

## Overview

The Lightning Component framework is designed to help you build responsive UIs for Force.com Apps. It uses JavaScript on the client side and Apex on the server side. This framework is based on the Aura open-source framework.

## Getting Started

1. To enable Lightning components in your Developer Edition organization, from Setup, click **Develop > Lightning Components**. Select the **Enable Lightning Components** checkbox.
2. To create a Lightning app, use the Developer Console. Click **Your Name > Developer Console**. In the menu bar, click **File > New > Lightning Application**.

The following example creates a Lightning app that contains a component. ns refers to your namespace.

```
<aura:application>
  <h1>Hello App</h1>
  <ns:helloComponent />
</aura:application>

<!--helloComponent.cmp-->
<aura:component>
  <h1>Hello Lightning!</h1>
</aura:component>
```

<aura:component> can contain HTML or other Lightning components.

## Component Bundles

A bundle contains a component or app and all its related resources. To create the resources, click the resource buttons on the component sidebar in the Developer Console.

Client-side Controller	Actions for the component or app
Helper	Shared JavaScript code
Renderer	Custom JavaScript renderer
Styles	CSS declarations

## Expressions

Use the `{!...}` syntax to evaluate an expression. For example,

`{!c.change}` calls a client-side controller and `{!v.myText}` refers to a component attribute value. `{!v.body}` outputs the body of the component, which is an array of components.

## Client-Side Controllers

Controller functions, also known as actions, handle user interactions. This component displays a string and a button, which when pressed updates the string value.

```
<aura:component>
  <aura:attribute name="myText" type="String"
    default="A string waiting to change"/>
  <v.myText>
  <ui:button label="Go" press="{!c.change}"/>
</aura:component>
```

Here's the controller action:

```
change : function(cmp, event, helper) {
  cmp.set("v.myText", "new string");
  helper.doSomething(cmp);
}
```

The helper resource takes the following form:

```
doSomething : function(cmp, myObj) {
  //Do something else here
}
```

## App Events

App events communicate data between components. The event contains the attributes you are passing.

```
<aura:event type="APPLICATION">
  <aura:attribute name="myObj"
    type="namespace.MyObj__c"/>
</aura:event>
```

The event is fired in your JavaScript code.

```
update : function(cmp, event, helper) {
  var myObj = cmp.get("v.myObj");
  var myEvent = $A.get("e.namespace.theEvent");
  myEvent.setParams({ "myObj": myObj }).fire();
}
```

In the handling component, add this handler:

```
<aura:handler event="namespace.theEvent"
  action="{!c.updateEvent}"/>
```

Handle the event in a client-side controller.

```
updateEvent : function(cmp, event, helper) {
  helper.updateObj(cmp, event.getParam("myObj"));
}
```

## Invoke An Action on Component Initialization

Add the init handler and handle the event in the doInit action in your client-side controller.

```
<aura:handler name="init" value="{!this}" action="{!c.doInit}"/>
```

## CSS

Use the `.THIS` selector with your CSS declarations to prevent styling conflicts. `.THIS` is replaced by your component name at runtime.

```
.THIS h1 {
  padding-left: 40px;
}
```

## Static Resources

Place resources in a .zip file and upload to Static Resources. Then, use the `<link>` tag within the `<aura:application>` tag.

```
<link href="/resource/path/to/myCSS" rel="stylesheet"/>
```

## Find Component by ID

Use `aura:id` to set a local ID for a component.

```
<ui:button aura:id="button1" label="button1"/>
```

Find the button component by calling `cmp.find("button1")`, where `cmp` is a reference to the component containing the button.

## Common JavaScript Functions

These are common functions for components and events:

Get value	<code>cmp.get("v.myString");</code>
Set value	<code>cmp.set("v.myString", "some string");</code>
Get event parameter	<code>myEvt.getParam("myAttr");</code>
Set event parameters	<code>myEvt.setParams({ "myAttr" : myVal })     .fire();</code>

## Core Elements

These are common elements in a Lightning app:

<aura:application>	Root element for a .app resource.
<aura:attribute>	Defines an attribute for an app, component, interface, or event.
<aura:component>	A component that can be reused in apps or other components.
<aura:event>	Defines a component or app event that is fired from a JavaScript controller action.
<aura:iteration>	Iterates over a collection of items and renders the body of the tag for each item.
<aura:if>	Renders content within the tag if a specified condition is true.
<aura:renderIf>	Renders content within the tag if a specified condition is true. Preferred to <aura:if> if a server trip is required to instantiate the components that aren't initially rendered.
<aura:set>	Sets the value of an attribute on a component reference.
<aura:text>	Renders plain text and escapes the {! syntax.

## Core Form

These components are used in forms:

<ui:button>	An HTML button element used to submit a form or execute an action.
<ui:inputCheckbox>	An HTML input element of type checkbox.
<ui:inputDate>	An HTML input element for entering a date.
<ui:inputDateTime>	An HTML input element for entering a date and time.
<ui:inputEmail>	An HTML input element of type email.
<ui:inputNumber>	An HTML input element for entering a number.
<ui:inputPhone>	An HTML input element for a phone number.
<ui:inputText>	An HTML input element of type text.

## Core Output

These components are used for outputting values:

<ui:outputCheckbox>	Displays a read-only checkbox in a checked or unchecked state.
<ui:outputDate>	Displays a date based on locale.
<ui:outputDateTime>	Displays a date and time based on locale.
<ui:outputEmail>	Displays an email address in an HTML anchor element.
<ui:outputNumber>	Displays a number based on locale.
<ui:outputPhone>	Displays a phone number in an HTML anchor element.
<ui:outputText>	Displays a string of text.

## Common \$A Methods

The Aura object is the top-level object in the JavaScript framework code. \$A is shorthand for Aura.

\$A.get()	Returns an app-level event.
\$A.enqueueAction()	Queues an action for batch execution.
\$A.log()	Logs an error.
\$A.newCmpAsync()	Dynamically creates a component.
\$A.run()	Runs code in external JavaScript.

## Load or Save Data with Apex Controllers

All methods on server-side controllers must be static. Only methods explicitly annotated with @AuraEnabled are available.

This controller has a method to return a list of opportunities.

```
public class OpportunityController {
    @AuraEnabled
    public static List<Opportunity> getOpportunities() {
        List<Opportunity> opportunities =
            [SELECT Id, Name, CloseDate FROM Opportunity];
        return opportunities;
    }
}
```

## Wiring a Component to a Controller

Add a controller system attribute to the <aura:component> tag to wire a component to an Apex controller. For example:

```
<aura:component controller="myNamespace.MyApexController" >
```

## Calling an Apex Controller

Apex controllers are called from client-side controllers. The getOpps client-side controller action calls the getOpportunities Apex controller action.

```
"getOpps" : function(component) {
    var a = component.get("c.getOpportunities");
    // Create a callback that is executed after
    // the server-side action returns
    a.setCallback(this, function(action) {
        if (action.getState() === "SUCCESS") {
            alert(action.getReturnValue());
        } else {
            alert(action.getState());
        }
    });
    // Add the Apex action to the queue
    $A.enqueueAction(a);
}
```

## Working with Components

To reference a component, use Cmp.<myNamespace>.<myComponent>

For example:

```
Cmp.ui.button button = new Cmp.ui.button(label = 'Click Me');
```

## Integration with the Salesforce1 Mobile App

To add a Lightning component to Salesforce1 Mobile App, use this markup:

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

Implementing appHostable interface make a component available for Salesforce1 Mobile App. To activate it, create a custom Lightning Component tab for the component and include the tab in the Mobile navigation menu.