Assembly In Processes

Visual Components Premium 4.3 | Version: February, 2021

This tutorial will introduce how process modelling assembly feature can be used in processes. This tutorial requires basic knowledge from Visual Components UI, process modelling workflow and basics of assemblies.

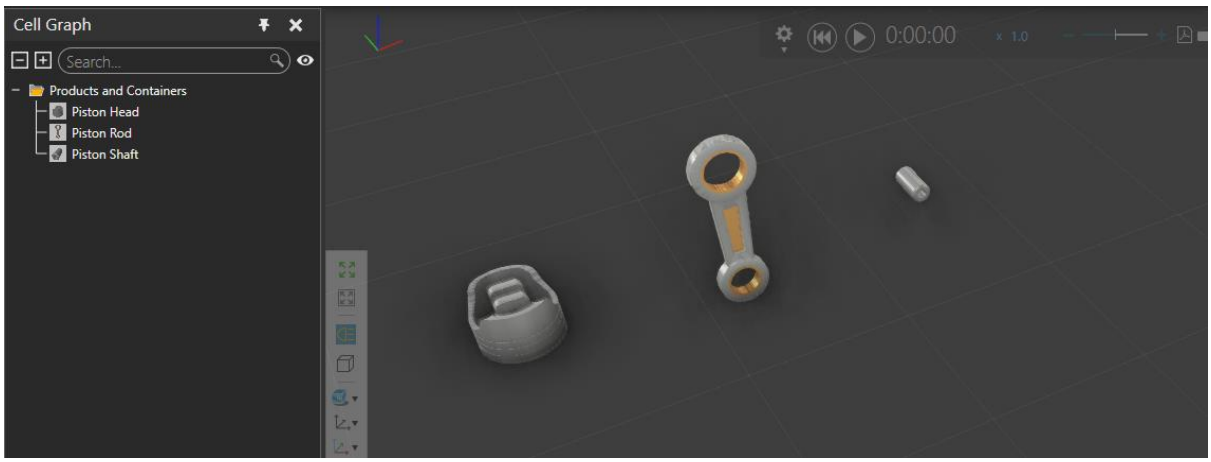
In this tutorial, you will go through:

- Assembly setup in examples
- Creating Assembly vs Product types
- Using GetAssembly-statement in assembly process
- Creating inventories and material pallets
- Assembly of Assemblies
- Construct- and GetProducts statements
- Using assembly properties in process
- Using CheckAssembly

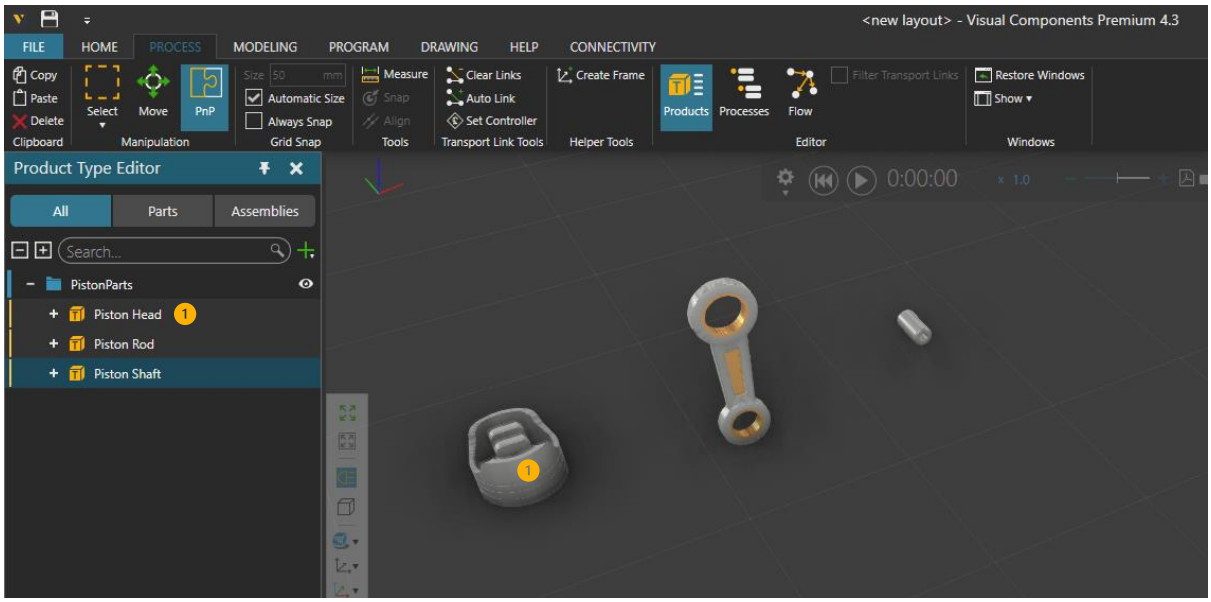
Support
support@visualcomponents.com
Visual Components Forum
forum.visualcomponents.com

Assembly setup in examples

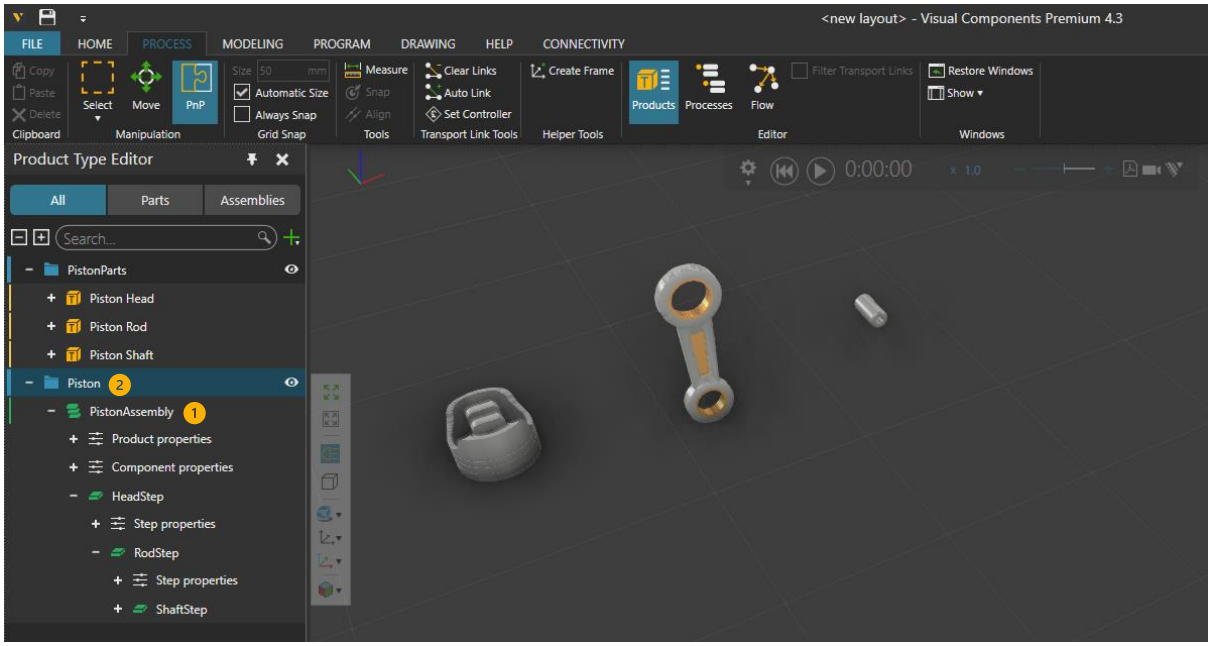
In following tutorial examples, we will use two different assemblies: one for the piston and second for the engine assembly.



