Increment Product Properties

Visual Components 4.1  |  Version: December 3, 2018

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.
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.
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.

![DistributionBook tab](image)

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 visit:
#
# 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, 100, 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.

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:**

```
Part
ProdID (string): 111
Count (int): 5

IncrementProductProperty

Part
ProdID (string): 111
Count (int): 6
```
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.

3. Add an **Integer** property, name it **StepValue**, and then clear the **Rebuild** checkbox. The default value should be zero.
4. In the Works Process, access the TaskCreation script.

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.

```
from vcScript import *
app = getApplication()
comp = getComponent()

taskargs = ["ParentProdID","ListOfProdID","ListOfCompID",
    "SignalValue","PropertyName","PropertyValue","Any",
    "Step1","Step2","PatternName","StartRange","EndRange",
    "TaskName","ToolName","TCPName","All","SyncMessage",
    "VariableName","DefaultValue","MinValue","MaxValue",
    "Else","Text","ComponentName","Time","ListOfProc",
    "OrderName","SensorConveyor","ListName","Simultaneous",
    "StepDirection","StepValue"]

for arg in taskargs:
    if not comp.getProperty( arg ):
        if arg == "Units":
            prop = comp.createProperty( VC_STRING, arg, VC_I
            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.

```python
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.

```python
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.
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:

   ```python
   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.
5. Add the following code.

```python
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.
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.

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

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.

   ```python
   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.

   ```python
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.

   ![Diag1](image1.png)

   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.

   ![Diag2](image2.png)
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.

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

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.