Lightning Web Component (LWC)

By

Rupom Chakraborty

Sr.Solution Architect

Shyamla Plaza, Behind Mythrivanam,
Ameerpet , Hyderabad, Telangana State, India
Ph: 86 86 86 42 86

Note

The prerequisite of this LWC course material is that, you must have knowledge on Salesforce Lightning Development and fair exposure to all the concepts of AURA Development.

Salesforce Lightning Web Component -LWC Introduction

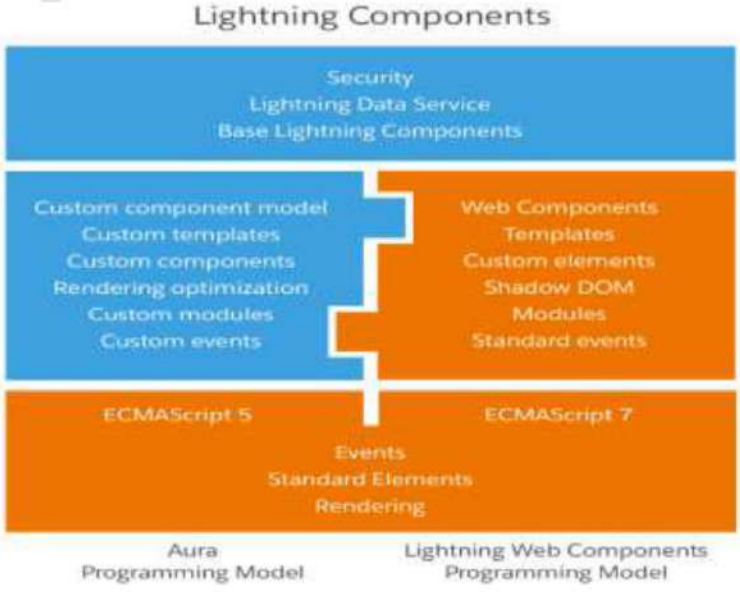
LWC Definition

Lightning Web Components (LWC) is a stack of modern lightweight frameworks built on the latest web standards.Lightning web components are custom HTML elements built using HTML and modern JavaScript.

LWC Technology

Lightning Web Components consist of **four separate technologies:**

- **Custom Elements:** The HTML elements with custom templates, tag names like and behaviours are made with a set of JavaScript APIs. HTML Living Standard specification has definitions of these Custom Elements.
- **Shadow DOM:** We use this for isolating CSS and JavaScript which could be like. Living Standard DOM specification has the definitions of this.
- **HTML templates:** User-defined templates in HTML are rendered only when called upon. HTML Living Standard specification has the definition of the tag.
- **ES Modules:** The ES Modules specification defines the inclusion and reuse of JS documents in a standard based, modular, performant way.



RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Advantage/Characteristics

- **Easy to Learn:** LWC is basically takes the form through native web standards that is in the browser. It means that no added abstraction layer like Aura Framework or any other framework, we only need standard JavaScript to develop.

- **Better Performance:** Because of the no added abstraction layer, LWC is likely to render faster than aura components since performance is important to deliverability.

- **Faster loading sites:** Like lightning, LWC is faster in loading the developed components than Aura Components and is lightweight framework which is built on web standards.

- **More Standards, less proprietary:** LWC has built-in browser security features from Web Components Standards, so the usage of out-of-the-box is more and less of customization. We all know that Aura is proprietary so with LWC, the more we know about web standards; more we'll have the skill that can be used in other technologies as well.

- **Common Components/Service Components:** We can now write components that have no User Interface in LWC, and those components can be reused in other components which are more efficient than Static Resources.

- **Easier to ramp for developers:** No additional framework is needed to learn in order to develop LWC and hence transition for the developers is a lot easier.

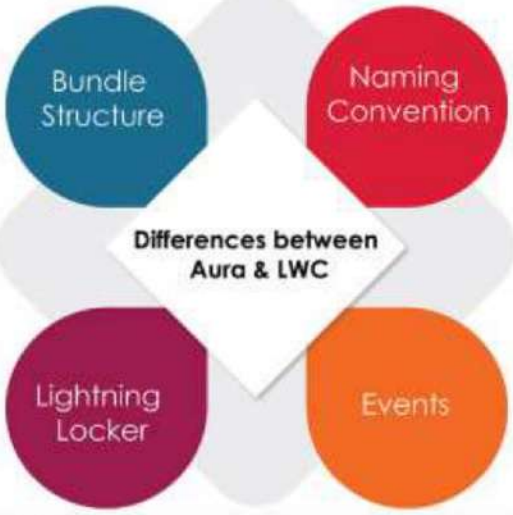
- **Better security, better testing and better browser compatibility:** With LWC, CSS, Script and DOM Isolation are better and has more limited event scope. With each of these we have more consistency in designing Components. LWC also supports Two-way data binding with which we can coordinate how data moves between Components.

- **Components Reusability:** LWC components are completely reusable entity providing efficient way of component handling structure.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Difference Between LWC & AURA (LWC Vs AURA)

- **Bundle structure:** LWC requires manual creation of a folder to host all your component files. Component HTML, JS, Configuration files are mandatory. CSS and SVG are optional.
- **Naming Convention:** The only noted difference in naming is while you're referring component in another component. The new syntax uses kebab-case, instead of the camelCase (<c-hellowrold> instead of <c:hello world>) and the component must have a closing tag.
- **Events:** Unlike component or application events in Lightning Component, LWC used Standard DOM events.
- **Lightning Locker:** Lightning Locker is enabled by default in all custom LWC.



LWC Supported Features/Elements

Supported Features

- Lightning Experience
- Salesforce App
- Lightning Communities
- Lightning App Builder
- Community Builder
- Standalone Apps
- Lightning Components for Visualforce

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

- Lightning Out (beta)
- Custom Tabs
- Utility Bars
- Flows
- First-Generation Managed Packages
- Second-Generation Managed Packages
- Unlocked Packages
- Unmanaged Packages
- Change Sets
- Metadata API—LightningComponentBundle
- Tooling API—LightningComponentBundle, LightningComponentResource
- EMP API
- Embedded Service Chat
- Gmail and Outlook integration

Unsupported Features

- Salesforce Console APIs (Navigation Item API, Workspace API, Utility Bar API)
- URL Addressable Tabs
- Conversation Toolkit API, Omni Toolkit API, Quick Action API
- Standard Action Overrides, Custom Actions, Global Actions, List View Actions, Related List View Actions
- Chatter Extensions

Supported JavaScript

Lightning Web Components JavaScript support includes:

- ES6 (ECMAScript 2015)
- ES7 (ECMAScript 2016)
- ES8 (ECMAScript 2017)—excluding Shared Memory and Atomics
- ES9 (ECMAScript 2018)—including only Object Spread Properties (not Object Rest Properties)

How to Choose - Lightning Web Components OR Aura

Lightning web components perform better and are easier to develop than Aura components. However, when you develop Lightning web components, you also may need to use Aura, because LWC doesn't yet support everything that Aura does.

Always choose Lightning Web Components unless you need a feature that isn't supported.

LWC Environment Setup

01. Download and Install SFDX CLI from https://developer.salesforce.com/tools/sfdxcli

02. Download and Install MS Visual Code from https://code.visualstudio.com/

03. Open Visual Code and Install Salesforce Extension Pack

Create a Project

01. Open Visual Studio Code.
02. Press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux, then type **create project**. Select SFDX: Create Project, and press **Enter**.
03. Leave the default project type selection **Standard** as is, and press **Enter**.
04. Enter **LWC_Project** as project name, and press **Enter**.
05. Choose a directory on your local machine where the project will be stored. Click **Create Project.**

Authorize Your Dev Hub

01. In Visual Studio Code, press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux.
02. Type **sfdx**.
03. Select **SFDX: Authorize a Dev Hub**.
04. Log in using your Dev Hub org credentials.
05. Click **Allow**.

Create a Scratch Org

01. In Visual Studio Code, press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux.
02. Type **sfdx**.
03. Select **SFDX: Create a Default Scratch Org**....

04. Press **Enter** to accept the default project-scratch-def.json.

05. Press **Enter** to accept the default trailhead scratch org alias.

06. Press **Enter** to accept the default 7 days scratch org duration.

07. Be patient, creating a scratch org can take a minute.

Create a Lightning Web Component

01. Open **Visual Studio Code.**

02. Press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux, then type **Create Lightning Web Component**. Press **Enter**.

03. Enter the **Name lwc_1711_eg1_XXX** . Press **Enter.**

04. Press **Enter** to accept the default project folder

05. Be patient, creating a LWC can take a minute.

Create a Lightning Web Component using CLI Terminal

01. Open Visual Studio Code.

02. Press **Command + Shift + P** on macOS or **Ctrl + Shift + P** on Windows or Linux, then type **focus terminal**. Press **Enter**.

03. Enter **sfdx force:lightning:component:create -n lwc_1711_eg1_XXX -d force-app/main/default/lwc -- type lwc**, and confirm with **Enter**.

These are the parameters you used in the command.

-n — This defines the name of the Lightning web component folder and its files.

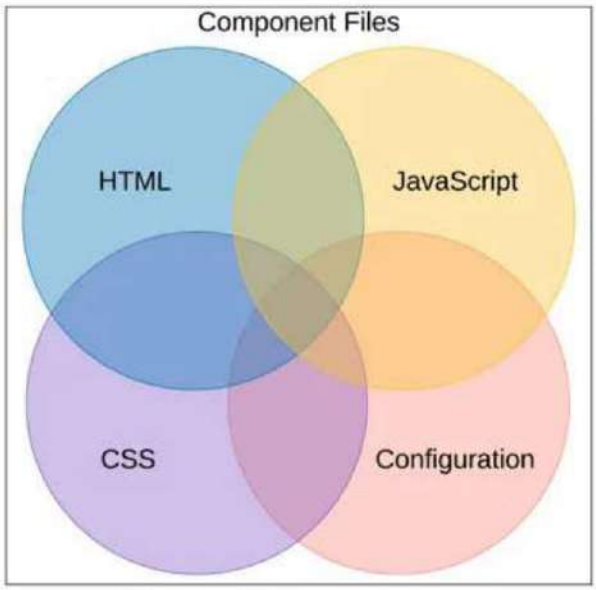
-d — This defines the target directory where the Lightning web component should be created. The target directory must be named lwc

--type — This specifies that you want to create a Lightning Web Component.

LWC Component Structure

LWC Bundle consist of 4 Elements:

1. Component HTML File
2. Component JavaScript File
3. Component Configuration File
4. Component CSS File

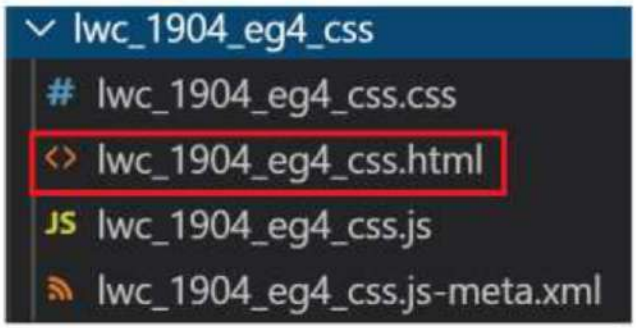


Component HTML File

Every UI component must have an HTML file with the root tag **<template>.**

The HTML file follows the naming convention **<component>.**html.

