**productAddItem.evt**

<aura:event type="COMPONENT">

    <aura:attribute name="products" type="List" />

</aura:event>

Events are defined as a separate entity from components, controllers and helper, and The event definition also determines the event type: COMPONENT or APPLICATION.

**productForm.cmp (Child Component)**

<aura:component>

    <aura:attribute name="products" type="List" />

    <lightning:button label="Add Product" press="{!c.clickAddProduct}" />

    <aura:registerEvent name="addProduct" type="c:productAddItem" />

</aura:component>

this form component has a button which has attached to the event. The component registers an event (**registerEvent** is your sending component) named **addProduct**.

Keep this name in mind, because the receiving component will call this same name. It’s binds the sender and receiver together.

The button references the **clickAddProduct** function in its controller, which begins the event process.

**productFormController.js**

({

    clickAddProduct: function(component, event, helper) {

        var products = component.get(“v.products");

        var createEvent = component.getEvent(“addProduct");

        createEvent.setParams({“products" : products});

        createEvent.fire();

    }

})

**products.cmp (Parent Component)**

<aura:component>

    <aura:attribute name=“products" type=“List"/>

     <aura:handler name="addProduct" event="c:productAddItem" action="{!c.handleAddProduct}" />

     <c:productForm />

</aura:component>

Just like we registered the event in the **child component**, now we **handle** it in the **parent component** with a **handler**.

Note that the name, **addProduct**, is the same as when we registered the event. The action attribute defines the next step in the loop, which calls a function in the controller called **handleAddProduct**.

**productsController.js**

({

    handleAddProduct: function(component, event, helper) {

        var newProduct = event.getParam(“products");

        var action = component.get("c.saveProduct");

        action.setCallback(this, function(response) {

        });

        $A.enqueueAction(action);

    }

})

This is the receiving controller that handles the incoming event and communicates with the apex controller. The function, saveProduct, is a reference to the apex controller.  The final line of code queues up the action (the call to the server), the event cycle is complete.