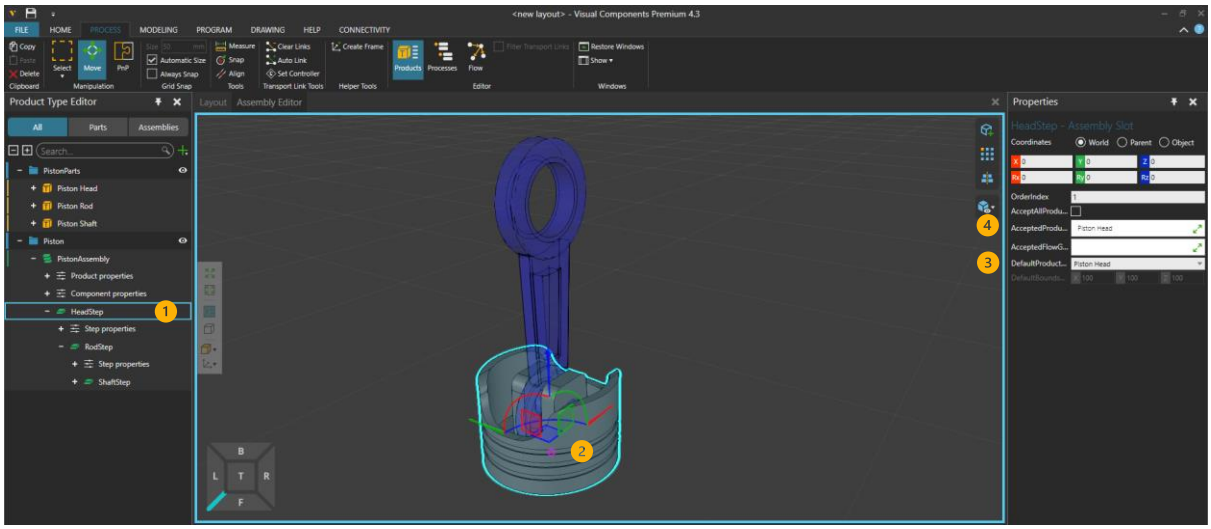
The piston assembly is built from three components from eCat; Piston Head, Piston Rid and Piston Chaft.



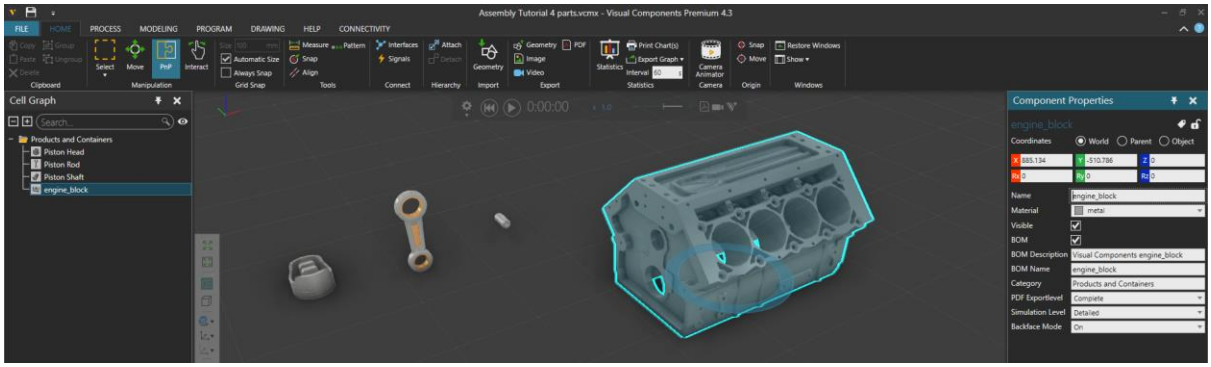
In the process tab > product type editor, new product types (1) are defined with matching components assigned to each type.



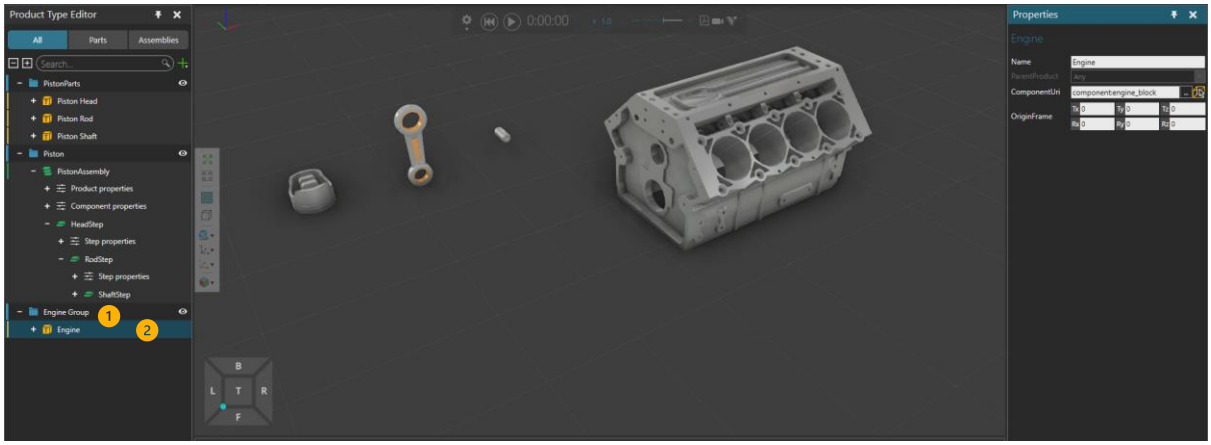
New assembly (1) with three steps (HeadStep, RodStep and ShaftStep) is created to a new flow group (2).



In Assembly Editor, in each step (1), step's slot (2) is selected and matching product type is selected to both DefaultProductType (3) and to AcceptedProductTypes (4).

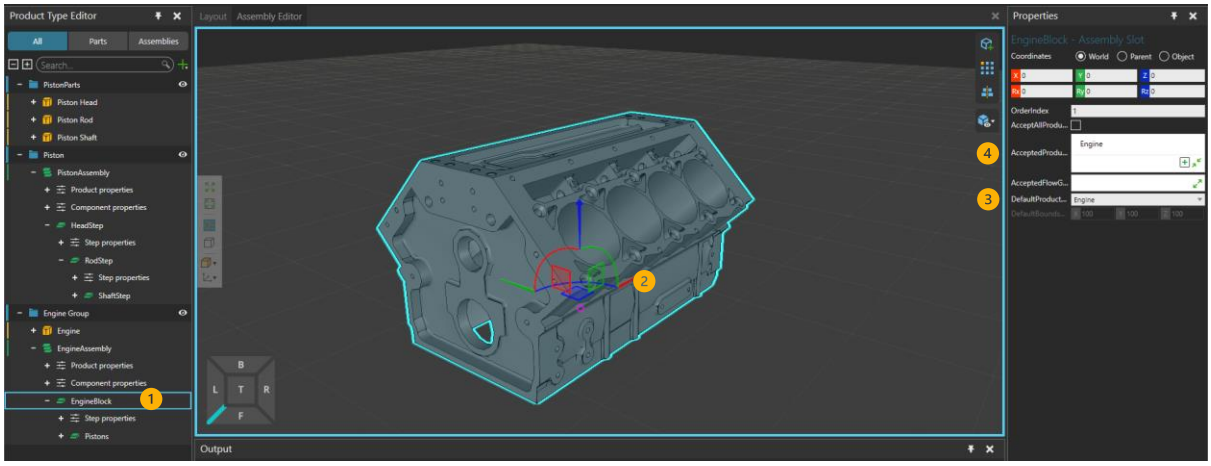


Second assembly is built from engine_block -component and existing PistonAssembly -assembly.