Eg: **lwc_1711_eg1_XXX.html**



RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Use :

HTML file is used to design/build the UI part of LWC

Component JavaScript File

Every component must have a JavaScript file. If the component renders UI, the JavaScript file defines the HTML element.

- The JavaScript file follows the naming convention <*component*>.js
  Eg: : **lwc_1711_eg1_XXX.js**
- JavaScript files in Lightning web components are ES6 modules. By default, everything declared in a module is local—it's scoped to the module.
- To import a class, function, or variable declared in a module, use the import statement.

Use:

- Js file is used to write function, Logic
- Js file is Client-Side Controller.
- Js file is to Import class, function, variable etc.

Component Configuration File

Every component must have a configuration file. The configuration file defines the metadata values for the component, including the design configuration for Lightning App Builder and Experience Builder.

- The configuration file follows the naming convention <component>.js-meta.xml, such as **lwc_1711_eg1_XXX.js-meta.xml.**
- Include the configuration file in your component's project folder, and push it to your org along with the other component files.

Use:

- Js-meta.xml file is used to control the exposure of LWC component.

- Js-meta.xml file is used to control the target location of the component usage.

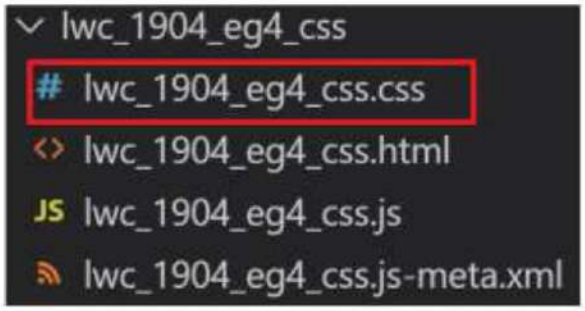
Component CSS File

A component can include a CSS file (Optional).

- The CSS file follows the naming convention <*component*>.css
  Eg: : **lwc_1711_eg1_XXX.css**

Use:

- Use to define custom style for a component.
- Supports CSS.



RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Decorators

The Lightning Web Components programming model has three decorators that add functionality to a property or function.

- **@api**

  This defines a variable/method or property as Public.

  Public properties are reactive. If the value of a public property changes, the component rerenders. To expose a public property, decorate a field with @api.

  Public properties define the API for a component.

  When we want to expose a public property, we decorate it with @api.
  Parent component can make use of the Public Property.

  A component that declares a public property can set only its default value.

- **@track**

  This defines and captures reactive fields. Fields where data changes (Dynamic).

  Fields are reactive. If a field's value changes, and the field is used in a template or in a getter of a property that's used in a template, the component rerenders and displays the new value.

  Private reactive property. Private Property can be used only in the component where it is defined.

- **@wire**

  This is used to call inbuilt Data Services.

  To read Salesforce data, Lightning web components use a reactive wire service. When the wire service provisions data, the component rerenders. Components use @wire in their JavaScript class to specify a wire adapter or an Apex method.

  @wire is used to read Salesforce data.

  It is reactive.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Component Creation (Practical)

Creating HTML file

Eg1 : Creating LWC Component with Buttons

lwc_1904_eg2_buttons.js | lwc_1904_eg2_buttons.js-meta.xml | lwc_1904_eg2_buttons.html

force-app > main > default > lwc > lwc_1904_eg2_buttons > lwc_1904_eg2_buttons.html > template > lightning-card

```
<template>
    <lightning-card title="Emp Details">

            <lightning-button label="Submit" variant="base"></lightning-button>
            <lightning-button label="Submit" variant="brand"></lightning-button>
            <lightning-button label="Submit" variant="destructive"></lightning-button>
            <lightning-button label="Submit" variant="success"></lightning-button>

            <lightning-button-icon alternative-text="Submit" icon-name="utility:alert"></lightn

            <lightning-button-menu alternative-text="Show" icon-name=utility:broadcast>
                <lightning-menu-item label="Menu1"></lightning-menu-item>
                <lightning-menu-item label="Menu2"></lightning-menu-item>
                <lightning-menu-item label="Menu3"></lightning-menu-item>
            </lightning-button-menu>

    </lightning-card>
</template>
```

Eg2 : Creating LWC Component to capture Employee Details with Buttons component created above

lwc_1904_eg2_buttons.js | lwc_1904_eg2_buttons.js-meta.xml | lwc_1904_eg3_compcall.html

force-app > main > default > lwc > lwc_1904_eg3_compcall > lwc_1904_eg3_compcall.html > template

```
<template>
    <lightning-card title="Employee Details">
        <lightning-input label="Enter Emp Name :" type="text" name="empname"></lightning-input>
        <lightning-input label="Enter Emp Age :" type="number" name="empage"></lightning-input>
        <lightning-input label="Enter Emp Data of Birth :" type="date" name="empdob"></lightning
    </lightning-card>

    <c-lwc_1904_eg2_buttons></c-lwc_1904_eg2_buttons>

</template>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Creating Complete LWC (Html , js , js-data.xml)

Eg1 : Creating LWC Calculator to calculate 2 Numbers.

lwc_2504_eg2_AddSubMul.html

```
<template>
    <lightning-card title="LWC Calculator">
        <lightning-input type="number" label="Enter Aval" name="aval" onchange={callme}></lightning-in
        <lightning-input type="number" label="Enter Bval" name="Bval" onchange={callme}></lightning-in
        <lightning-button label="Calculate" onclick={show}></lightning-button>
        <br/><br/><br/><br/>
        {resultValue}
    </lightning-card>
</template>
```

lwc_2504_eg2_AddSubMul.js

```
import { LightningElement,track } from 'lwc';

export default class Lwc_2504_eg2_AddSubMul extends LightningElement {
@track resultValue
number1;
number2;
callme(event)
{
    const evtname = event.target.name; // Name of the Element where the event has occured
    if(evtname=='aval')
    {
       this.number1=event.target.value; // get aval and assign to varaible num1
    }
    else
    {
        this.number2=event.target.value; // get aval and assign to varaible num2
    }
}
show()
{
   const n1 = parseInt(this.number1);
   const n2 = parseInt(this.number2);
   this.resultValue ='Sum of Aval: '+n1+' and Bval : '+n2+ ' is : '+(n1+n2);
}
}
```

lwc_2504_eg2_AddSubMul.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions

Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

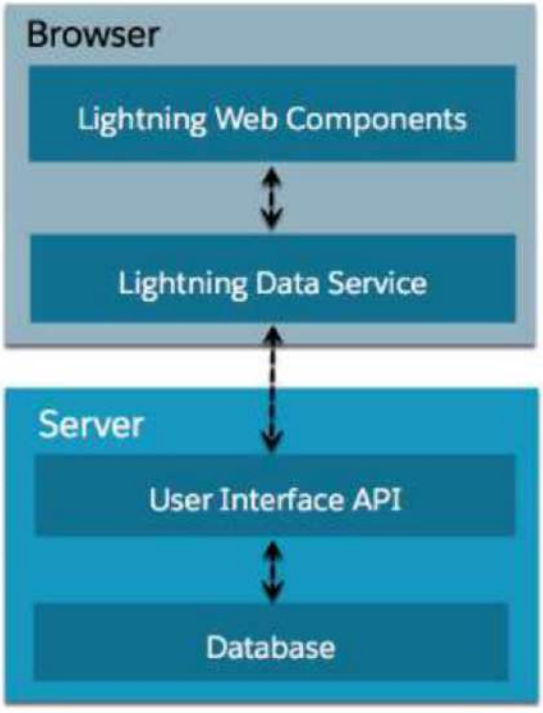
LWC Lightning Data Service (LDS)

Lightning Data Service is a centralized data caching framework which is used to load, save, create and delete a record without server-side apex code.

LDS in LWC are of three Types :

1. **Components Based**—**lightning-record-edit-form**, **lightning-record-form**, and **lightning-record-view-form**
2. **Wire adapters**
   - **Property** or
   - **Functions** in the **lightning/ui*Api** modules
3. **Imperatively Call Apex methods** imported via **@salesforce/apex/methodName**

LWC – LDS Structural Flow Diagram



Components Based

LWC – Component based LDS are of 3 Types :

Form Function	
Display, create, or edit records	*lightning:recordForm*
Display records only	*lightning:recordViewForm* (with *lightning:outputField*)
Create or edit records only	*lightning:recordEditForm* (with *lightning:inputField*)

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

lightning-record-view-form

lightning-record-view-form component to creates a form that displays Salesforce record data for specified fields associated with that record. The fields are rendered with their labels and current values as read-only.

Attributes

NAME	TYPE	ACCESS	REQUIRED	DEFAULT	DESCRIPTION
density		global			Sets the arrangement style of fields and labels in the form. Accepted values are compact, comfy, and auto (default). Use compact to display fields and their labels on the same line. Use comfy to display fields below their labels. Use auto to let the component dynamically set the density according to the user's Display Density setting and the width of the form.
object-api-name		global	✓		The API name of the object.
record-id		global	✓		The ID of the record to be displayed.

lightning-record-edit-form

lightning-record-edit-form component to creates a form that's used to add a Salesforce record or update fields in an existing record. The component displays fields with their labels and the current values, and enables you to edit their values.

lightning-record-edit-form supports the following features.

- Editing a record's specified fields, given the record ID.
- Creating a record using specified fields.
- Customizing the form layout
- Custom rendering of record data

Attributes

NAME	TYPE	ACCESS	REQUIRED	DEFAULT	DESCRIPTION
density		global			Sets the arrangement style of fields and labels in the form. Accepted values are compact, comfy, and auto (default). Use compact to display fields and their labels on the same line. Use comfy to display fields below their labels. Use auto to let the component dynamically set the density according to the user's Display Density setting and the width of the form.
field-names		global			Reserved for internal use. Names of the fields to include in the form.
form-class		global			A CSS class for the form element.
layout-type		global			The type of layout to use to display the form fields. Possible values: Compact, Full.
object-api-name		global	✓		The API name of the object.
record-id		global			The ID of the record to be displayed.
record-type-id		global			The ID of the record type, which is required if you created multiple record types but don't have a default.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

lightning-record-form

lightning-record-form component quickly creates forms to add, view, or update a record.

Using this component to create record forms is easier than building forms manually with lightning-record-edit-form or lightning-record-view-form.

The lightning-record-form component provides these helpful features:

- Switches between view and edit modes automatically when the user begins editing a field in a view form
- Provides Cancel and Save buttons automatically in edit forms
- Uses the object's default record layout with support for multiple columns
- Loads all fields in the object's compact or full layout, or only the fields you specify

Attributes

NAME	TYPE	ACCESS	REQUIRED	DEFAULT	DESCRIPTION
columns		global			Specifies the number of columns for the form.
density		global			Sets the arrangement style of fields and labels in the form. Accepted values are compact, comfy, and auto (default). Use compact to display fields and their labels on the same line. Use comfy to display fields below their labels. Use auto to let the component dynamically set the density according to the user's Display Density setting and the width of the form.
fields		global			List of fields to be displayed. The fields display in the order you list them.
layout-type		global			The type of layout to use to display the form fields. Possible values: Compact, Full. When creating a new record, only the full layout is supported.
mode		global			Specifies the interaction and display style for the form. Possible values: view, edit, readonly. If a record ID is not provided, the default mode is edit, which displays a form to create new records. If a record ID is provided, the default mode is view, which displays field values with edit icons on updateable fields.
object-api-name		global	✓		The API name of the object.
record-id		global			The ID of the record to be displayed.
record-type-id		global			The ID of the record type, which is required if you created multiple record types but don't have a default.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Wire adapters

- Wire is a reactive service to read Salesforce data.
- It is Built on Lightning Data service.
- Components use @wire in their JavaScript class to read data from one of the wire adapters in the lightning/ui*Api namespace.

wire service syntax:

