Developing Lightning Components

Thomas Crouse
Salesforce Developer, Trifecta Technologies Inc.

Contact Search Example

Topics

- Lightning Components vs Visualforce
- The Lightning Component bundle
- Core concepts
- Sample contact search app

What's This All About?

- Modern UI framework
  - Dynamic single-page web apps
  - Mobile and desktop friendly
  - Built on Aura
- Apps are built from components
  - Units of functionality, small and large
  - Encapsulation and decoupling

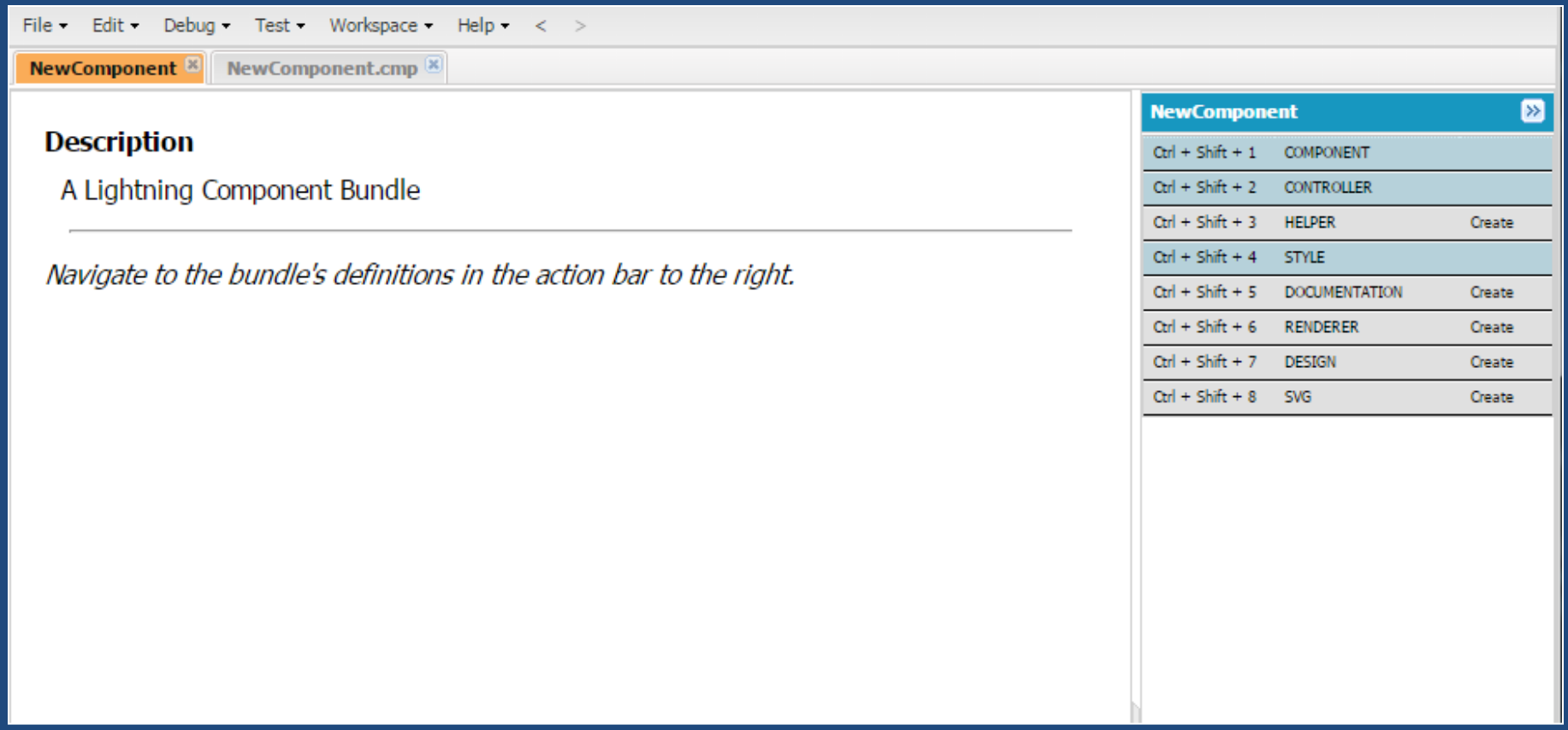
Visualforce

- Stateful server
  - State is encoded string, potentially-large
- Page-centric
  - Heavy on server calls
- Lots of generated code and markup

Lightning Components

- Stateful client, stateless server
  - Instant updates after client-side changes
  - Only necessary data retrieved from server
- Event-driven communication
  - Decoupled and light-weight
- Mobile-friendly

The Component Bundle


File
Edit
Debug
Test
Workspace
Help
NewComponent
NewComponent.cmp
Description
A Lightning Component Bundle
Navigate to the bundle's definitions in the action bar to the right.
NewComponent
Ctrl + Shift + 1 COMPONENT
Ctrl + Shift + 2 CONTROLLER
Ctrl + Shift + 3 HELPER Create
Ctrl + Shift + 4 STYLE
Ctrl + Shift + 5 DOCUMENTATION Create
Ctrl + Shift + 6 RENDERER Create
Ctrl + Shift + 7 DESIGN Create
Ctrl + Shift + 8 SVG Create