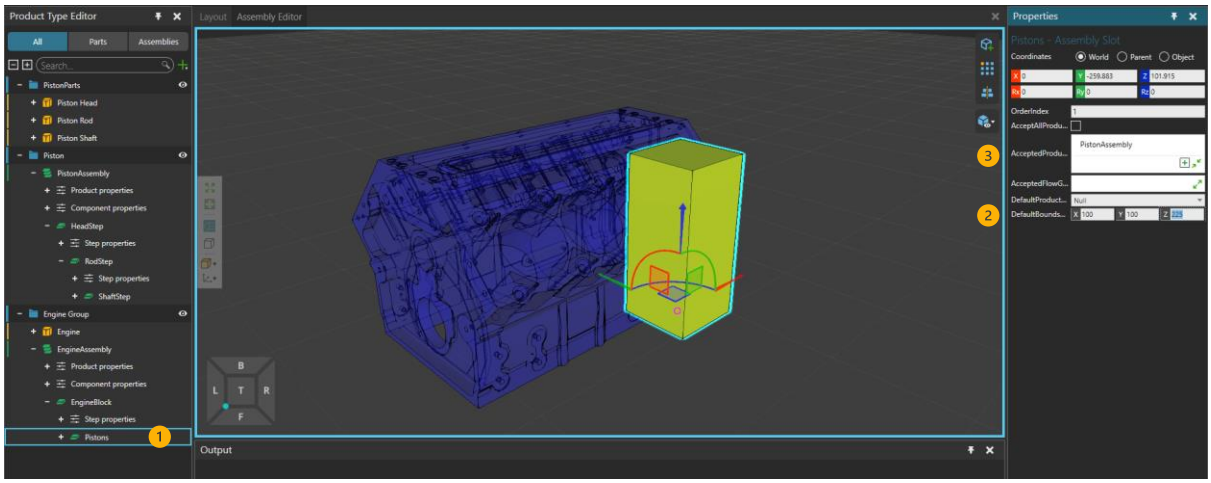


New flow group (1) (Engine Group) and product type (2) (Engine) are defined.

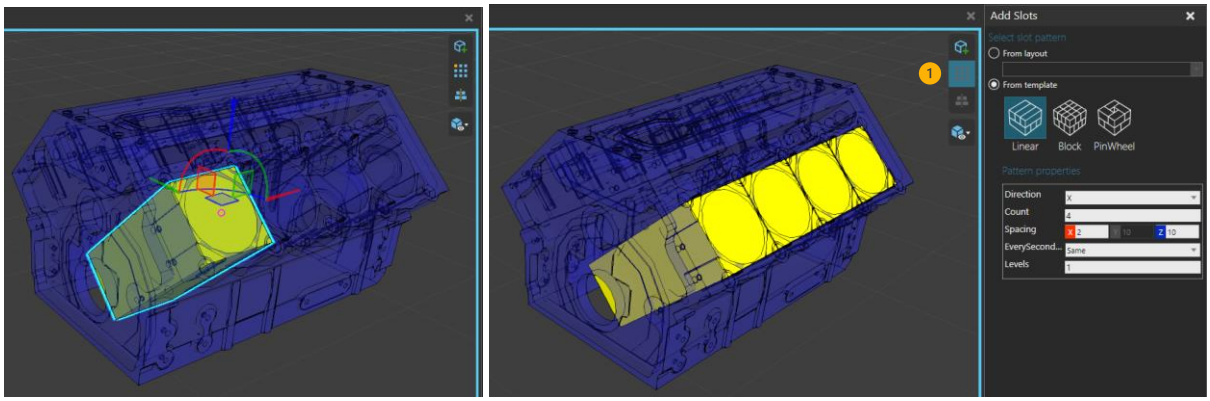
New assembly (1) (EngineAssembly) is created to Engine Group with 2 steps (EngineBlock and Pistons).



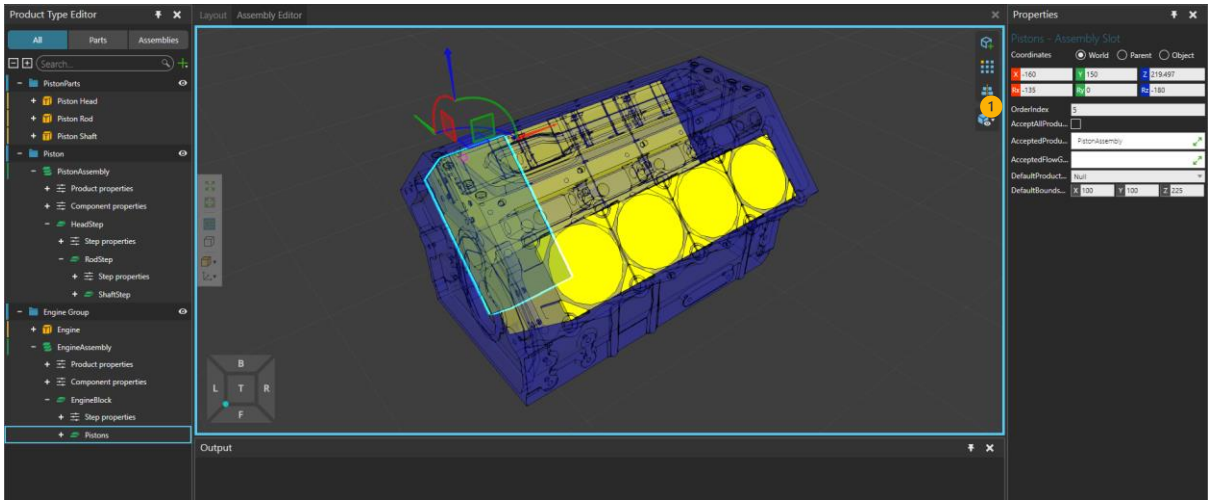
In the Assembly Editor, EngineAssembly's first step's (1) (EngineBlock) slot (2) is selected and Engine -product type is selected to both DefaultProductType (3) and AcceptedProductTypes (4).



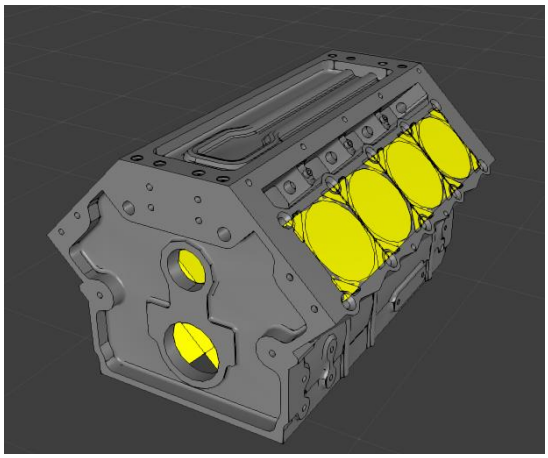
In second step (1) (Pistons) DefaultBoundsDimensions (2) is set to match PistonAssembly's dimensions (~100x100x225) and PistonAssembly is defined into AcceptedProductTypes (3).