```
import { adapterId } from 'adapterModule';
@wire(adapterId, adapterConfig)
propertyOrFunction;
```

- **adapterId (Identifier)** ---> The identifier of the wire adapter.
- **adapterModule (String)** —>The identifier of the module that contains the wire adapter function, in the format namespace/moduleName.
- **adapterConfig (Object)** —> A configuration object specific to the wire adapter. Configuration object property values can be either strings or references to objects and fields imported from @salesforce/schema.
- **Property Or Function** —> A private property or function that receives the stream of data from the wire service.
    - If a **property** is decorated with @wire, the results are returned to the property's data property or error property.
        - **Eg:** The **wire** service either provides with the list of account to the **wiredOpportunities.data** property, or returns an error to the **wiredOpportunities.error** property.

```
import { LightningElement, api, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';

export default class Record extends LightningElement {
    @api recordId;

    @wire(getRecord, { recordId: '$recordId', fields: ['Opportunity.Name'] })
    record;
}
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

- If a **function** is decorated with @wire, the results are returned in an object with a data property and an error property.
    - **Eg:** If a function is decorated with **@wire**, the results are returned in an object with a **data** property or an **error** property.

```
import { LightningElement, api, track, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';

export default class WireFunction extends LightningElement {
   @api recordId;
   @track record;
   @track error;

   @wire(getRecord, { recordId: '$recordId', fields: ['Opportunity.Name'] })
   wiredAccount({ error, data }) {
      if (data) {
         this.record = data;
      } else if (error) {
         this.error = error;
      }
   }
}
```

- **To use apex methods via @wire, you must annotate it with cacheable=true.**

Wire with dynamic Parameters:

- A parameter with $ to indicate that it's dynamic and reactive.
- It refers to a property of the component instance.
- If its value changes, the template rerenders.

Import references to salesforce Object and fields

```
import objectName from '@salesforce/schema/object';
```

Eg:

- import PROPERTY_OBJECT from '@salesforce/schema/Property__c';
- import ACCOUNT_OBJECT from '@salesforce/schema/Account';

To import a reference to a field, use this syntax.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

import FIELD_NAME from '@salesforce/schema/object.field';

Eg:

- import POSITION_LEVEL_FIELD from '@salesforce/schema/Property__c.Name';
- import ACCOUNT_NAME_FIELD from '@salesforce/schema/Account.Name';

To import a reference to a field via a relationship, use this syntax.

import SPANNING_FIELD_NAME from '@salesforce/schema/object.relationship.field';

Eg:

- import ACCOUNT_OWNER_NAME_FIELD from '@salesforce/schema/Account.Owner.Name';

Using lightning/ui*Api Wire Adapters

Wire adapters and JavaScript functions in these modules are built on top of Lightning Data Service (LDS) and User Interface API. Use these wire adapters and functions to work with Salesforce data and metadata . Here is are the some of the useful wire adapters

getListUi

Use this wire adapter to get the records and metadata for a list view . For Example, the following wire adapter gets the allaccounts list view data.

```
@wire(getListUi, { objectApiName: ACCOUNT_OBJECT, listViewApiName: 'AllAccounts' })
```

getObjectInfo

Use this wire adapter to get metadata about a specific object. The response includes metadata describing fields, child relationships, record type, and theme. The following example shows how to get the account object data using the wire adapter .

```
@wire(getObjectInfo, { objectApiName: ACCOUNT_OBJECT })
```

getPicklistValues

Use this wire adapter to get the picklist values for a specified field. the following wire adapter shows how to get the picklist value .

```
@wire(getPicklistValues, { recordTypeId: '012000000000000AAA', fieldApiName:
INDUSTRY_FIELD })
```

getPicklistValuesByRecordType

Use this wire adapter to get the values for every picklist of a specified record type.

```
@wire(getPicklistValuesByRecordType, { objectApiName: ACCOUNT_OBJECT, recordTypeId: '012000000000000AAA' })
```

getRecord

Use this wire adapter to get a record's data.

```
@wire(getRecord, { recordId: '001456789012345678', fields: [NAME_FIELD] })
```

getRecordUi

Use this wire adapter to get layout information, metadata, and data to build UI for one or more records.

```
@wire(getRecordUi, { recordIds: ['001456789012345678', '00145678908765432 1'], layoutTypes: ['Full'], modes: ['View'] })
```

Call an Apex Method Imperatively

- No need to set the Apex backend to *cacheable=true*.
- Instead of resulting a data-error object, it results a *Promise*.
- The idiom above can be called inside of a controller method.

When to Call Apex Method Imperatively

In the following scenarios, you must call an Apex method imperatively as opposed to using @wire.

- To call a method that isn't annotated with cacheable=true, which includes any method that inserts, updates, or deletes data.
- To control when the invocation occurs.
- To work with objects that aren't supported by User Interface API, like Task and Event.
- To call a method from an ES6 module that doesn't extend LightningElement

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC LDS Practical

Form Based

lwc_0205_eg3_lds_RecordForm.html

force-app > main > default > lwc > lwc_0205_eg3_lds_RecordForm > lwc

```
<template>
    <lightning-card title="Lightning Record Form">
        <lightning-layout>
            <lightning-layout-item size="4">
                <lightning-record-form
                  record-id={recordId}
                  object-api-name="Account"
                  layout-type="Full"
                  mode="view"
                >
                </lightning-record-form>
            </lightning-layout-item>

            <lightning-layout-item size="4">
                <lightning-record-form
                  record-id={recordId}
                  object-api-name="Account"
                  layout-type="Full"
                  mode="edit"
                >
                </lightning-record-form>
            </lightning-layout-item>

            <lightning-layout-item size="4">
                <lightning-record-form
                  record-id={recordId}
                  object-api-name="Account"
                  layout-type="Full"
                  mode="readonly"
                >
                </lightning-record-form>
            </lightning-layout-item>
        </lightning-layout>
    </lightning-card>
</template>
```

- lwc_0205_eg1_lds_RecordViewForm
  - lwc_0205_eg1_lds_RecordViewForm.html
  - lwc_0205_eg1_lds_RecordViewForm.js
  - lwc_0205_eg1_lds_RecordViewForm.js-meta.xml
- lwc_0205_eg2_lds_RecordEditForm
  - lwc_0205_eg2_lds_RecordEditForm.html
  - lwc_0205_eg2_lds_RecordEditForm.js
  - lwc_0205_eg2_lds_RecordEditForm.js-meta.xml
- lwc_0205_eg3_lds_RecordForm
  - lwc_0205_eg3_lds_RecordForm.html
  - lwc_0205_eg3_lds_RecordForm.js
  - lwc_0205_eg3_lds_RecordForm.js-meta.xml

lwc_0205_eg3_lds_RecordForm.html ● | lwc_0205_eg3_lds_RecordForm.js ×

force-app > main > default > lwc > lwc_0205_eg3_lds_RecordForm > lwc_0205_eg3_lds_RecordForm.js

```
import { LightningElement,api } from 'lwc';

export default class Lwc_0205_eg3_lds_RecordForm extends LightningElement {
    @api recordId ="0012x0000047v0EAAQ";
}
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Wire Adapter: LWC for Searching a Contact using Wire

lwc_0905_eg1_WireSearchContact.html ● | lwc_0905_eg1_WireSearchContact.js ● | lwc_0905_eg1_Wire

force-app > main > default > classes > ContactSearch.cls

```
public with sharing class ContactSearch
{
    @AuraEnabled(cacheable=true)
    public static List<Contact> findcontacts(String searchKey)
    {
        String key = '%'+  searchKey +'%';
        Return [select id,Name,Email,Phone from Contact where FirstName like :key];
    }
}
```

lwc_0905_eg1_WireSearchContact.html ● | lwc_0905_eg1_WireSearchContact.js-meta.xml

force-app > main > default > lwc > lwc_0905_eg1_WireSearchContact > lwc_0905_eg1_WireSearchContact.html > ...

```
<template>
    <lightning-card title="Wire SSC with Params to Search contact by FirstName" icon-name="custom:custom43">
        <lightning-input type="search" label="Enter The Search Text" onchange={handleIsearch} value={searchKey}></lightning

        <template if:true={contacts.data}>
            <template for:each={contacts.data} for:item="contact">
                <p key={contact.id}> {contact.Name} ======> {contact.Email} ======> {contact.Phone} </p>
            </template>
        </template>

        <template if:true={contacts.error}>
            errors = {contacts.error};
        </template>
    </lightning-card>
</template>
```

lwc_0905_eg1_WireSearchContact.html ● | lwc_0905_eg1_WireSearchContact.js ●

force-app > main > default > lwc > lwc_0905_eg1_WireSearchContact > lwc_0905_eg1_WireSearchContact.js >

```
import { LightningElement,wire,track } from 'lwc';
import findcontacts from '@salesforce/apex/ContactSearch.findcontacts';

export default class Lwc_0905_eg1_WireSearchContact extends LightningElement
{

  @track searchKey;
  @wire(findcontacts,{searchKey: '$searchKey'})contacts;


  handleIsearch(event)
  {
    this.searchKey = event.target.value;
  }
}
```

lwc_0905_eg1_WireSearchContact.html ● | lwc_0905_eg1_WireSearchContact.js ● | lwc_0905_eg1_WireSearc

force-app > main > default > lwc > lwc_0905_eg1_WireSearchContact > lwc_0905_eg1_WireSearchContact.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Apex Method Imperatively – Create LWC Component to create contact

lwc_0905_eg2_ImperativelyCreateCon.html

force-app > main > default > lwc > lwc_0905_eg2_ImperativelyCreateCon > lwc_0905_eg2_ImperativelyCreateCon.html > ...

