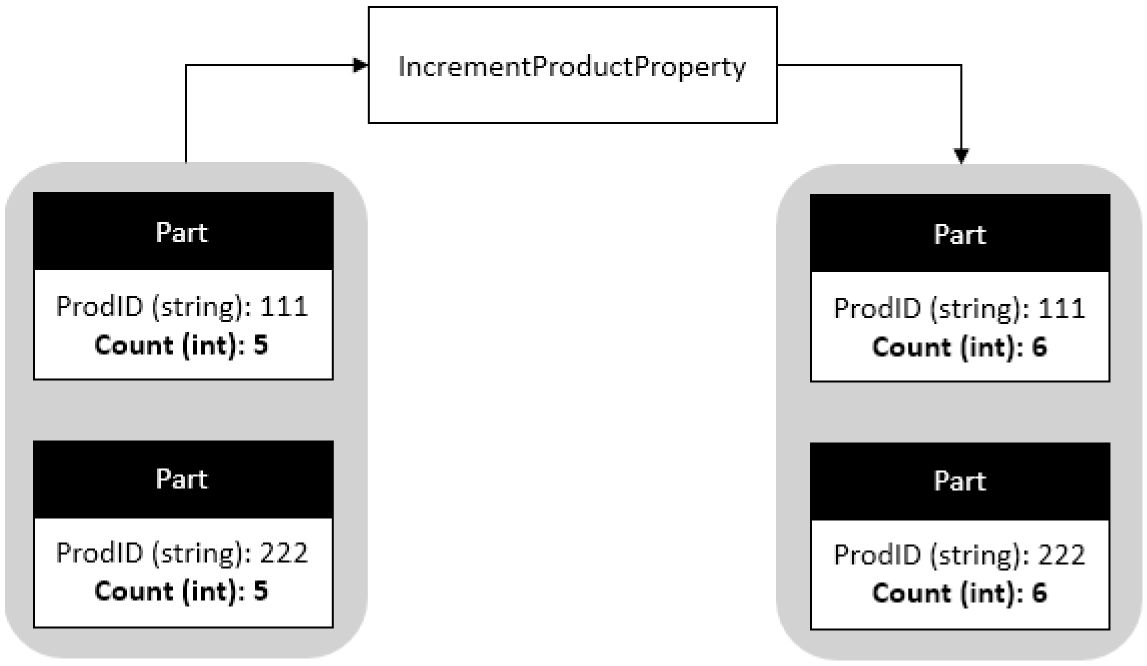
Goal:

Increment the value of an integer property in multiple products contained in a Works Process component

Specifications:

- Filter parts using a list of ProdID values.

Diagram:



Select Different Task Arguments

1. In the Works Process, access the **TaskCreation** script.

1. In the script editor, find the **showhide()** function.

2. Modify the **elif** statement for IncrementProductProperty to use ListOfProdID instead of SingleProdID.

   ```
   elif task == "IncrementProductProperty": sel = ["ListOfProdID","PropertyName","StepDirection","StepValue"]
   ```

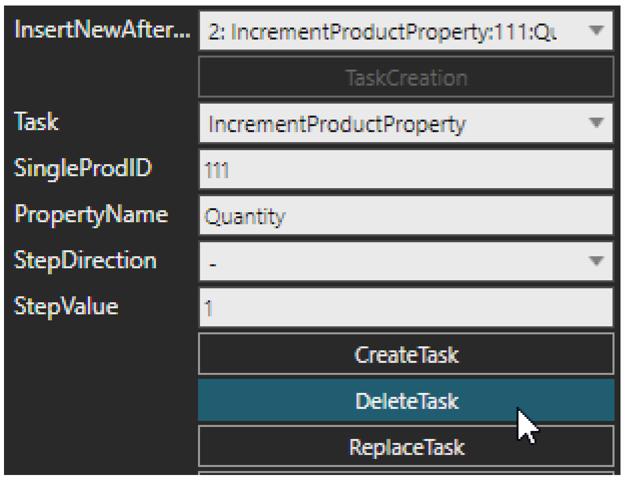
Modify Task Function

1. In the Works Process, access the **OnRun** script.

2. In the script editor, find **WT_IncrementProductProperty()** function, and then replace it with the following code.

   ```
   def WT__IncrementProductProperty(prodList,propName,direction,step):
     #get list of given ProdIDs separated by comma and remove whitespace
     prodList = [x.strip() for x in prodList.split(",")]

     #get part in Works Process that matches ProdID in list
     parts = [x for x in cont1.Components if x.ProdID in prodList]

     #filter parts that have property matching the given name
     parts = [x for x in parts if x.getProperty(propName)]

     #by default, task args are passed to function as strings, so convert step
     step = int(step)

     #change value based on direction
     for c in parts:
       p = c.getProperty(propName)
       if direction == "+":
         p.Value += step
       else:
         p.Value -= step
   ```

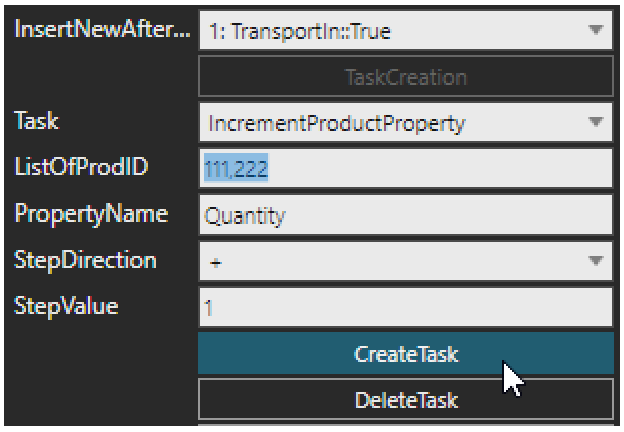
3. Save the layout, and then delete the existing **IncrementProductProperty** task in the process.