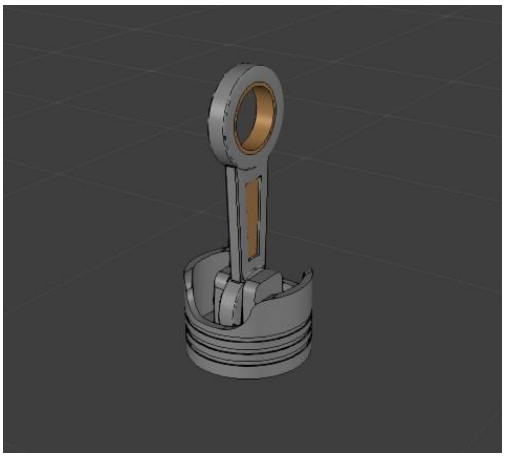


Step is rotated and moved to match the parent step's geometry (EngineBlock) and more slots are added with AddSlots -tool (1).



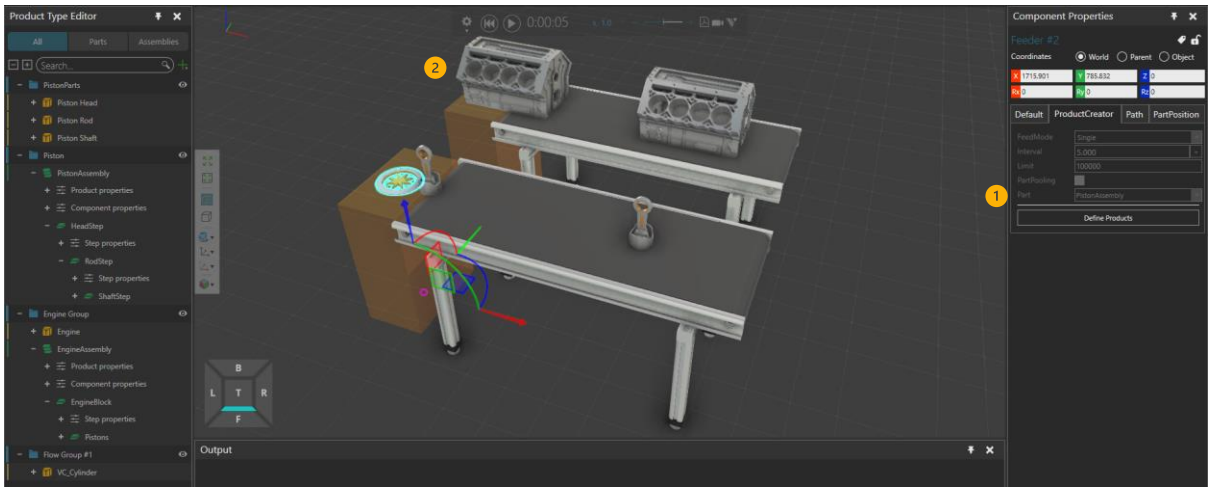
Slots can be selected, rotated and moved to the other side of the engine assembly and desired slots' assembly order can be define with OrderIndex property (1) per slot.





These two assemblies are used in following examples.

Creating Assemblies vs Product types

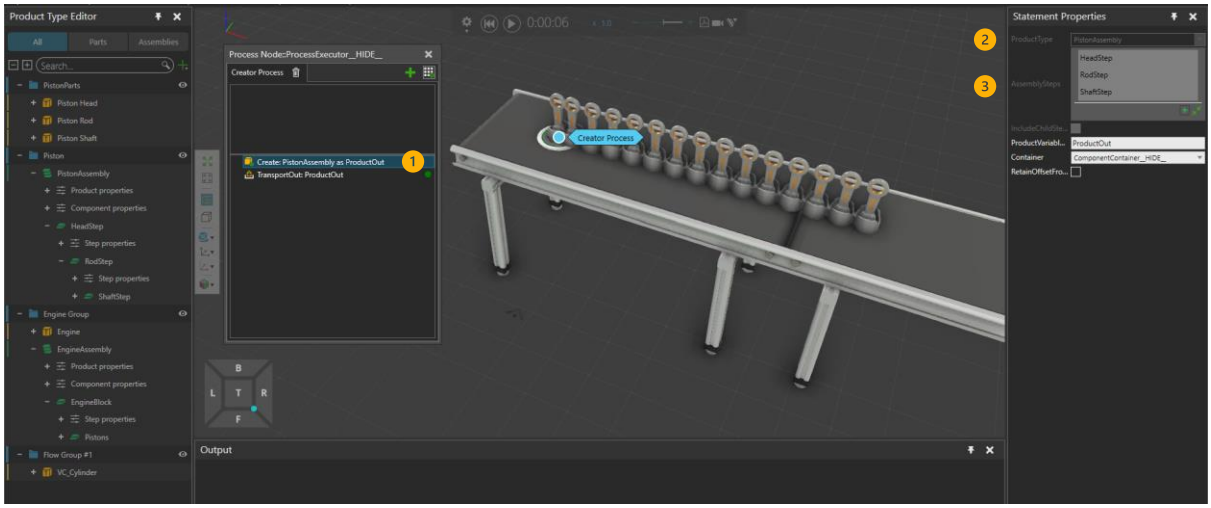


Assemblies can be fed from the feeder components (1) in a same way than any other product type. Only steps with slots that have DefaultProductType defined, are created from the feeder (2).

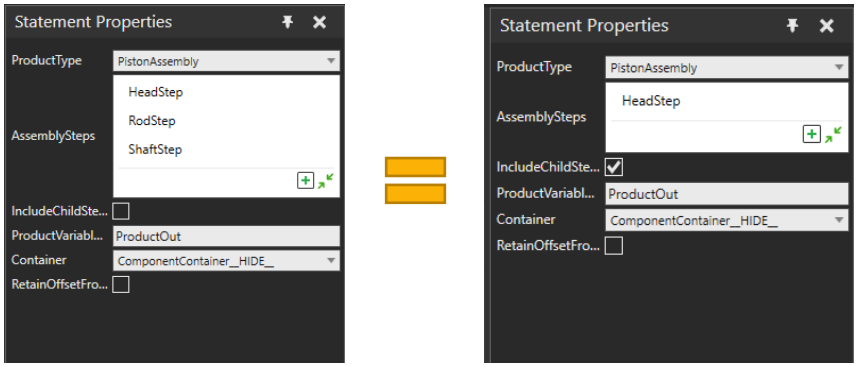
In this example, EngineAssembly is created with only the first step visible, because DefaultProductType is not defined for the slots in Pistons -step.

EngineAssembly is a “normal” product type but with assembly information included.

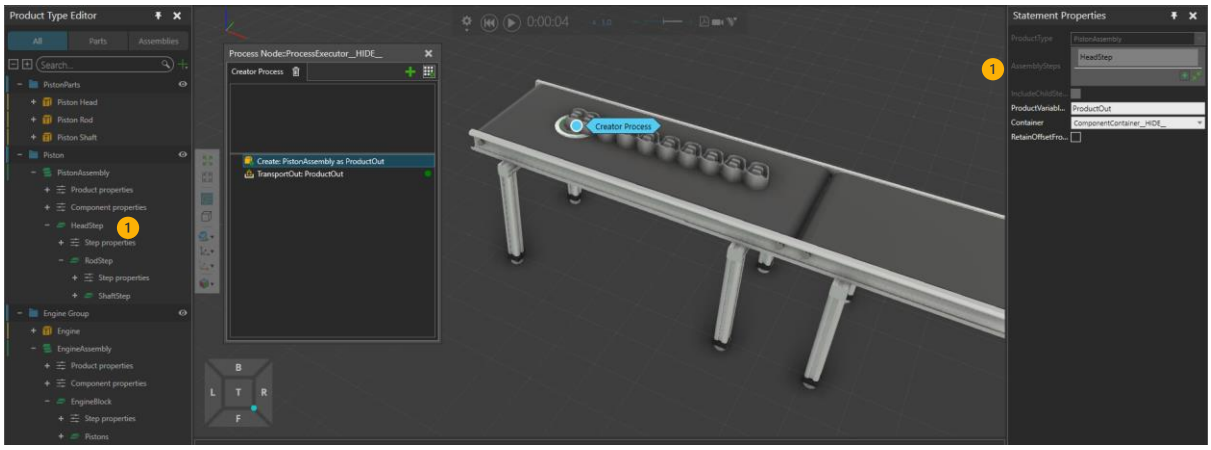
An another way to create assemblies or products used in an assembly, is Create -statement. Statement creates the assembly / product type(s) into process’ container and returns variable containing the assembly / parent product type(s)