```
<template>
    <lightning-card title="Create Contact with SSC Imperatively" icon-name="custom:custom43" >
        <lightning-input label="Enter The First Name" onchange={callFName}></lightning-input>
        <lightning-input label="Enter The Last Name" onchange={callLname}></lightning-input>
        <lightning-input label="Enter The Phone Name" onchange={callPhone}></lightning-input>

        <lightning-button label="Create Contact" variant="success" onclick={createcontactssss}></lightning-button>

    </lightning-card>
</template>
```

0905_eg2_ImperativelyCreateCon.html ● JS lwc_0905_eg2_ImperativelyCreateCon.js ●

pp > main > default > lwc > lwc_0905_eg2_ImperativelyCreateCon > JS lwc_0905_eg2_ImperativelyCreateCon

```
import { LightningElement,track,wire } from 'lwc';
import CreateCon from '@salesforce/apex/CreateContact.CreateCon';

export default class Lwc_0905_eg2_ImperativelyCreateCon extends LightningElement
{
    @track Fname;
    @track Lname;
    @track Phone;

    callFName(event)
    {
        this.Fname = event.target.value;
    }
    callLname(event)
    {
        this.Lname = event.target.value;
    }
    callPhone(event)
    {
        this.Phone = event.target.value;
    }

    createcontactssss(event)
    {
        CreateCon({
                    fname: this.Fname,
                    lname: this.Lname,
                    phone: this.Phone
                })
        .then(abc =>{
                        alert(abc);
                    })
        .catch(error=>{
                        alert(error.body.message);
                    });
    }
}
```

lwc_0905_eg2_ImperativelyCreateCon.html ● JS lwc_0905_eg2_ImperativelyCreateCon.js ● lwc_

rce-app > main > default > lwc > lwc_0905_eg2_ImperativelyCreateCon > lwc_0905_eg2_ImperativelyCre

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Events

A notification by the Browser , LWC Component regarding an action is called events

- Lightning web components dispatch standard DOM events.
- Events in Lightning web components are built on DOM Events,
- LWC Components can also create and dispatch custom events.

The DOM events system is a programming design pattern that includes these elements.

- An event name, called a type
- A configuration to initialize the event
- A JavaScript object that emits the event

Events Flow in LWC

❑ Create and Dispatch Events

- Create and dispatch events in a component's JavaScript class.
- To create an event, use the CustomEvent() constructor.
- To dispatch an event, call the EventTarget.dispatchEvent() method.

❑ Listen Events

- To listen for events, use an HTML attribute with the syntax **on**eventtype

❑ Handle Events

- There are two ways to listen for an event
- **Declaratively** from the component's HTML template
- **Programmatically** using an imperative JavaScript API

❑ Configure Event Propagation

- After an event is fired, it can propagate up through the DOM

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

❑ Communicate Between Components

- There are two ways to communicate between components that aren't in the same DOM tree.
- **Singleton library** that follows the publish-subscribe pattern
- **Lightning message channel.**

Create and Dispatch Events

Create Event:

- To create an event, use the **CustomEvent()** constructor.
- The **CustomEvent()** constructor has one required parameter, which is a string indicating the event type
- Event Type name should have
  - ✓ No uppercase letters
  - ✓ No spaces
  - ✓ Use underscores to separate words.
  - ✓ Don't prefix your event name with the string **on**
- Passing Data in LWC event : To pass data up to a receiving component, set a **detail** property in the CustomEvent constructor.

Syntax :

```
const eventname = new CustomEvent('eventname')

this.dispatchEvent(new CustomEvent('eventname'));
```

Dispatch Event:

- Use dispatchEvent() method to dispatch the event.

Syntax :

```
this.dispatchEvent(eventname);

this.dispatchEvent(new CustomEvent('eventname'));
```

Listen/ Handle Events

There are 2 ways to Handle(Listem LWC Event)

- **Declaratively** from the component's HTML template
  - Declare the **listener** in markup in the template of the parent(owner) component,
  - Using on keyword as prefix before the declared and dispatch event.
  - Eg : <c-lwc_1711_eg1_xxx onnotification={handleNotification}>

- **Programmatically** using an imperative JavaScript API
  - Define both the **listener** and the **handler** function in the
  - Eg : this.template.**addEventListener**(*eventname*', this.***handleNotification***);

Configure Event Propagation

After an event is fired, it can propagate up through the DOM . Define event propagation behavior using two properties on the event. Event propogation can be controlled using two properties on the event.

- **bubbles**
  A Boolean value indicating whether the event bubbles up through the DOM or not. Defaults to false.
- **composed**
  A Boolean value indicating whether the event can pass through the shadow boundary. Defaults to false.

Controlling Event Propagation

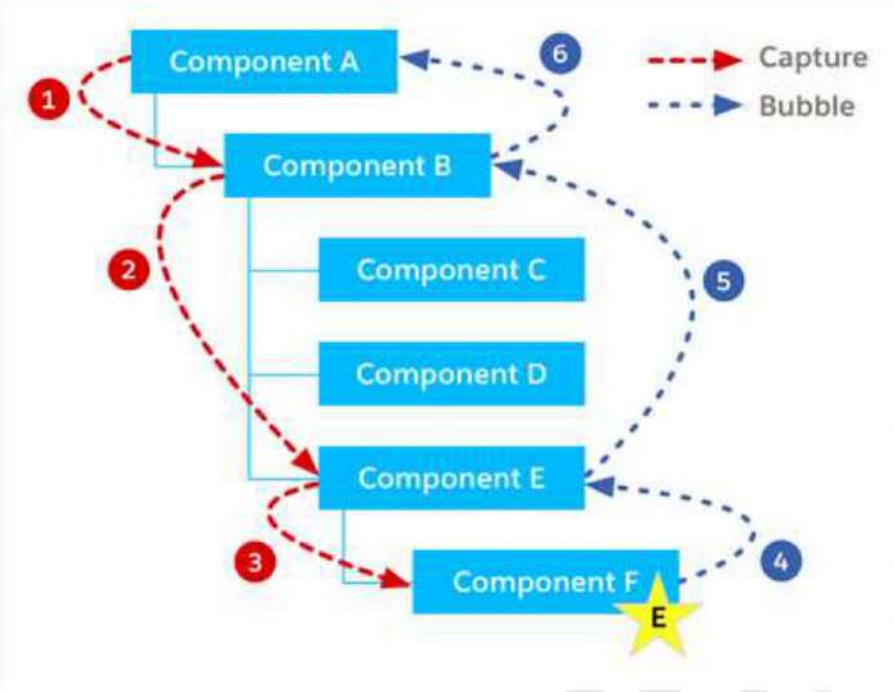
bubbles: false and composed: false
The event doesn't bubble up through the DOM and doesn't cross the shadow boundary.

bubbles: true and composed: false
The event bubbles up through the DOM, but doesn't cross the shadow boundary.

bubbles: true and composed: true

The event bubbles up through the DOM, crosses the shadow boundary, and continues bubbling up through the DOM to the document root.

bubbles: false and composed: true
Lightning web components don't use this configuration



Syntax :