InsertNewAfter...	2: IncrementProductProperty:111:Qu
	TaskCreation
Task	IncrementProductProperty
SingleProdID	111
PropertyName	Quantity
StepDirection	-
StepValue	1

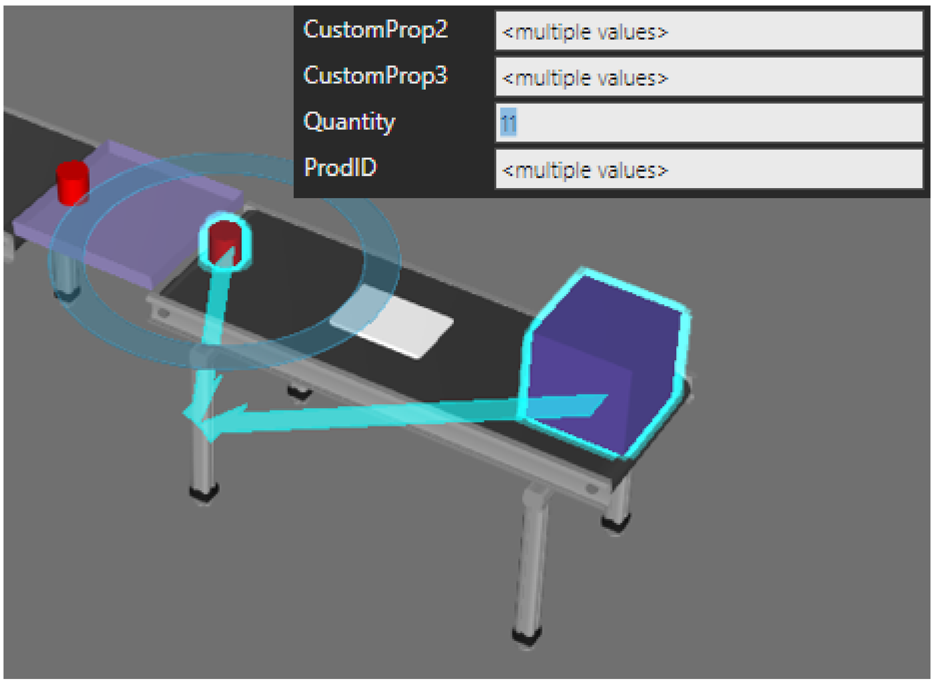
This allows the Works Process to update to the latest version of the task in the GUI.

4. Insert an **IncrementProductProperty** task before the TransportOut task in the Works Process. The task should increment the Quantity property of ProdID 111 or 222 by one.



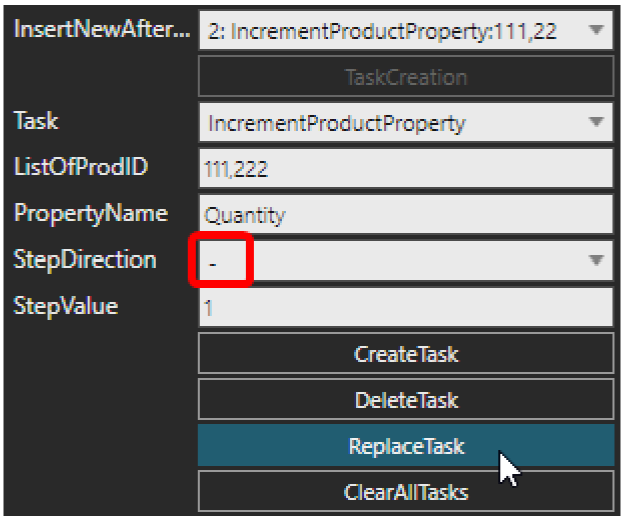
InsertNewAfter...	1: TransportIn::True
	TaskCreation
Task	IncrementProductProperty
ListOfProdID	111,222
PropertyName	Quantity
StepDirection	+
StepValue	1

5. Run the simulation, verify the Quantity of a part with ProdID of 111 or 222 is changed from 10 to 11, and then reset the simulation.



CustomProp2	<multiple values>
CustomProp3	<multiple values>
Quantity	11
ProdID	<multiple values>

6. Update the IncrementProductProperty task to decrement the value by one.



InsertNewAfter...	2: IncrementProductProperty:111,22
	TaskCreation
Task	IncrementProductProperty
ListOfProdID	111,222
PropertyName	Quantity
StepDirection	-
StepValue	1

7. Run the simulation, verify the Quantity of parts with ProdID of 111 or 22 changes from 10 to 9, and then reset the simulation.

Review

In this tutorial you learned how to increment and decrement the value of a component property using a Works Process task. You know how to add new properties and use them as task arguments. You know how to create a new task, select its arguments, and define its actions. You also know how to redefine a task to support one or more products using a list of ProdID values.