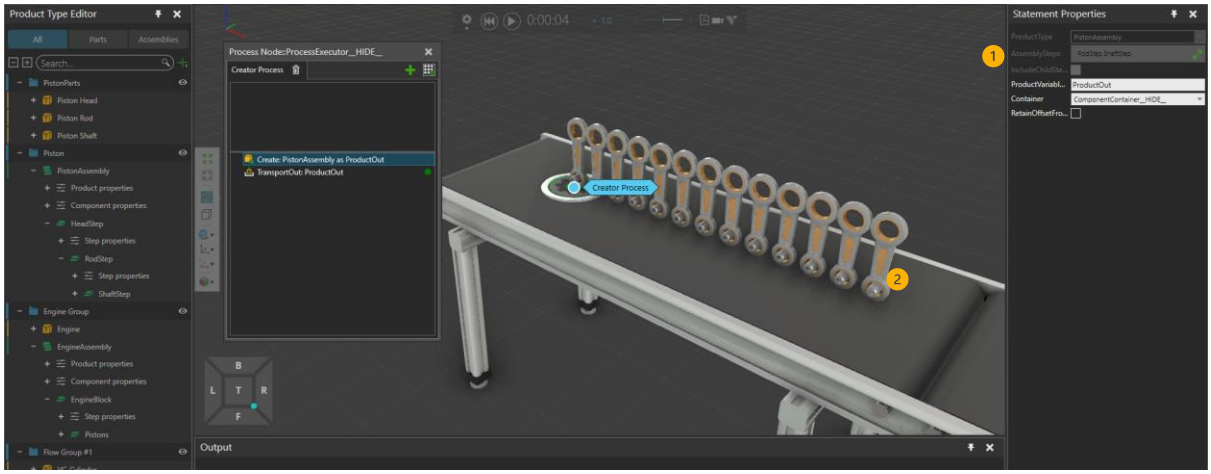
In this example, Create statement (1) is used to create PistonAssembly (2). AssemblySteps -selection (3) can be used to define in detail, which steps are included into the created assembly / product type.



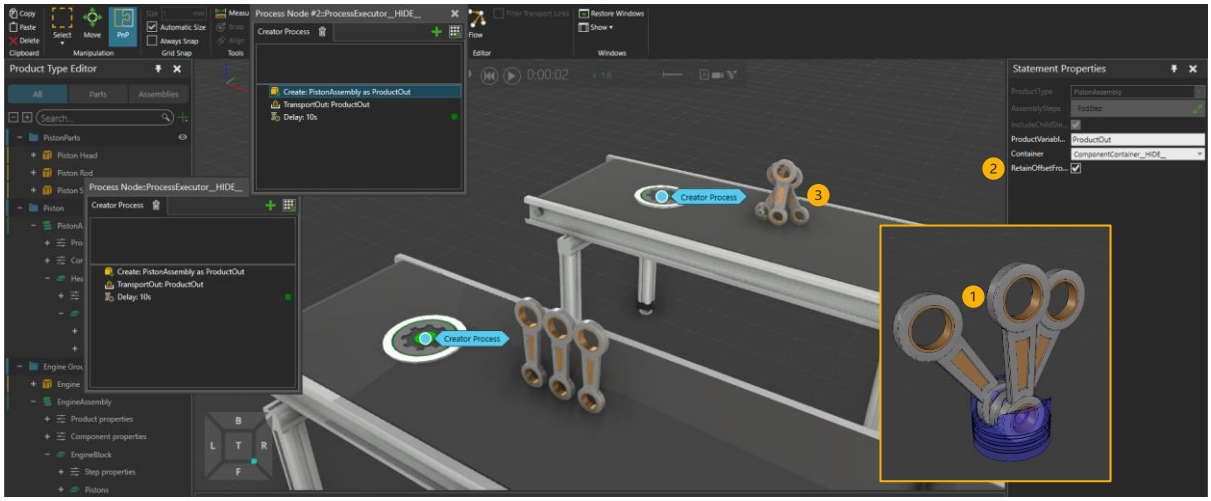
Optionally IncludeChildSteps -property can be used to include all child steps of the selected steps into the created product or assembly. If all steps are selected from Assembly steps, this will result same assembly than when first step is selected from the assembly and IncludeChildSteps is enabled.



If Assembly's first step (1) is included into selection, then created product will be an assembly. Any other steps selected will be created and attached to their parent steps. If parent step is not created, then products in step will be attached to closest parent in assembly.



If assembly's first step is not included into selection (1), then "normal" product types defined in selected steps are created into process. If steps child step(s) are also selected, then child product types will be attached to their parents (2).



If a step includes multiple slots with DefaultProductType defined (1), then all of those products are created. If RetainOffsetFromAssembly (2) is enabled, then created products' orientation is retained from assembly into container (3).

Using GetAssembly-statement in assembly process

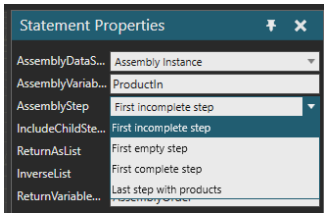
GetAssembly statement can be used to get access to Assembly Data either through arrived assembly instance or directly from assembly definition (assembly in product type editor).

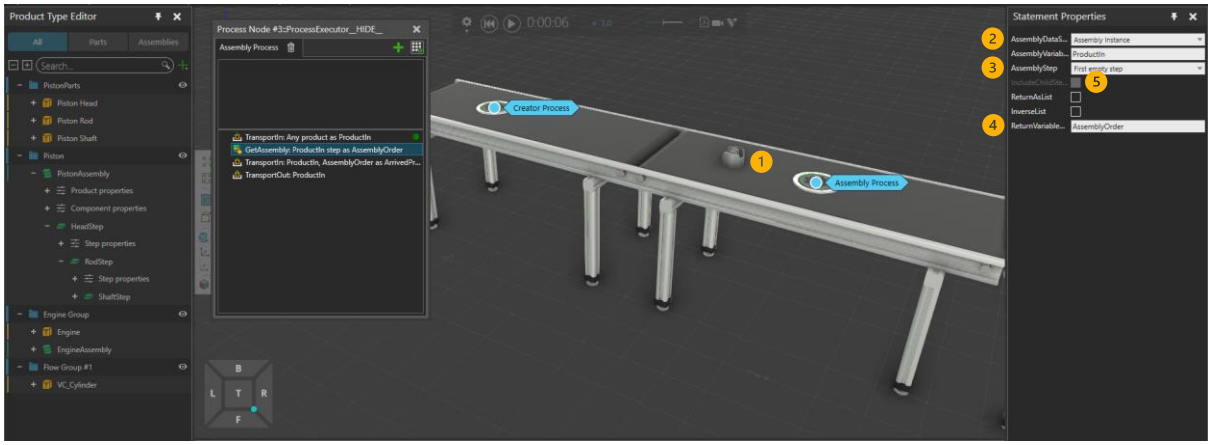
Data source is selected with AssemblyDataSource -property in statement. This data is saved from the GetAssembly statement into "AssemblyOrder" variable.