```
this.dispatchEvent( new CustomEvent('notify', { bubbles: true }) );
```

Communicate Between Components

Singleton - publish-subscribe(PubSub) :

In a publish-subscribe pattern

- One component publishes an event.
- Other components subscribe to receive and handle the event.
- Every component that subscribes to the event receives the event.
- Used for Application Event
- Restricted to single page/app

Lightning message Service(Beta):

Use to communicate between components within a single Lightning page or across multiple pages.

- **Create a Message Channel**

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

To create a Lightning message channel, use the LightningMessageChannel metadata type.

- **Publish on a Message Channel**
  To publish messages on a message channel from a Lightning web component**, include the @salesforce/messageChannel** scoped module in your component's JavaScript file and **call the Lightning message service's publish() function.**

- **Subscribe and Unsubscribe from a Message Channel**
  To subscribe and unsubscribe from messages on a message channel from a Lightning web component, **include the @salesforce/messageChannel** scoped module in your component's JavaScript file and **use the Lightning message service's subscribe() and unsubscribe() functions.**

LWC Lifecycle Hooks

Lightning web components have a lifecycle managed by the framework.

- The framework creates components,
- inserts them into the DOM
- renders them,
- and removes them from the DOM.
-

- **Run Code When a Component Is Created**
  The constructor() method fires when a component instance is created. Don't add attributes to the host element during construction. You can add attributes to the host element in any other lifecycle hook.

These requirements from the HTML: Custom elements spec apply to the constructor().

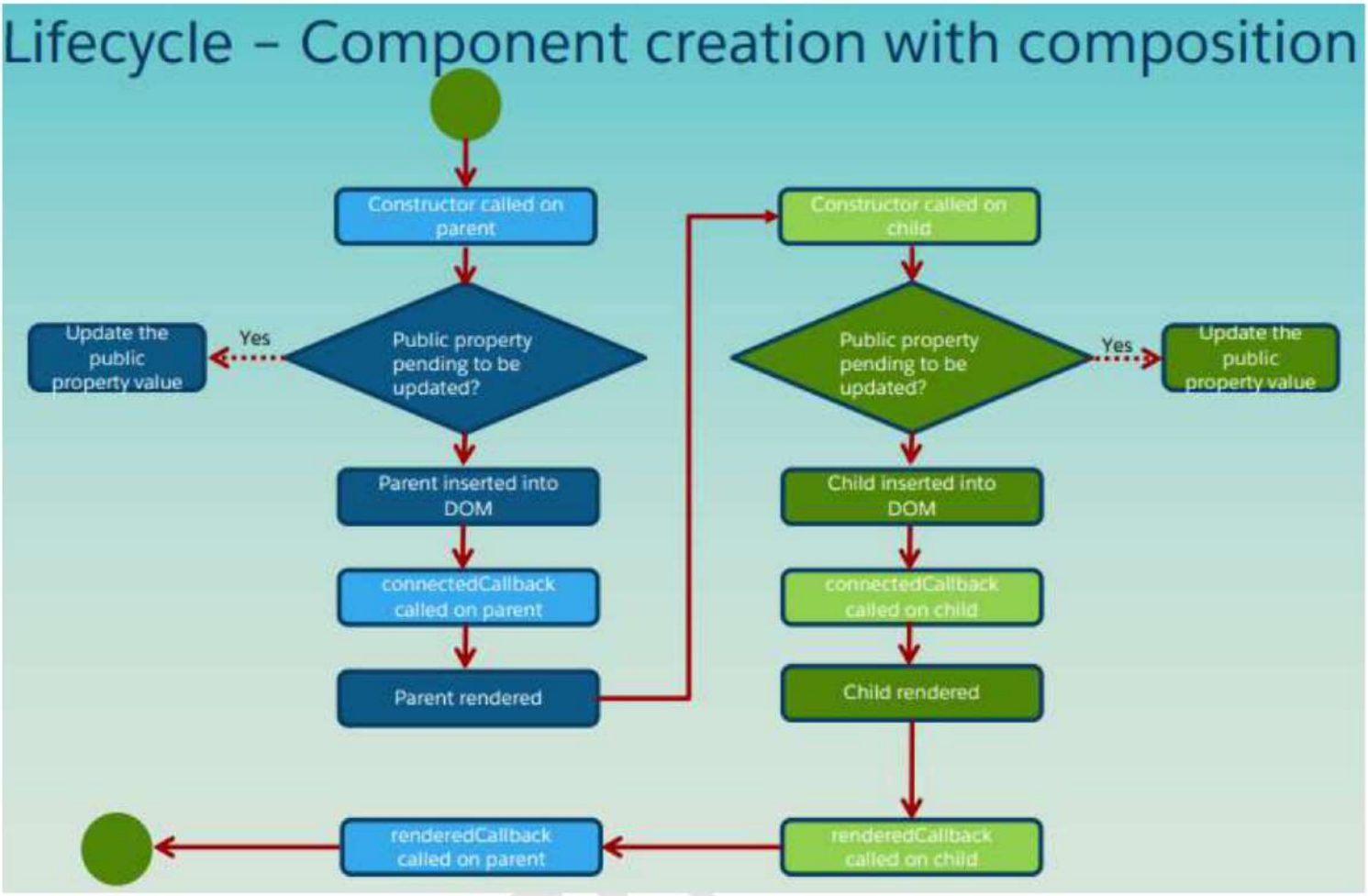
- The first statement must be super() with no parameters. This call establishes the correct prototype chain and value for this. Always call super() before touching this.
- Don't use a return statement inside the constructor body, unless it is a simple early-return (return or return this).
- Don't use the document.write() or document.open() methods.
- Don't inspect the element's attributes and children, because they don't exist yet.
- Don't inspect the element's public properties, because they're set after the component is created.

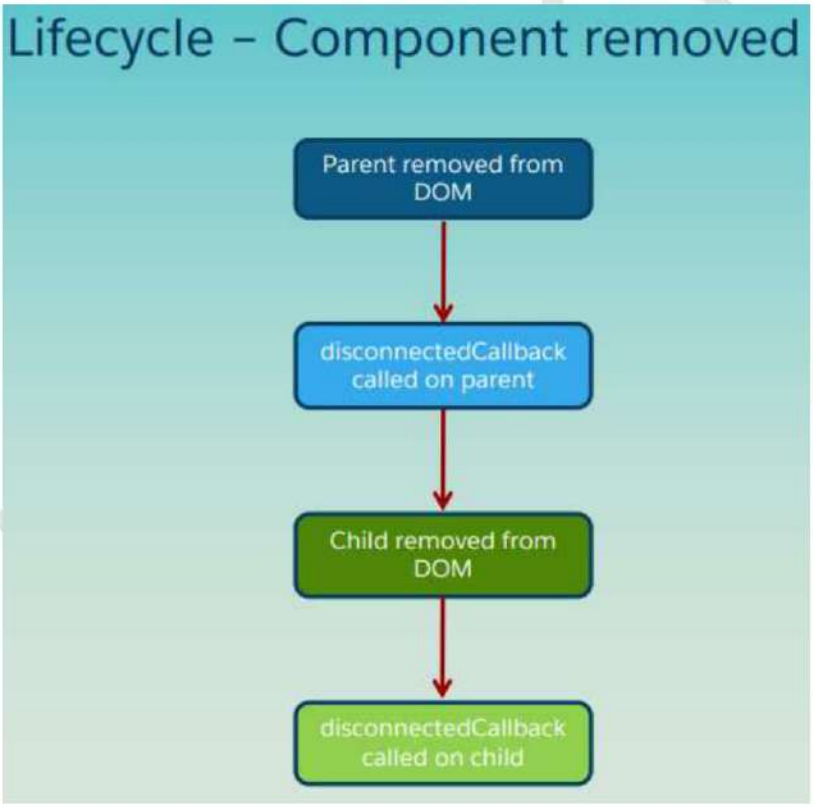
- **Run Code When a Component Is Inserted or Removed from the DOM**
  The connectedCallback() lifecycle hook fires when a component is inserted into the DOM. The disconnectedCallback() lifecycle hook fires when a component is removed from the DOM.
    - The connectedCallback() lifecycle hook fires when a component is inserted into the DOM. The disconnectedCallback() lifecycle hook fires when a component is removed from the DOM.
    - Both hooks flow from parent to child. You can't access child elements from the callbacks because they don't exist yet. To access the host element, use this.template.
    - The connectedCallback() hook can fire more than once.
- **Run Code When a Component Renders**
  The renderedCallback() is unique to Lightning Web Components. Use it to perform logic after a component has finished the rendering phase.
- **Handle Component Errors**
  The errorCallback() is unique to Lightning Web Components. Implement it to create an error boundary component that captures errors in all the descendent components in its tree. It captures errors that occur in the descendant's lifecycle hooks or during an event handler declared in an HTML template.

Triggered at a specific phase of a component instance's lifecycle

- connectedCallback() → Called when the element is inserted into document
- disconnectedCallback() → Called when an element is removed from document
- renderCallback() → Called after every render of the component
- errorCallback(error,stack) → Called when a descendant component throws an error in one of its lifecycle hooks. This hook is specific to Lightning Web Components

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286





RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Event Practical

Event Declaratively – Creating VLC player volume control

A. Creating and Dispatching Event

lwc_1005_eg1_eventeg.js ● lwc_1005_eg1_eventeg.html ●

orce-app > main > default > lwc > lwc_1005_eg1_eventeg > lwc_1005_eg1_eventeg.html > ...

```
<template>
    <lightning-card title="Volume" icon-name="utility:volume_high">
        <lightning-layout>
            <lightning-layout-item>
                <lightning-button label="Increase"  icon-name="utility:right" onclick={increasehandler}></lightning-button>
            </lightning-layout-item>

            <lightning-layout-item>
                <lightning-button label="Decrease"  icon-name="utility:left" onclick={decreasehandler}></lightning-button>
            </lightning-layout-item>
        </lightning-layout>
    </lightning-card>
</template>
```

lwc_1005_eg1_eventeg.js ● lwc_1005_eg1_eventeg.html ●

force-app > main > default > lwc > lwc_1005_eg1_eventeg > lwc_1005_eg1_eventeg.js > ...

```
import { LightningElement,api } from 'lwc';

export default class Lwc_1005_eg1_eventeg extends LightningElement {

    @api abc ='volume';

    increasehandler()
    {
        //const inc = new CustomEvent('increasesound');
        //this.dispatchEvent(inc);

        this.dispatchEvent(new CustomEvent('increasesound', { detail: 'Volume'}));
    }

    decreasehandler()
    {
        const dec = new CustomEvent('decreasesound') ;
        this.dispatchEvent(dec);
    }

}
```

lwc_1005_eg1_eventeg.js ● lwc_1005_eg1_eventeg.js-meta.xml ● lwc_1005_eg1_eventeg.ht

force-app > main > default > lwc > lwc_1005_eg1_eventeg > lwc_1005_eg1_eventeg.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

B. Listening and Handling Event

lwc_1005_eg2_eventpalyer.html

```
<template>
    <lightning-card title="VLC Player" icon-name="utility:screen">
      <p> {control} {voluumenum}</p>

      <c-lwc_1005_eg1_eventeg onincreasesound={increasethesound} ondecreasesound={decreasethesound}></c-lwc_1005_eg1_eventeg>
    </lightning-card>
</template>
```

lwc_1005_eg2_eventpalyer.js

```
import { LightningElement } from 'lwc';

export default class Lwc_1005_eg2_eventpalyer extends LightningElement {

    control;
    voluumenum = 0;
    increasethesound(event)
    {
        const A = event.detail;
        alert(A);
        this.voluumenum=   this.voluumenum + 1;
    }

    decreasethesound()
    {
        if(this.voluumenum > 0)
        {
            this.voluumenum = this.voluumenum - 1;
        }
    }

}
```

lwc_1005_eg2_eventpalyer.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Event Declaratively – Passing Data in Event

A. Child LWC – Dispatching event with Data

lwc_1005_eg3_createcon.html

```
<template>
    <lightning-card title="Create Contact with SSC Imperatively" icon-name="custom:custom43" >
        <lightning-input label="Enter The First Name" onchange={callFName}></lightning-input>
        <lightning-input label="Enter The Last Name" onchange={callLname}></lightning-input>
        <lightning-input label="Enter The Phone Name" onchange={callPhone}></lightning-input>

        <lightning-button label="Create Contact" variant="success" onclick={createcon}></lightning-button>

    </lightning-card>
</template>
```

lwc_1005_eg3_createcon.js

```
import { LightningElement,track } from 'lwc';

export default class Lwc_1005_eg3_createcon extends LightningElement {

    @track Fname;
    @track Lname;
    @track Phone;

    callFName(event)
    {
        this.Fname = event.target.value;
    }
    callLname(event)
    {
        this.Lname = event.target.value;
    }
    callPhone(event)
    {
        this.Phone = event.target.value;
    }

    createcon()
    {
       this.dispatchEvent(new CustomEvent('senddata',{ detail: {fn: this.Fname, ln: this.Lname, ph: this.Phone} }));
    }

}
```

lwc_1005_eg3_createcon.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

B. Parent LWC – Revieving Data from Child

lwc_1005_eg4_displaycon.html

force-app > main > default > lwc > lwc_1005_eg4_displaycon > lwc_1005_eg4_displaycon.html > ...

```
<template>
    <lightning-card title="Create Contact">
        <c-lwc_1005_eg3_createcon onsenddata={getdata}></c-lwc_1005_eg3_createcon>
    </lightning-card>


    <lightning-card title="Contact Created Information">
        <p> The contact First Name Entered is : {contactfirstname}</p>
        <p> The contact Last Name Entered is : {contactlastname}</p>
        <p> The contact Phone Entered is : {contactphone}</p>
    </lightning-card>
</template>
```

lwc_1005_eg4_displaycon.html | lwc_1005_eg4_displaycon.js

rce-app > main > default > lwc > lwc_1005_eg4_displaycon > lwc_1005_eg4_displaycon.js > ...

```
import { LightningElement,track } from 'lwc';

export default class Lwc_1005_eg4_displaycon extends LightningElement {

    @track contactfirstname;
    @track contactlastname;
    @track contactphone;

    getdata(event)
    {
        this.contactfirstname = event.detail.fn;
        this.contactlastname = event.detail.ln;
        this.contactphone = event.detail.ph;
    }
}
```

lwc_1005_eg4_displaycon.html | lwc_1005_eg4_displaycon.js | lwc_1005_eg4_displaycon.js

rce-app > main > default > lwc > lwc_1005_eg4_displaycon > lwc_1005_eg4_displaycon.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Event Progamatically with Event Propogation

lwc_1605_eg1_eventprogramaticaly.html

force-app > main > default > lwc > lwc_1605_eg1_eventprogramaticaly > lwc_1605_eg1_eventprogramatic

```
<template>
    <lightning-button label="Submit" onclick={callme}></lightning-button>
</template>
```

lwc_1605_eg1_eventprogramaticaly.js

rce-app > main > default > lwc > lwc_1605_eg1_eventprogramaticaly > lwc_1605_eg1_eventprogramaticaly.js > ...

