Passing items from one Object to another

Say you had a game where your player character needed to collect objects and carry them around. Also it needed to drop them off or give them to another character.

Objects that are children of a parent object can easily fulfil this role. Thankfully the parenthood of an item can be assigned at will to another object.

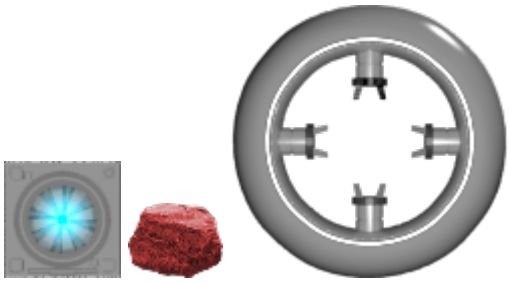
Let's work through an example where an object will travel from left to right. If it collides with the item, the object will “collect” it. And when the object collides with the dropoff object, it will pass the item to the dropoff.

Default Project

Create a project using Orx's init script.

Assets

Three objects are required for this tutorial. Download each and save them to your data/texture folder in your project.



Defining the Objects in the Config

Start with the object config:

```
[Carrier]
Graphic  = @
Texture  = boss-core.png
Pivot    = center
Position = (-300, 0, 0)
Body     = CarrierBody

[Item]
Graphic  = @
Texture  = item.png
Pivot    = center
```

```
Position = (0, 0, -0.1)
Body     = ItemBody

[Dropoff]
Graphic         = @
Texture         = dropoff.png
Pivot           = center
Position        = (300, 0, 0)
Body            = DropoffBody
AngularVelocity = 20
```

As each will require a body for collision detection, define that next:

```
[CarrierBody]
Dynamic  = true
PartList = CarrierBodyPart

[CarrierBodyPart]
Type      = box
SelfFlags = carrier
CheckMask = item
Solid     = false

[ItemBody]
Dynamic  = true
PartList = ItemBodyPart

[ItemBodyPart]
Type      = box
SelfFlags = item
CheckMask = carrier # dropoff
Solid     = false

[DropoffBody]
Dynamic  = true
PartList = DropoffBodyPart

[DropoffBodyPart]
Type      = box
SelfFlags = dropoff
CheckMask = item
Solid           = false
```

Just a few notes on the bodies defined above:

- The carrier can collide with the item
- The item can collide with the carrier and the dropoff
- The dropoff can collide with the item

Next, make a Scene object to contain the three objects:

```
[Scene]
ChildList = Carrier # Item # Dropoff
```

Now we can display this to the screen in the init() function by changing:

```
orxObject_CreateFromConfig("Object");
```

to:

```
orxObject_CreateFromConfig("Scene");
```

Compile and run. There should be a carrier object on the left, an item in the middle, and a dropoff object on the right. The latter will be spinning slowly.

The carrier will need to travel right, make the delivery, and then return to the left. We'll create a position FX to achieve this:

```
[MovementFX]
SlotList = MovementFXSlot
Loop     = false

[MovementFXSlot]
Type       = position
Curve      = sine
StartTime  = 0.0
EndTime    = 7.0
Absolute   = false
StartValue = (0, 0, 0)
EndValue   = (520, 0, 0)
```

And add this FX to the carrier:

```
[Carrier]
Graphic  = @
Texture  = carrier.png
Pivot    = center
Position = (-300, 0, 0)
Body     = CarrierBody
FXList   = MovementFX
```

Run this and the carrier will move to the right, then back to the left. We haven't supplied any actions on physics events yet. Let's do that now.

Physics Handler

Create the following Physics Handler:

```
orxSTATUS orxFASTCALL PhysicsEventHandler(const orxEVENT *_pstEvent)
```

```
{
    orxSTATUS eResult = orxSTATUS_SUCCESS;

    if (_pstEvent->eID == orxPHYSICS_EVENT_CONTACT_ADD)
    {
        orxPHYSICS_EVENT_PAYLOAD *pstPayload;
        pstPayload = (orxPHYSICS_EVENT_PAYLOAD *)_pstEvent->pstPayload;

        orxOBJECT *recipient, *sender;
        recipient = orxOBJECT(_pstEvent->hRecipient);
        sender = orxOBJECT(_pstEvent->hSender);

        const orxSTRING recipientName = orxObject_GetName(recipient);
        const orxSTRING senderName = orxObject_GetName(sender);

        if (orxString_Compare(recipientName, "Item") == 0 &&
orxString_Compare(senderName, "Carrier") == 0) {
            orxObject_SetParent(recipient, sender);
        }
        if (orxString_Compare(senderName, "Item") == 0 &&
orxString_Compare(recipientName, "Carrier") == 0) {
            orxObject_SetParent(sender, recipient);
        }

        if (orxString_Compare(recipientName, "Dropoff") == 0 &&
orxString_Compare(senderName, "Item") == 0) {
            orxObject_SetParent(sender, recipient);
        }
        if (orxString_Compare(senderName, "Dropoff") == 0&&
orxString_Compare(recipientName, "Item") == 0) {
            orxObject_SetParent(recipient, sender);
        }
    }

    return eResult;
}
```

I'll explain the above. The physics event handler checks if the event type is a `orxPHYSICS_EVENT_CONTACT_ADD`. Which means a collision. Next, the two colliding object names are retrieved.

Here comes the meat of the tutorial: if the item collides with the carrier, assign the item as child of the carrier.

In the same way, if the item collides with the dropoff, make the item as a child of the dropoff.

Add the `PhysicsEventHandler` in the init() function:

```
orxEvent_AddHandler(orxEVENT_TYPE_PHYSICS, &PhysicsEventHandler);
```

Compile and run. The carrier will collide with the item, and “pick it up” with the

`orxObject_SetParent` function. In effect, the item becomes a child of the carrier, and becomes part of the parent space. Because the item's graphic has a `Pivot = center`, it will be placed in the center of the item.

There is an alternative function that allows you to parent the item to an object, while at the same time preserving the current world position of the child object. That function is:

`orxObject_Attach`

You can replace the above functions, ie:

```
orxObject_SetParent(recipient, sender);
```

with:

```
orxObject_Attach(recipient, sender);
```

> Attaching still means that a child is parented.

Compile and run again. The carrier will bump into the item and start carrying at the point it was collided. You'll notice the same (but odd) effect when the item collides with the dropoff. It won't be attached in the center of the drop off, but at the last collision point. Because the parent slowly turns, the item will slowly rotate around the outside of it.

You might like the item to attach itself to some other position on the carrier. You might like to ensure the the item, for example, attaches at 5 pixels from the top, and 10 from the left of the carrier. Maybe the character is a person, and you want the item to attach to his hands. You can use the function:

`orxObject_SetWorldPosition`

...right before you attach.

I'll leave that one to you to experiment with.