This assembly order variable can be passed to other statements e.g. to TransportIn statement. Assembly order contains information from assembly, selected steps from the GetAssembly statement and each steps' slot information, including:

- What is the assembly instance to / from products are transported OR is new assembly instance created.
- What product types or assemblies are needed (AcceptedProductTypes/Groups/all)
- To which locations arriving products are transported (Step's origin relatively to it's parent step and slot positions relatively to step's origin)
- How arrived products are parented (Step's hierarchy in assembly)
- In which order products arrive / leave
    - When arriving, parent step(s) first, slots in ascending order (OrderIndex)
    - When leaving child step first, slots in descending order (OrderIndex)
    - If any slots in same step have same order index, then transportation is parallel.
    - If steps are siblings (on same level under common parent), then topmost step is transported in first / transported out last from the siblings.

If DataSource (1) is “Assembly Instance”, then user have options to access assembly instance’s (2) different steps using AssemblyStep property (3):

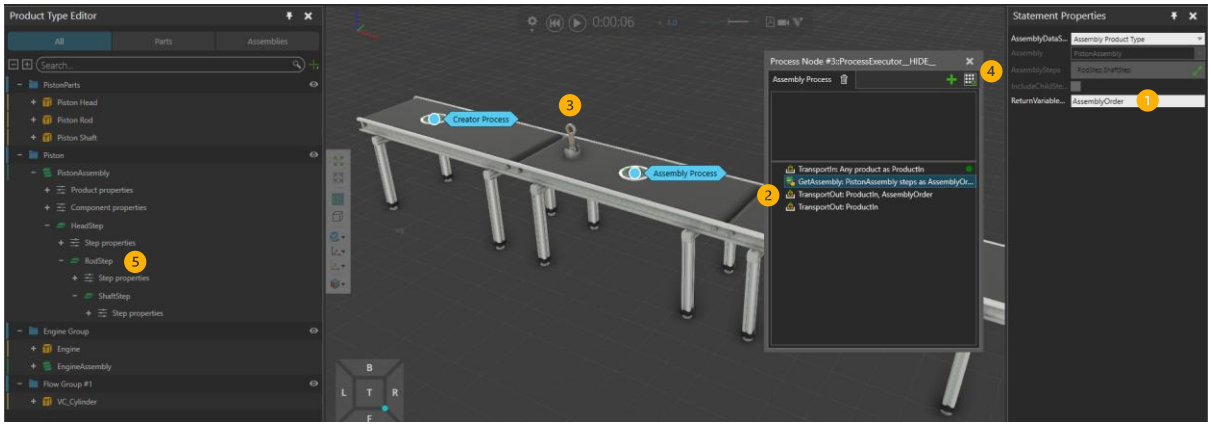




For example, when assembly instance (1) (ProductIn) enters the process (PistonAssembly with only “HeadStep” created from Creator Process), “Assembly Instance” as assembly data source (2) and “First empty step” selection (3) would return AssemblyOrder variable (4) containing information from “Rodstep” and it’s slots. If IncludeChildSteps -property (5) is enabled, then AssemblyOrder variable would contain both RodStep and ShaftStep steps and their slot information.

If AssemblyDataSource (1) is “Assembly Product Type”, then user have options to access assembly’s different steps using AssemblySteps (2) -property.

When “Assembly Instance” is used as assembly data source, then AssemblyOrder data is generated from arrived assembly instance (ProductIn). For example, “First empty step” returns first step from assembly that is totally empty. When “Assembly Product Type” is used, then user needs to check which Assembly instance has arrived to process and then use matching GetAssembly statement (with the same Assembly Product Type).

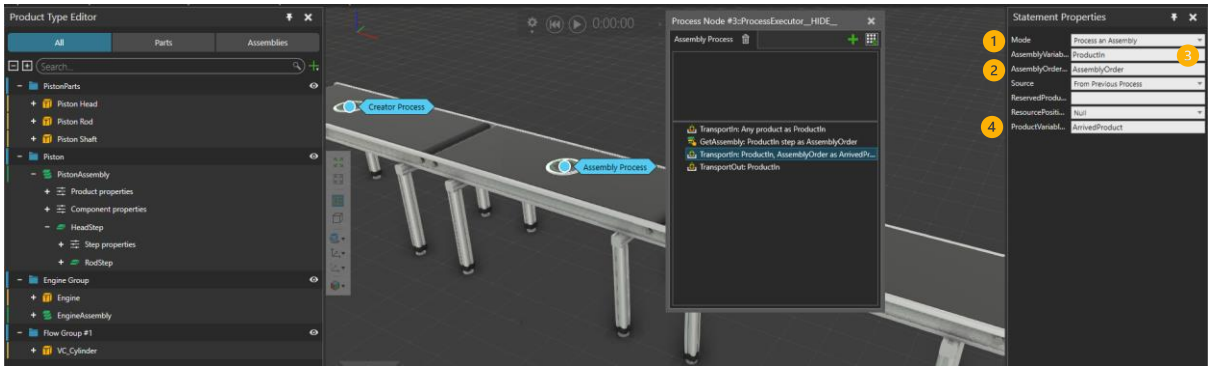


If AssemblyOrder (1) (from GetAssembly with “Assembly Product Type” as AssemblyDataSource) is used to transport out products from the assembly (2), then any available products in selected steps are added AssemblyOrder and then transported out.

In this example, an assembly with HeadStep and RodStep is created (3). In process, GetAssembly access the Assembly Product Type and RodStep and ShaftStep are selected (4). This AssemblyOrder can be used in TransportOut statement but only existing products (5) in those steps will be transported out. In case of TransportIn, Only missing products will be transported into assembly (in this case, only products in ShaftStep as products in RodStep are already present).

GetAssembly statement with “Assembly Product Type” as AssemblyDataSource can be used to create a new assembly into process. When step(s) are selected in GetAssembly’s AssemblySteps -selection and resulting AssemblyOrder is transported in, new assembly instance is created automatically and arriving products are attached to this new assembly.

In this example, “Piston Head” product (1) is created from the Creator Process and first step is selected from the Assembly (including one slot with “Piston Head” product defined in AcceptedProductTypes). In process, GetAssembly (“Assembly Product Type” as AssemblyDataSource) with HeadStep selected (2) is used to transport in (3) only the selected step’s slots, in this case containing one product type of “Piston Head”. New assembly is created in process (Production).