```
import { LightningElement } from 'lwc';

export default class Lwc_1605_eg1_eventprogramaticaly extends LightningElement {

    callme(event)
    {
        this.dispatchEvent(new CustomEvent('nofify', { detail: {nm: 'Rupom Hi'}, bubbles: true, composed: true}));
    }
}
```

lwc_1605_eg2_eventprogramaticaly.html

force-app > main > default > lwc > lwc_1605_eg2_eventprogramaticaly > lwc_1605_eg2_eventprogramaticaly

```
<template>
    <c-lwc_1605_eg1_eventprogramaticaly></c-lwc_1605_eg1_eventprogramaticaly>
    The message passed is : {name}
</template>
```

lwc_1605_eg2_eventprogramaticaly.html | lwc_1605_eg2_eventprogramaticaly.js

rce-app > main > default > lwc > lwc_1605_eg2_eventprogramaticaly > lwc_1605_eg2_eventprogramaticaly.js >

```
import { LightningElement,api,track } from 'lwc';

export default class Lwc_1605_eg2_eventprogramaticaly extends LightningElement
{
    @track name;

     constructor()
    {
        super();
        this.template.addEventListener('nofify', this.handleNotification);
    }

    handleNotification(event)
    {
        alert('Hi The Even was captured');
        alert(event.detail.nm);
        this.name = event.detail.nm;
        console.log(name);
    }

}
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC in VF Framework

Deploying a Lightning web component using Lightning Out has a few modest requirements to ensure connectivity and security.

Verify that the remote web container, or origin server, supports these requirements.

- Ability to modify the markup served to the client browser, including both HTML and JavaScript. You must be able to add the Lightning Out markup.
- Ability to acquire a valid Salesforce session ID.
- Ability to access your Salesforce instance

LWC In VF:

- Create LWC as required
- Lightning app must use extends="ltng:outApp"
- LWC Component must be called with <aura:dependency> tag.
- Use <apex:includeLightning/> , $Lightning.createComponent() & $Lightning.use in VF to create LWC

Syntax:

```
<aura:application access="GLOBAL" extends="ltng:outApp">
  <aura:dependency resource="c:myAppComponent"/>
</aura:application>
```

LWC in VF Practical

01.Creating LWC

lwc_2604_eg1_wirecreateaccount.html

force-app > main > default > lwc > lwc_2604_eg1_wirecreateaccount > lwc_2604_eg1_wirecreateaccount.html > ...

```
<template>
    <lightning-card title="Account">
        <lightning-input type="text" label="Account Name" name="name" onchange={callName}></lightning-input>
        <lightning-input type="phone" label="Account Phone" name="name" onchange={callName}></lightning-input>
        <lightning-input type="text" label="Account Industry" name="name" onchange={callName}></lightning-input>
        <lightning-input type="text" label="Account Rating" name="name" onchange={callName}></lightning-input>

        <lightning-button label="Submit" onclick={callme}></lightning-button>
    </lightning-card>
</template>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

lwc_2604_eg1_wirecreateaccount.html ● JS lwc_2604_eg1_wirecreateaccount.js ●

rce-app > main > default > lwc > lwc_2604_eg1_wirecreateaccount > JS lwc_2604_eg1_wirecreateaccount.js > Lwc_2604_eg1_wirecreateaccou

```
import { LightningElement ,wire, track} from 'lwc';
import { createRecord } from 'lightning/uiRecordApi';
export default class Lwc_2604_eg1_wirecreateaccount extends LightningElement {
    @track name;
    @track phone;
    @track industry;
    @track rating;

    callName(event)
    {
        this.name=event.target.value;
    }
    callPhone(event)
    {
        this.phone=event.target.value;
    }
    callInd(event)
    {
        this.industry=event.target.value;
    }
    callRat(event)
    {
        this.rating=event.target.value;
    }

    callme()
    {
        const fields={'Name':this.name,'Phone':this.phone,'Industry':this.industry,'Rating':this.rating};
        const recordData = {apiName:"Account",fields};
        createRecord(recordData).then(response=>{
            console.log('Account Created Successfully with Acid: '+response.id);
            alert('Account Created Successfully with Acid: '+response.id);
        }).catch(error=>{
            console.log('Account Creation Failed '+error.body.message);
            alert('Account Creation Failed '+error.body.message);
        });
    }
}
```

lwc_2604_eg1_wirecreateaccount.html ● JS lwc_2604_eg1_wirecreateaccount.js ● lwc_2604_eg

force-app > main > default > lwc > lwc_2604_eg1_wirecreateaccount > lwc_2604_eg1_wirecreateaccount.j

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>

</LightningComponentBundle>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

02.Creating Lightning app out

LA_1705_eg2_lwc2.app *

```
<aura:application extends="ltng:outApp" >
    <aura:dependency resource="lwc_2604_eg1_wirecreateaccount" />
</aura:application>
```

03.Creating VF Page

```
<apex:page standardController="Account" >
    <apex:includeLightning />
    <apex:form >
        <apex:pageBlock title="Account Info">
            <apex:pageBlockSection title="VF Comp">
                <apex:inputField value="{!Account.name}"/>
                <apex:inputField value="{!Account.Phone}"/>
                <apex:commandButton value="Submit" action="{!Save}"/>
            </apex:pageBlockSection>
            <apex:pageBlockSection title="LWC Comp">
               <div id="two"></div>
            </apex:pageBlockSection>
        </apex:pageBlock>
    </apex:form>
    <script>
        $Lightning.use("c:LA_1705_eg2_lwc2",function()
                       {
                           $Lightning.createComponent(
                               "c:lwc_2604_eg1_wirecreateaccount",
                               {       //Blank
                               },
                               "two",
                               function(cmp)
                               {
                               }
                           );
                        });
    </script>
</apex:page>
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC Debug

Lightning web components in production mode

- Minified JavaScript
- Proxied values

Minified JavaScript

Minification means that we compress JavaScript into as few bytes as possible by removing any unnecessary characters and elements like line breaks, whitespace, tabs, code comments and so forth. This reduces the overall traffic that's required for sending a file to a browser. Minifaction also changes the names of functions or variables, for example const mySuperVariable can become const d . Every Lightning Web Component JavaScript file that your browser receives from Salesforce in production mode is minified.

Without any special setting, you can use the pretty format option in Chrome DevTools (or similar counterpart in your preferred browser) to get some sort of code formatting. This doesn't give you full readability because of the changed names of variables and functions, but it's pretty decent for a first check. This code is already debuggable, which means you can set breakpoints, inspect values during runtime, and use Chrome DevTools to work with the debugged values.

Proxied values

Next, we proxy certain things, like data that is provisioned via decorators (@api, @track, @wire). Some of that is to be considered read-only, like @api and @wire decorated properties. By using JavaScript Proxies we make sure that they stay read-only. And for @track decorated properties, we use proxies to observe data mutation. This also means that you only see the Proxy object and you either have to use something like JSON.stringify() in Chrome DevTools (note: if you have an Object with circular references, it'll throw when stringified), or you have to inspect the object structure itself.

LWC Debug Tools

- Enable Debug Mode in Salesforce to receive error msg.
- Browser Debug Mode (Inspect – CTRL+Shift+I)

LWC -AURA Coexistence

An interoperability layer enables Lightning web components and Aura components to work together in an app.

Lightning web components and Aura components can work together, with following limitations.

01.Compose Aura Components from Lightning Web Components

You can compose Aura components from Lightning web components, but not the other way around. To communicate down the hierarchy, parents set properties on children.

02.Send Events to an Enclosing Aura Component

To communicate from child to parent, dispatch an event. Lightning web components fire DOM events. An enclosing Aura component can listen for these events, just like an enclosing Lightning web component can. The enclosing Aura component can capture the event and handle it. Optionally, the Aura component can fire an Aura event to communicate with other Aura components or with the app container.

03.Share JavaScript Code Between LWC and Aura

To share JavaScript code between Lightning web components and Aura components, put the code in an ES6 module.

Coexistence Strategy

Coexistence of LWC and Aura are done by two strategies to consider:

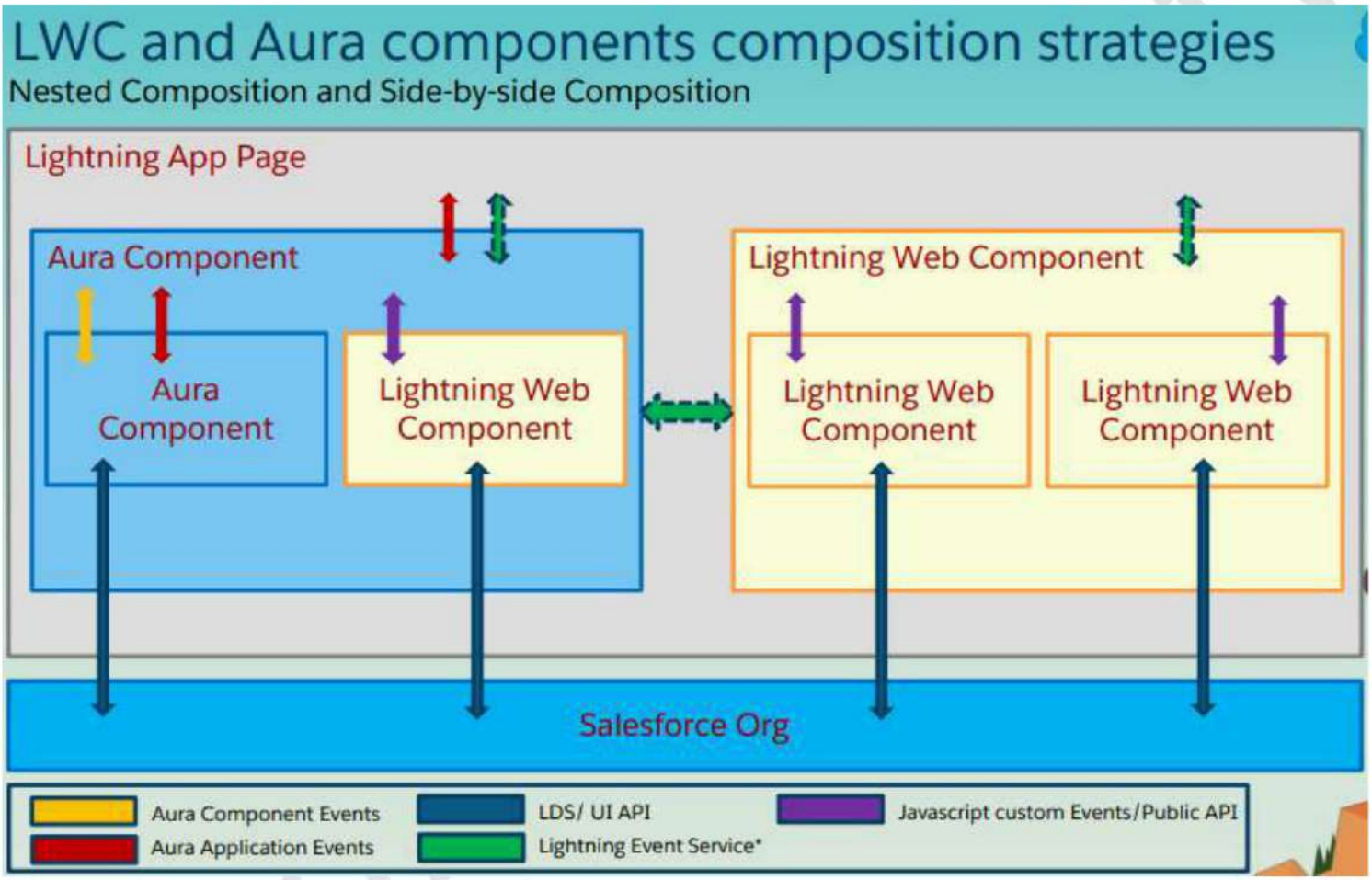
- Nested
- Side-by-side compositions.