The Component

```
<aura:component >
    <aura:attribute access="public" name="record" type="Contact" required="true"/>

    <aura:registerEvent name="contactSelectedEvent" type="c:ContactSelected"/>
    <aura:handler name="contactSelectedEvent" action="{!c.navigateToView}"/>

    <div class="contactEntry" onclick="{!c.selectContact}">
        <ui:outputText value="{!v.record.Name}"/><br/>
        <ui:outputPhone value="{!v.record.Phone ? v.record.Phone : 'N/A'}"/><br/>
        <ui:outputEmail value="{!if( v.record.Email, v.record.Email, 'N/A' )}"/>
    </div>
</aura:component>
```

- Declare attributes
- Register fire-able events
- Declare event handlers
- Include all other markup (HTML treated as first-class components)
  - Define local ids using aura:id="myId"

Expressions

- Similar syntax as in Visualforce
  - Operators and functions available

- Value Providers
  - "v" for accessing attributes (view)

```
<aura:attribute name="record" type="Account"/>
<ui:outputText value="{!v.record.Name}"/>
```

  - "c" for accessing actions (controller)

```
<ui:button label="Search" press="{!c.search}"></ui:button>
```

Lightning Events

```
<aura:event type="COMPONENT" >
    <aura:attribute name="contactsList" type="Contact[]"/>
</aura:event>
```

- Either COMPONENT or APPLICATION
  - Component: handled by self or by parent
  - Application: publish-subscribe model
    - All other components with a handler are notified
- Also system events and Salesforce1 events
  - init, aura:waiting and aura:doneWaiting
  - force:createRecord and force:navigateToSObject

Handling Component Events

```
<aura:registerEvent name="contactSelectedEvent" type="c:ContactSelected"/>
<aura:handler name="contactSelectedEvent" action="{!c.navigateToView}"/>
```

```
<c:ContactSearchComplete contactsFoundEvent="{!c.searchResultsFound}"/>
```

Handling Application Events

```
<aura:handler event="c:someAppEvent" action="{!c.handleEvent}"/>
```

Component Style

```
.THIS.contactEntry {
    background-color: cyan;
    border: 1px solid black;
    border-radius: 5px;
    margin: 3px;
    padding: 3px;
}
```

- Every rule preceded by .THIS
  - Namespacing
- Vendor prefixes automatically added
  - -moz and -webkit
- But what about external resources?

External Styles/Scripts

```
<ltng:require styles="/resource/bootstrap"
    scripts="/resource/resourceName"
    afterScriptsLoaded="{!c.afterScriptsLoaded}" />
```

- Asynchronous resource loading
- One-time loading
- Encapsulation

- External CSS should be namespaced

Component Controller

- Expose actions and handlers to the component
- Auto-wired to reference component and event
- Can include reference to helper for reusable JavaScript

```
({
    searchResultsFound : function(component, event, helper) {
        helper.searchResultsFound(component, event);
    },

    showSpinner : function (component, event, helper) {
        var spinner = component.find("spinner");
        var evt = spinner.get("e.toggle");
        evt.setParams({ isVisible : true });
        evt.fire();
    },

    hideSpinner : function (component, event, helper) {
        var spinner = component.find("spinner");
        var evt = spinner.get("e.toggle");
        evt.setParams({ isVisible : false });
        evt.fire();
    }
})
```

Component Helper

- Same structure as controller
  - Methods not accessible from component
  - Can define any parameters

```
({
    add : function(a, b) {
        return a + b;
    },

    reallyBadIdea : function() {
        this.reallyBadIdea();
    }
})
```

Apex with Components

```
1  public with sharing class ContactSearch {
2
3      @AuraEnabled
4      public static List<Contact> getContacts(String searchTerm, Integer maxResults) {
5          List<Contact> results = new List<Contact>();
6
7          if (String.isNotEmpty(searchTerm)) {
8              String wildcardSearchTerm = searchTerm + '%';
9              Integer recordLimit = Integer.valueOf(maxResults);
10
11             results = [
12                 SELECT Id, Name, Phone, Email
13                 FROM Contact
14                 WHERE Name LIKE :wildcardSearchTerm
15                 ORDER BY Name
16                 LIMIT :recordLimit
17             ];
18         }
19         return results;
20     }
21 }
```

Apex with Components

```
<aura:component controller="ContactSearch">
```

```
helperSearch : function(component, event) {
    var searchText = component.find("searchTerm").get("v.value");
    var recordLimit = component.get("v.maxResults");

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

    action.setParams({
        searchTerm: searchText,
        maxResults: recordLimit
    });

    action.setCallback(this, function(a){
        var contactsFoundEvent = component.getEvent("contactsFoundEvent");
        contactsFoundEvent.setParams({
            contactsList: a.getReturnValue()
        });
        contactsFoundEvent.fire();
    });

    $A.enqueueAction(action);
}
```

Using Your New Components

Lightning Application

```
<aura:application >
    <aura:attribute name="contacts" type="Contact[]"/>
```

- Top-level container
- https://<SF instance>.lightning.force.com/<namespace>/<bundle name>.app
  - https://na24.lightning.force.com/c/ContactSearch.app
- Supports setting String attributes via query string
- Not usable within Salesforce1 app

Components in Salesforce1

- Usable as Lightning Tabs
  - Appear in Salesforce1 navigation menu

```
<aura:component implements="force:appHostable">
```

- Lightning App Builder (Pilot Spring ‘15)
- Lightning Extensions (Pilot)
  - Override tabs/fields within Salesforce1

Additional Resources

- Lightning Components Trailhead
  - https://developer.salesforce.com/trailhead/module/lightning_components
- Lightning Components Developer Guide
  - https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/
- Salesforce Developer Relations Blog
  - https://developer.salesforce.com/blogs/developer-relations/
- Aura Framework
  - https://github.com/forcedotcom/aura

Thanks!