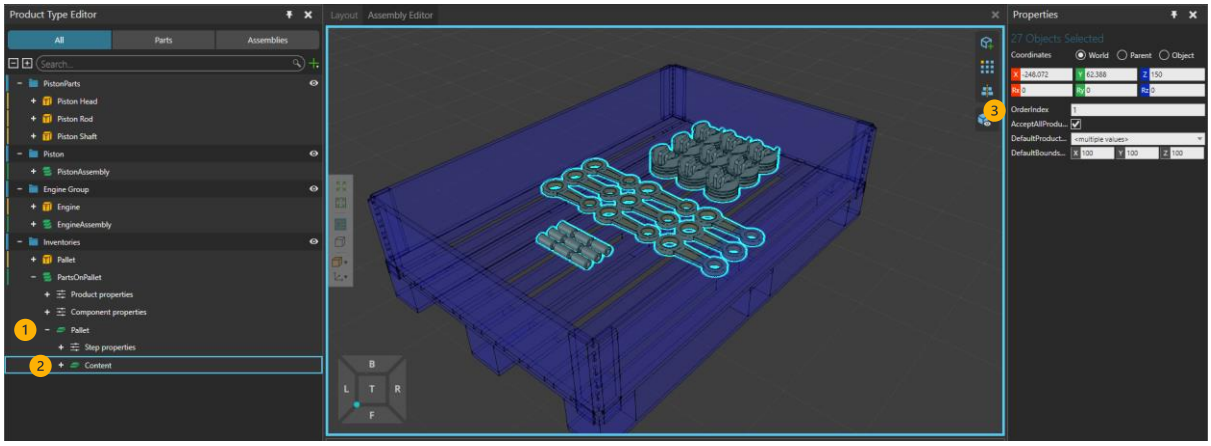


When products (or assemblies as they are products also) are transported in or out using AssemblyOrder variable (thus using GetAssembly statement and some selected steps), TransportIn / TransportOut / StartTransportIn / StartTransportOut needs to be in “Process an Assembly” mode (1). When in this mode, AssemblyOrder variable (2) can be used to transport in or out selected steps.

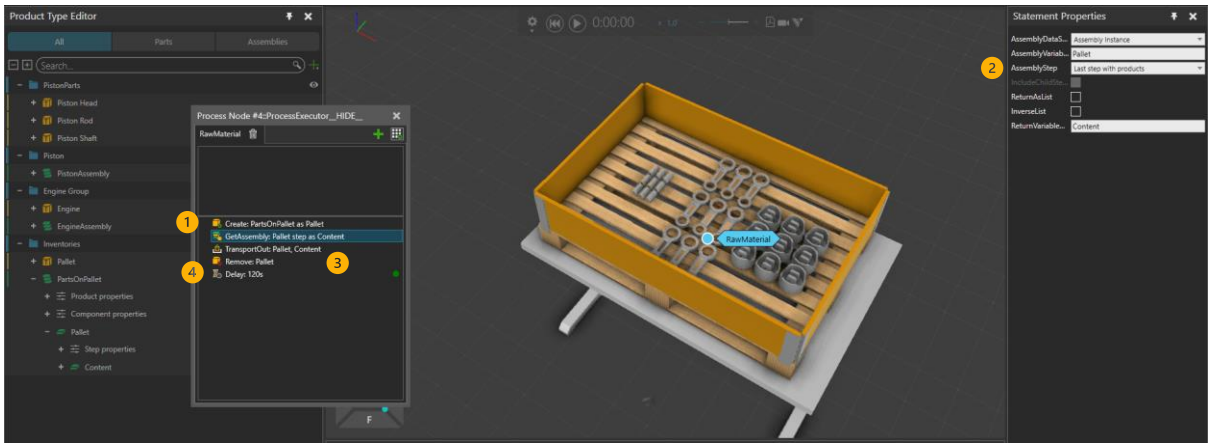
If an assembly is already in the process (e.g. via TransportIn), then AssemblyVariable is defined (3) (ProductIn in this case). Incoming products from following TransportIn are attached into this assembly and in case of TransportOut, leaving products are taken from the defined assembly. Entering or leaving products are saved into ProductVariable (4)

If AssemblyVariable is not defined (empty), then in TransportIn, a new assembly is created automatically (containing arrived products) and saved into ProductVariable (4).

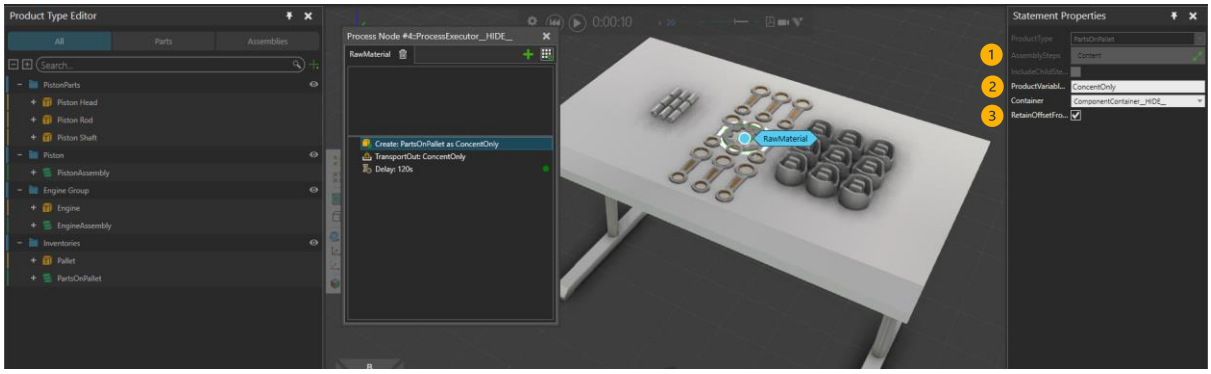
Creating inventories and material pallets



Assemblies can be used to set up products in production into e.g. material pallets, shelves and trays. In this example, PartsOnPallet assembly is defined. It is build from Pallet (1) (with Pallet product type as DefaultProductType) and Content -step built from Piston Heads, -Rods and -Shafts (2). All of the slots in the Content step share the same OrderIndex value (3), thus they are transported out at the same time.

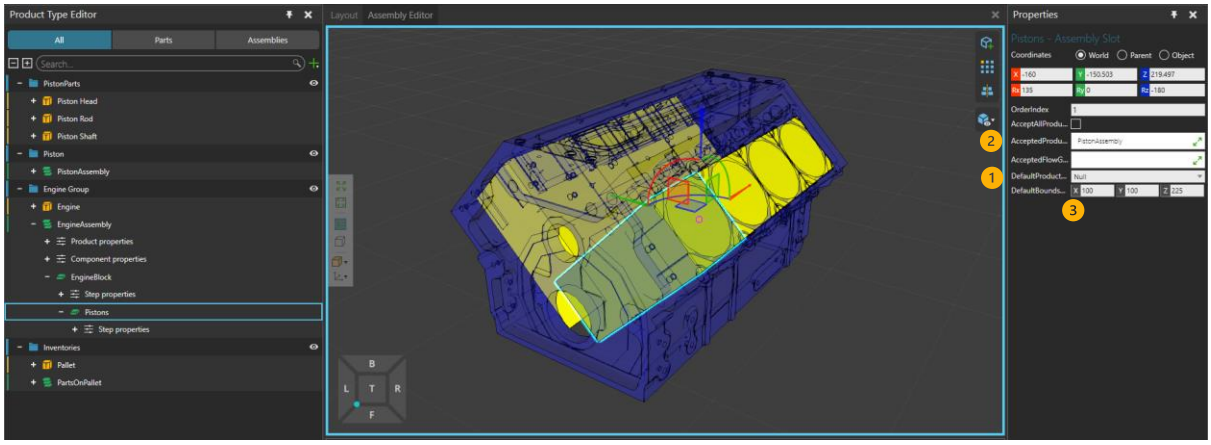


This assembly can be created into process and selected step(s) can be transported out thus simulating available parts in process. In this example whole pallet is created (1), but only Content -step (2) is transported out. When last product from the step is transported out, remaining assembly is removed (3) and delay simulating material refresh time is executed (4). New pallet with content is created when routine starts from the beginning.