Nested Composition:

A nested composition is a component within a component, either an Aura component or Lightning web component. This is like a parent-child relationship. The rule is you can have both Aura components and Lighting web components inside a parent Aura component, but Lightning web components can only contain other Lightning web components.

Side-by-side Composition

The other type of composition is side-by-side in which Aura component and Lightning web component are not within the same hierarchy but are placed separately in a Lightning App Page. Here the parent Aura component and the parent Lightning web component are in Side-by-Side strategy. And within the parent Aura component and the parent Lightning web component, the strategy used is nested composition.



RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

LWC – Testing

Jest is a powerful tool with rich features for writing JavaScript tests. Use Jest to write unit tests for all of your Lightning web components.

Run Jest tests at the command line or (with some configuration) within your IDE. Jest tests don't run in a browser or connect to an org, so they run fast. When run in "watch mode" they give you immediate feedback while you're coding. Jest tests work only with Lightning web components, they don't work with Aura components.

Write Jest tests to:

- Test a component in isolation
- Test a component's public API (@api properties and methods, events)
- Test basic user interaction (clicks)
- Verify the DOM output of a component
- Verify that events fire when expected

LWC Testing Steps:

- **Install sfdx-lwc-jest**
  Install sfdx-lwc-jest and its dependencies into each Salesforce DX project. sfdx-lwc-jest works in Salesforce DX projects only.

- **Run Jest Tests for Lightning Web Components**
  Run your unit tests frequently or continuously during component development.

- **Write Jest Tests for Lightning Web Components**
  Write your component tests in local JavaScript files. Commit them to version control along with the component itself. Jest tests aren't saved to Salesforce.

- **Write Jest Tests for Lightning Web Components That Use the Wire Service**
  Components use the wire service to get data from Salesforce. To test how these components handle data and errors from the wire service, use the @salesforce/sfdx-lwc-jest test utility.

- **DOM Inspection Tests Are Subject to Change**
  The content and structure of HTML, CSS, and the DOM in Lightning Experience can change at any time and can't be considered a stable API. UI tests that use tools like Selenium WebDriver to reach into component internals require your ongoing maintenance.

- **Jest Test Patterns and Mock Dependencies**
  While testing with Jest, follow these patterns and practices to isolate behavior and maximize the efficiency of your unit tests.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Migrating AURA to LWC

Aura components and Lightning web components can exist in the same application.

- ❑ As of Today No Code Migration Tool Available.
- ❑ Third Party non supported tool/interfaces are not recommended and supported.
- ❑ Developers to migrate manually.
- ❑ Developers to decide only to migrate those AURA components which are in LWC, else choose AURA for no supported LWC Components
- ❑ An Aura component can contain a Lightning web component.
- ❑ A Lightning web component **can't** contain an Aura component.

Migrate Aura Components to Lightning Web Components

Migration Strategy

The programming model for Lightning Web Components is fundamentally different than the model for Aura components. Migrating a component is not a line-by-line conversion, and it's a good opportunity to revisit your component's design. Before you migrate an Aura component, evaluate the component's attributes, interfaces, structures, patterns, and data flow.

Pick a Component to Migrate

To decide whether to migrate a component, first understand how Aura components and web components work together, and then evaluate how and where the component is used.

Migrate Component Bundle Files

The component bundle file structure is different for the Aura programming model versus the Lightning Web Components programming model.

RESOURCE	AURA FILE	LIGHTNING WEB COMPONENTS FILE
Markup	`sample.cmp`	`sample.html`
Controller	`sampleController.js`	`sample.js`
Helper	`sampleHelper.js`	`sample.js`
Renderer	`sampleRenderer.js`	`sample.js`
CSS	`sample.css`	`sample.css`
Documentation	`sample.auradoc`	Not currently available
Design	`sample.design`	`sample.js-meta.xml`
SVG	`sample.svg`	Not currently available

Migrate Markup

An Aura .cmp file contains markup, including Aura-specific tags. Let's look at how this markup maps to equivalent concepts in Lightning web components.

- **Migrate Attributes**
  Migrate attributes from `<aura:attribute>` tags in an Aura component to JavaScript properties in a Lightning web component.
- **Migrate Iterations**
  Migrate `<aura:iteration>` tags in an Aura component to `for:each` in a Lightning web component.
- **Migrate Conditionals**
  Migrate `<aura:if>` tags in an Aura component to `if:true` and `if:false` in a Lightning web component.
- **Migrate Expressions**
  Migrate expressions from markup in an Aura component to JavaScript in a Lightning web component.
- **Migrate Global Value Providers**
  Migrate global value providers in an Aura component to scoped module imports in a Lightning web component.
- **Migrate Initializers**
  Replace an `init` event handler in an Aura component with the standard JavaScript `connectedCallback()` method in a Lightning web component.
- **Migrate Facets**
  Migrate a facet in an Aura component to a slot in a Lightning web component.
- **Migrate Base Components**
  Base Lightning components have a different syntax when you use them in the two programming models. Base Lightning components are building blocks that Salesforce provides in the `lightning` namespace.
- **Migrate Registered Events**
  There's no equivalent in Lightning web components for the `<aura:registerEvent>` tag in Aura component markup to register that a component can fire an event.
- **Migrate Event Handlers**
  There's no equivalent in Lightning web components for the `<aura:handler>` tag in Aura component markup that configures an event handler.
- **Migrate Access Controls**
  Migrate access controls for an Aura component to the appropriate value for `<isExposed>` in a Lightning web component's configuration file. Migrate access controls for an attribute in an Aura component to the appropriate combination of a JavaScript property decorator and value for `<isExposed>`.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Migrate Events

Migrate component events in Aura components to standard DOM events in Lightning web components.

Create an event

Instead of the proprietary `Event` object in Aura components, use the `Event` or `CustomEvent` standard DOM objects. There's no equivalent in Lightning web components for the `<aura:registerEvent>` tag in Aura component markup to register that a component can fire an event.

Fire an event

Instead of `event.fire()` in an Aura component, use `this.dispatchEvent(myEvent)`, which is a standard DOM method, in Lightning web components.

Handle an event

An Aura component uses the `<aura:handler>` tag in markup to define a handler. Alternatively a component can declare a handler action when it references another component in its markup. This Aura component uses `c:child` in its markup and declares a `handleChildEvent` handler for the `sampleComponentEvent` that `c:child` fires.

```
<c:child sampleComponentEvent="{!c.handleChildEvent}"/>
```

A Lightning web component can similarly declare a declarative handler. The event name in the declarative handler is prefixed by `on`.

```
<c-child onsampleComponentEvent={handleChildEvent}></c-child>
```

The event handler function, `handleChildEvent`, is defined in the JavaScript file of the component.

In a Lightning web component, you can also programmatically set up a handler using the standard `addEventListener()` method in the component's JavaScript file.

Retrieve an event payload

Use the `event.detail` property to retrieve the event payload unless otherwise stated.

Migrate Interfaces

Implementing an Aura interface enables you to receive context data or to surface your custom component in different contexts, such as in the Lightning App Builder or Experience Builder.

AURA COMPONENT	LIGHTNING WEB COMPONENT	DESCRIPTION
`lightning:hasPageReference`	`import { CurrentPageReference } from 'lightning/navigation';` `@wire(CurrentPageReference) pageRef;`	Indicates that the component accepts a PageReference.

`flexipage:availableForAllPageTypes`	In `*.js-meta.xml`:
	`<targets>`
	`<target>lightning__AppPage</target>`
	`<target>lightning__RecordPage</target>`
	`<target>lightning__HomePage</target>`
	`<target>lightning__UtilityBar</target>`
		Indicates that the component is available for record pages, the utility bar, and any other type of pages. Specify each page type explicitly as a target

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

AURA COMPONENT	LIGHTNING WEB COMPONENT	DESCRIPTION
	</targets>	
flexipage:availableForRecordHome	In *.js-meta.xml: <targets> <target>lightning__RecordPage</target> </targets>	Indicates that the component is available for record pages only.
force:hasRecordId	import { LightningElement, api } from 'lwc'; @api recordId; In *.js-meta.xml: <targets> <target>lightning__RecordPage</target> </targets>	Indicates that the component takes a record Id as an attribute.
force:hasSObjectName	import { LightningElement, api } from 'lwc'; @api objectApiName; In *.js-meta.xml: <targets> <target>lightning__RecordPage</target> </targets>	Indicates that the component takes the API name of the current record's object type.
forceCommunity:availableForAllPageTypes	In *.js-meta.xml: <targets>	Indicates that the component can be used in Experience Builder.

AURA COMPONENT	LIGHTNING WEB COMPONENT	DESCRIPTION
	`<target>lightningCommunity__Page</target>` `</targets>`	

`clients:availableForMailAppPage`	In `*.js-meta.xml`:
	`<targets>`
	`<target>lightning__Inbox</target>`
	`</targets>`
		Indicates that the component can be used in Lightning App Builder to add to email application panes for the Outlook and Gmail integrations.

`clients:availableForMailAppPage`	In `*.js-meta.xml`:
	`<targets>`
	`<target>lightning__Inbox</target>`
	`</targets>`
		Indicates that the component can be used in Lightning App Builder to add to email application panes for the Outlook and Gmail integrations.

Migrate CSS

Lightning web components use standard CSS syntax. Remove the proprietary THIS class that Aura components use.

Lightning web components use standard CSS syntax. Remove the proprietary THIS class that Aura components use.

```
.THIS .red {
    background-color: red;
}
```

To

```
.red {
    background-color: red;
}
```


RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

Migrate JavaScript

Move JavaScript code from your client-side controller, helper, and renderer to the single JavaScript file in a Lightning web component.

Migrate Apex

Aura components and Lightning web components both use an Apex controller to read or persist Salesforce data. There are no syntax differences for the two programming models.

Lightning web components can also use the JavaScript @wire decorator to wrap calls to an Apex method or the User Interface API.

