# Witness W3D Queue Implementation

## Summary:

The rendering of items in queues or on paths is performed by Virtalis, based on information we supply to Virtalis. The sizes and of items, queue position, queue direction and orientation of items within queues is the information we provide to Virtalis to use to render items in queues.

For this discussion we assume a right handed coordinate system with X as screen horizontal, Y as screen vertical and Z as screen depth (right handed +Z moving towards the viewer)

Any geometries that are moved by Witness, will move logically “forwards”, the forwards direction is defined as the logical +Z axis of the shape not the global model +Z. I will try to explain this later.

## Queue Items:

Person, part, box, vehicle, carrier are some examples, are generally defined in Witness and displayed as styles within 2D Witness. Some of the queue items have their own sub-queue such as vehicle and carriers.

During the 2D display style the user can also view the 3D mapping. The 3D mapping selects the 3D geometry for the entity in the 3D view.

If the selected geometry, default “forward (+Z face)” is not correct the user will use any of the 3 orientation overrides, roll(about Z), pitch(about X), yaw(about Y), to make the selected geometry have the correct “forward face” in the +Z orientation.

The items are sized optionally by attributes (Width, Length and Height) for parts, or sized during the display entity style using the size override fields, the size is applied to entity after it is put in the correct forward orientation. The attribute sizing is slightly inconsistent as it uses the terms w*idth*(X axis), *length* (Z axis) and *height* (Y axis). The 3D mapping override fields use the terms w*idth*(X axis), *depth* (Z axis) and *height* (Y axis).

If no size is supplied the default 3D geometry size defined during construction of the 3D shape will be used.

The server commands sent to achieve this are:

```
<load geometry="C:\dg-pt-Part" />
<create time="0.0" geometry="C:\dg-pt-Part" instanceName="p1 - Entity (1)">
    <queueInfo queueParent="dg-pt-Part">
      <behaviour partPositioning="partOver" partRoll="0.0" partPitch="0.0" partYaw="0.0" />
      <position x="0.0" y="0.0" z="0.0" />
      <direction dx="0.0" dy="0.0" dz="1.0" />
    </queueInfo>
</create>
<update time="0.0" instanceName="p1 - Entity (1)" visible="false">
    <rotate order="zxy" x="0.0" y="0.0" z="45.0" />
</update>
```

The first XML element is the load command telling the server to load the geometry into the server from the disk.

Then the create command uses the load geometry to create a named instance in the 3D model, notice the queue info element for embedded queues, this will usually not be used unless the entity is transport type such as vehicle etc.

The update command then applies the orientation and optional scale transforms to orientate and size.

This will have created an invisible instance of an object in the 3D world to be used in queues.

All queue info XML is directly extracted from the Shape XML definition stored on the disk so any changes to the shape.xml for a geometry could influence commands sent to the server which in turn could cause test failures that are based on server command XML assertions.

## Queues:

The queue object will be either a path which is an extruded layout from 2D or a simple queue/icon positioning on the 2D. Again the 2D display dialog will allow access to the corresponding 3D mapping dialog. As for queue items the user can override size and orientation in the same manner as above.

The queue info briefly mentioned above is defined within the shape.xml file stored on disk with each geometry asset. The shape.xml takes the following form:

Example shape.xml for dg-ic-Machine:

```
<?xml version="1.0" encoding="utf-8"?>
<shape guid="{3BF4119E-E91C-49E2-959C-02E01717B55D}" name="dg-ic-Machine">
  <queueInfo queueParent="dg-ic-Machine">
    <behaviour partRoll="0.0" partPitch="0.0" partYaw="0.0" partPositioning="partOver" />
    <position x="0.0" y="-0.35" z="0.1" />
    <direction dx="-1.0" dy="0.0" dz="0.0" />
  </queueInfo>
  <localizedName zh="dg-ic-机器" es="dg-ic-Maquina" fr="dg-ic-Machine" de="dg-ic-Maschine" pl="dg-ic-
Machine" ja="dg-ic-マシン" />
</shape>
```

The important part of the above xml for queues is the *queueInfo* element. This defines the information that will be passed to the server to define how items will queue at this geometry. The *queueParent* attribute will generally be the main geometry in this case “dg-ic-Machine”. If a geometry is hierarchical with name sub-nodes the queue could be parented relative to a sub-node name, allowing animation of a sub-node to move the queue in actions such as moving robot arms etc.

The *behaviour* defines any local item re-orientation to be performed on the items when in the queue. This is a temporary transform applied to the queue item in addition to its own orientation and scale defined in the queue Items section above. This allows items to be changed to move in a non-forward direction along a queue for example moving cars sideways down a queue by giving a 90 degree *partYaw*. The *partPositioning* defines if the item in the queue sits on top of the queue

vector(*partOver*), sits below the part queue vector(*partUnder*) or is threaded on the vector(*partCentre*) like beads on a string .

The *position* defines the start point of the queue in global coordinate system relative to the named *queueParent*, the position should be scaled along with the parent geometry if needed.

The *direction* element defines a normalised (length 1), queue vector will start at the *position* moving outwards in the direction defined by the *dx, dy, dz* attributes.

So the above object is defined as having a queue that has a start position relative to the main parent "dg-ic-Machine" centre at a position of (0.0, -0.35, 0.1) The queue will stretch outwards from the centre by simple vector addition (-1, 0, 0) so the queue "end" is at (-1.0, -0.35, 0.1). Parts entering this queue will have no partItem transorms because none all partItem orientation fields are 0.0. We will always put the part sitting "on top" of the queue vector because part positioning is over, which will translate the part in the queue +Y by half a part item orientation height.

The server commands sent to create a queue are as follows:

```
<load geometry="C:\dg-ic-Machine" />
<create time="0.0" geometry="C:\dg-ic-Machine" instanceName="[114] m1(1) - Icon">
  <queueInfo queueParent="dg-ic-Machine">
    <behaviour partRoll="0.0" partPitch="0.0" partYaw="0.0" partPositioning="partOver" />
    <position x="0.0" y="-0.35" z="0.1" />
    <direction dx="-1.0" dy="0.0" dz="0.0" />
  </queueInfo>
</create>
<update time="0.000000" instanceName="="[114] m1(1) - Icon ">
  <translate x="232.000000" y="0.500000" z="160.000000" />
  <rotate order="zxy" x="0.000000" y="-180.000000" z="0.000000" />
</update>
```

This will first load the asset/geometry from disk along with the shape.xml

The create command will create an instance of the queue object in the 3D world. Notice that the queueInfo passed in the create command is directly matched to the shape.xml. Changes to shape.xml can cause tests asserting server commands to fail if queueInfo values change that are asserted. The queue partOrientation values also directly affect part orientation in queue during run, this directly affects how much queue space an item in the queue uses, so changing the position of all following items in the queue so changing test asserts on queue positions.

Items are then added into the queue using a special update command to place items in a queue at a position:

```
<update time="0.0" instanceName="p1 - Entity (1)">
    <partPosition instanceName="[114] m1(1) - Icon" position="0.5" />
</update>

<update time="0.0" instanceName="p1 - Entity (2)">
    <partPosition instanceName="[114] m1(1) - Icon " position="1.5" />
</update>
```

This will first add the "`p1 - Entity (1)`" to the `"[114] m1(1) - Icon"` this position has to be calculated based on the queue length used previously and the orientation within the queue of the item we are adding. The item will be rotated so that the item forward face is point towards the start, unless the partItem transorms apply and addition transform.