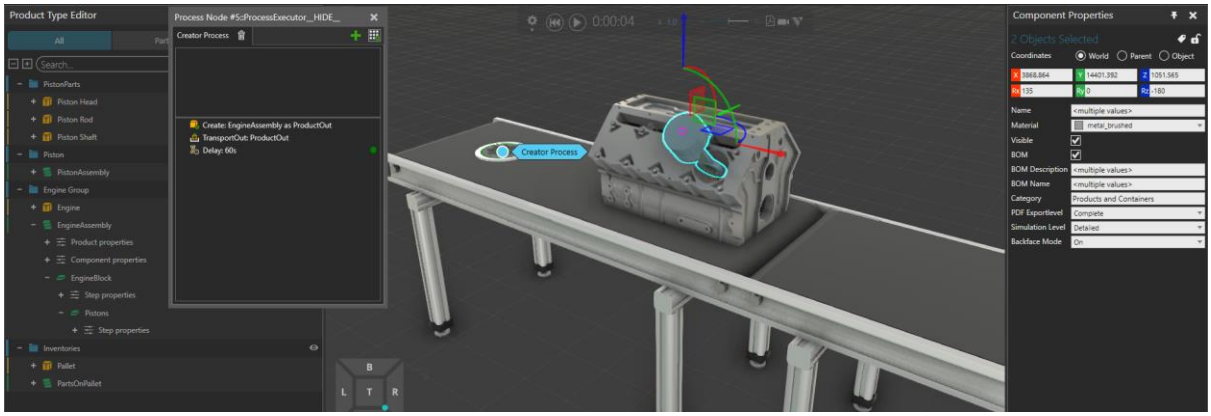


Same example, but only Content -step is created (1). Because the first step is not included, created products are not in assembly and Create -statement returns “normal” list of product types (2). This list can be transported out using TransportOut -statement. RetainOffsetFromAssembly -property retains the products’ offsets from the assembly step, when products are created into process (3).

Assembly of Assemblies



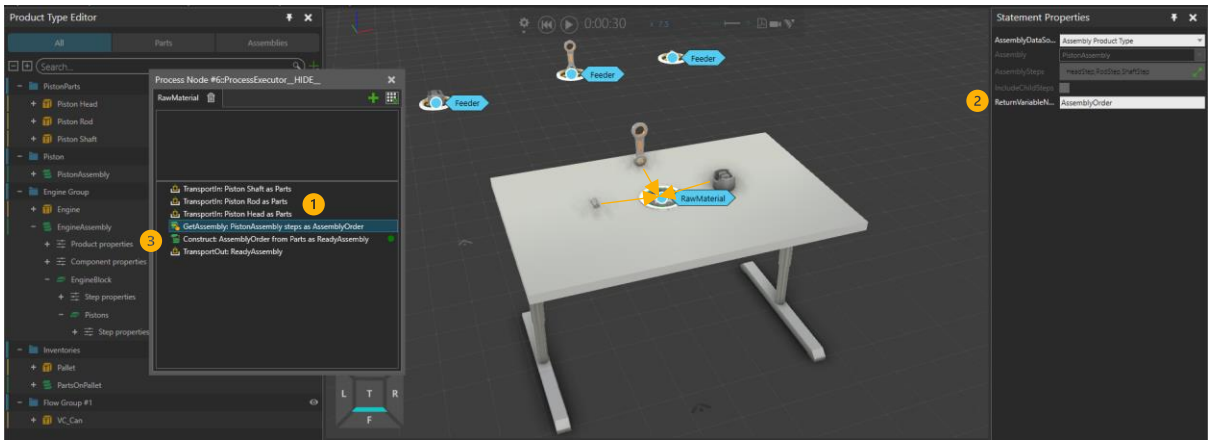
As any assembly is also a Product Type, Existing Assemblies can be used in another assembly as both DefaultProductType (1) and in AcceptedProductTypes (2). In Assembly Editor, sub-assembly is not visualized, so DefaultBoundsDimensions (3) needs to be used to visualize the space sub-assembly will take.



Assembly fed with “PistonAssembly” sub-assemblies.

Construct and GetProducts statements

Construct -statement can be used to build an assembly from existing products in process, regardless how they have entered into process.



In this example, three different products (Shaft, Rod and Head) enter the process via TransportIn (1) and their instances are saved into Parts -variable (list of product instances). GetAssembly is used to get full assembly order from the PistonAssembly (2). Both list of products (Parts) and assembly order (AssemblyOrder) are passed to Contruct statement (3).

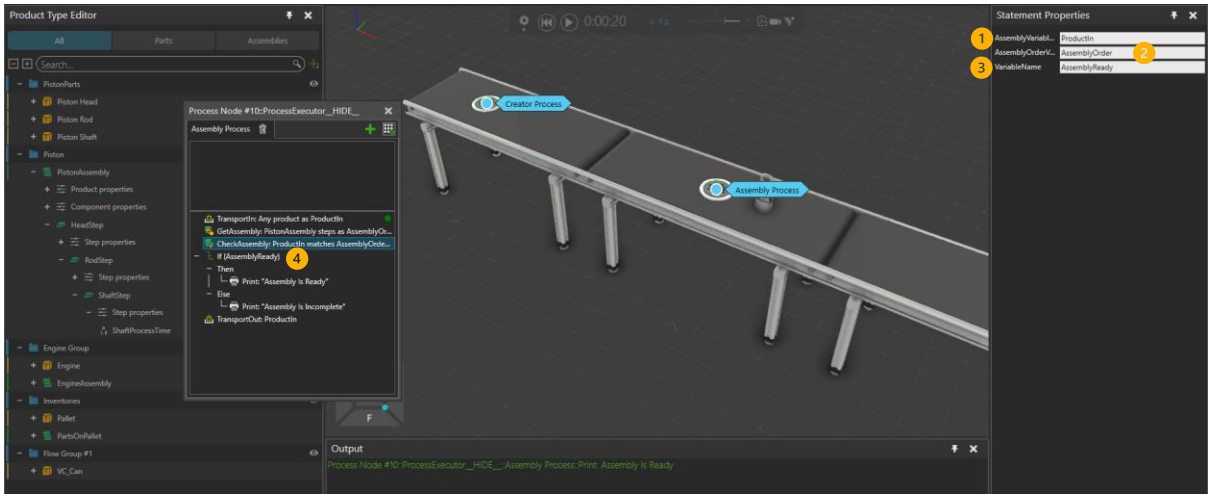
In Construct statement, new assembly (1) (ReadyAssembly) is created from the products in ProductVariable (2) (Parts) based on assembly order (3) (AssemblyOrder). Contruction of assembly is done in AssemblyTime (4) and products in the list are interpolated to correct positions defined in assembly.

New assembly can be created with Construct -statement even only some of the products required to create a full assembly are present.

Using assembly properties in process

In a same way than transporting steps, GetAssembly can be used to access individual step from the assembly (1) (either from Assembly Instance or Assembly Product Type). If this step in assembly has any step properties defined (2), that property can be set to variable with GetAssemblyProperty- statement (3). New variable can be used then in different statements (4) (delay simulating processing time in this case).

Using CheckAssembly



GetAssembly can be used to access one or more steps from Assembly Product Type. This AssemblyOrder can be passed to CheckAssembly Statement that compares arrived instance (1) to selected steps in AssemblyOrder (2) (or to whole assembly if all steps are selected). Resulting Boolean variable (3) can be used in other statements (4) depending is the assembly or selected step(s) complete (variable is true) or are there empty slots in assembly or in step(s) (resulting variable is False).

As an example, if assembly or step(s) is not complete, GetAssembly with DataSource as “Assembly Instance” (1) can be used to create AssemblyOrder containing e.g. a step that is not complete (2) (“First incomplete step”) and those missing components can be transported in.