Data Binding Behavior Differences With Aura

When you add a component in markup, you can initialize public property values in the component based on property values of the owner component. In Lightning Web Components, the data binding for property values is one-way. The data-binding behavior is different for the Lightning Web Components and Aura development models.

Base Components: Aura Vs Lightning Web Components

Many of the base Lightning web components have the same functionality and attributes as their Aura counterparts. Some have subtle differences.

Mapping of Aura Components to Lightning Web Components

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
`lightning`	`accordion`	`accordion`	`lightning:accordion` supports the initialization of multiple open sections in simple markup. To initialize `lightning-accordion` with multiple open sections, pass the array of section names using JavaScript.
`lightning`	`accordionSection`	`accordion-section`	`lightning:accordionSection` supports custom menu actions through an `actions` attribute that passes in a `lightning:buttonMenu` component. `lightning-accordion-section` supports custom menu actions through an `actions` slot for `lightning-button-menu`.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
`lightning`	`avatar`	`avatar`	-
`lightning`	`badge`	`badge`	-
`lightning`	`breadcrumb`	`breadcrumb`	-
`lightning`	`breadcrumbs`	`breadcrumbs`	-
`lightning`	`button`	`button`	-
`lightning`	`buttonGroup`	`button-group`	-
`lightning`	`buttonIcon`	`button-icon`	In `lightning-button-icon`, `iconClass` isn't supported.
`lightning`	`buttonIconStateful`	`button-icon-stateful`	-
`lightning`	`buttonMenu`	`button-menu`	-
`lightning`	`buttonStateful`	`button-stateful`	In `lightning:buttonStateful`, use the `state` attribute to show when the button is selected. In `lightning-button-stateful`, use the `selected` attribute.
`lightning`	`card`	`card`	In `lightning:card`, you can set the `title` and `footer` attributes as strings or objects, which enables you to pass components. You can use the `actions` attribute to specify a `lightning:button` or `lightning:buttonIcon` to display in the card's header. In `lightning-card`, you can set the `title` and `footer` attributes to text only. To include markup or other components, use named slots. You can specify actions only in an `action` named slot.
`lightning`	`carousel`	`carousel`	-
`lightning`	`checkboxGroup`	`checkbox-group`	-
`lightning`	`clickToDial`	`click-to-dial`	-
`lightning`	`combobox`	`combobox`	-
`lightning`	`container`	-	Lightning web component not yet available.
`lightning`	`datatable`	`datatable`	`lightning:datatable` doesn't support custom data types in table cells. `lightning-datatable` supports custom data types in table cells.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
`lightning`	`dualListbox`	`dual-listbox`	-
`lightning`	`dynamicIcon`	`dynamic-icon`	-
`lightning`	`fileCard`	-	Lightning web component not yet available.
`lightning`	`fileUpload`	`file-upload`	-
`lightning`	`flexipageRegionInfo`	-	In Aura, use `flexipageRegionInfo` to make the containing component aware of the width it can occupy. In Lightning web components, use the `flexipageRegionWidth` public property for this purpose.
`lightning`	`flow`	-	Lightning web component not yet available.
`lightning`	`formattedAddress`	`formatted-address`	-
`lightning`	`formattedDateTime`	`formatted-date-time`	-
`lightning`	`formattedEmail`	`formatted-email`	-
`lightning`	`formattedLocation`	`formatted-location`	-
`lightning`	`formattedName`	`formatted-name`	-
`lightning`	`formattedNumber`	`formatted-number`	In `lightning:formattedNumber`, use the `style` attribute to specify the type of number to display. In `lightning-formatted-number`, use the `format-style` attribute for this purpose.
`lightning`	`formattedPhone`	`formatted-phone`	-
`lightning`	`formattedRichText`	`formatted-rich-text`	-
`lightning`	`formattedText`	`formatted-text`	-
`lightning`	`formattedTime`	`formatted-time`	-
`lightning`	`formattedUrl`	`formatted-url`	-
`lightning`	`helptext`	`helptext`	-
`lightning`	`icon`	`icon`	-
`lightning`	`input`	`input`	-
`lightning`	`inputAddress`	`input-address`	-

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
`lightning`	`inputField`	`input-field`	`lightning:inputField` doesn't provide a way to indicate that a field has been modified but not saved. `lightning-input-field` supports a `dirty` attribute to indicate that the field has been modified by the user but not saved or submitted. For more information, see the `recordEditForm` differences.
`lightning`	`inputLocation`	`input-location`	-
`lightning`	`inputName`	`input-name`	-
`lightning`	`inputRichText`	`input-rich-text`	`lightning:inputRichText` requires you to use the `lightning:insertImageButton` child component to add an insert image button. It doesn't support an insert link button. `lightning-input-rich-text` provides buttons to insert images and links.
`lightning`	`insertImageButton`	-	An insert image button is supported in `lightning-input-rich-text` so you don't need `lightning:insertImageButton`.
`lightning`	`layout`	`layout`	`lightning:layout` allows HTML tags, text, expressions, and other components between `lightning:layoutItem` components. `lightning-layout` does not allow expressions or other components between `lightning-layout-item` components. However, you can place HTML tags and text between `lightning-layout-item` components.
`lightning`	`layoutItem`	`layout-item`	See the differences for `lightning-layout`.
`lightning`	`listView`	-	Lightning web component not yet available.
`lightning`	`map`	`map`	-
`lightning`	`menuItem`	`menu-item`	-
`lightning`	`navigation`	`navigation` module	In an Aura component, use the `lightning:navigation` component to access the navigation service. In a Lightning web component, import the `lightning/navigation` module in the component's JavaScript.
`lightning`	`notificationsLibrary`	`platformShowToastEvent` module	In Aura components, add the `notificationsLibrary` component to create notices and toasts. The library provides `showToast` and `showNotice` events.

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
			In Lightning web components, import the `lightning/platformShowToastEvent` module in your component's JavaScript. The module provides an event named `ShowToastEvent`. LWC doesn't support notices yet. `ShowToastEvent` supports the same parameters, modes, and variants as Aura's `showToast`.
`lightning`	`outputField`	`output-field`	See `recordViewForm` differences.
`lightning`	`overlayLibrary`		In Aura components, add the `overlayLibrary` component to create modal dialogs and popovers. The library provides `showCustomModal()`, `showCustomPopover()`, and `notifyClose()` methods.
`lightning`	`path`	-	Lightning web component not yet available.
`lightning`	`picklistPath`	-	Lightning web component not yet available.
`lightning`	`pill`	`pill`	`lightning:pill` requires the `media` attribute to include an avatar or icon. The `variant` attribute is not available on `lightning:pill`. `lightning-pill` can accept avatars and icons as nested components.
`lightning`	`pillContainer`	`pill-container`	-
`lightning`	`progressBar`	`progress-bar`	-
`lightning`	`progressIndicator`	`progress-indicator`	-
`lightning`	`progressRing`	`progress-ring`	
`lightning`	`quipCard`	-	Lightning web component not yet available.
`lightning`	`radioGroup`	`radio-group`	-
`lightning`	`recordEditForm`	`record-edit-form`	`lightning:recordEditForm` doesn't require you to nest `lightning:inputField` components directly in it. You can nest `lightning:inputField` components in HTML tags or other components such as `lightning:layout` to tailor the layout. `lightning-input-field` is intended to be used as a child of `lightning-record-edit-form`. You can nest `lightning-input-field` in HTML tags or another base component such as `lightning-layout` within

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
			`lightning-record-edit-form`. However, you can't nest `lightning-input-field` in a custom component.
`lightning`	`recordForm`	`record-form`	-
`lightning`	`recordViewForm`	`record-view-form`	`lightning:recordViewForm` doesn't require you to nest `lightning:outputField` components directly in it, although this is the expected usage. `lightning-output-field` must be a child of `lightning-record-view-form`. Don't nest `lightning-output-field` in another element like `lightning-layout`. You can place it in a container such as `<div>` within `lightning-record-view-form`.
`lightning`	`relativeDateTime`	`relative-date-time`	-
`lightning`	`select`	-	Lightning web component not yet available. In a mobile environment, use the native `<select>` element. In a desktop environment, consider using the `lightning-combobox` Lightning web component. To support multiple selection, use `lightning-dual-listbox`.
`lightning`	`slider`	`slider`	-
`lightning`	`spinner`	`spinner`	-
`lightning`	`tab`	`tab`	In `lightning:tab`, use the `id` attribute to assign a string to identify a tab. In `lightning-tab`, use the `value` attribute to assign the tab identifier. The `tabset` uses these strings to determine which tab was clicked.
`lightning`	`tabset`	`tabset`	`lightning:tabset` uses the `selectedTabId` attribute to specify the open tab. `lightning-tabset` uses the `active-tab-value` attribute.
`lightning`	`textarea`	`textarea`	-
`lightning`	`tile`	`tile`	In `lightning:tile`, you can use `<aura:set>` to add a `media` attribute that contains an avatar component or icon component.

RUPOM CHAKRABORTY - Capital Info Solutions
Shyamala Plaza, Behind Mythrivanam, Amerpet, Hyderabad, Telangana State, INDIA. Ph: 8686864286

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
			In `lightning-tile`, specify the avatar and icon components with `media` named slots.
`lightning`	`tree`	`tree`	-
`lightning`	`treeGrid`	`tree-grid`	-
`lightning`	`unsavedChanges`	-	Lightning web component not yet available.
`lightning`	`verticalNavigation`	`vertical-navigation`	-
`lightning`	`verticalNavigationItem`	`vertical-navigation-item`	-
`lightning`	`verticalNavigationItemBadge`	`vertical-navigation-item-badge`	-
`lightning`	`verticalNavigationItemIcon`	`vertical-navigation-item-icon`	-
`lightning`	`verticalNavigationItemOverflow`	`vertical-navigation-item-overflow`	-
`lightning`	`verticalNavigationSection`	`vertical-navigation-section`	-
`aura`	`expression`	Custom JavaScript	
`aura`	`html`	-	N/A
`aura`	`if`	`if:true / if:false`	
`aura`	`iteration`	`for:each`	
`aura`	`renderIf`	`if:true / if:false`	
`aura`	`template`	-	N/A
`aura`	`text`	-	N/A
`aura`	`token`	-	N/A
`aura`	`unescapedHTML`	-	N/A
`force`	`canvasApp`	-	N/A
`force`	`inputField`	`input-field`	`force:inputField` has been replaced by `lightning:inputField` in Aura. To create editable fields based on field types, use `lightning:inputField`

NAMESPACE	AURA COMPONENT	LIGHTNING WEB COMPONENT	DIFFERENCES
			with `lightning:recordEditForm` or their Lightning web component counterparts.
`force`	`outputField`	`output-field`	`force:outputField` has been replaced by `lightning:outputField` in Aura. To create read-only fields based on field types, use `lightning:outputField` with `lightning:recordViewForm` or their Lightning web component counterparts.
`force`	`recordData`	`@wire`	
`force`	`recordEdit`	`record-edit-form`	`force:recordEdit` has been replaced by `lightning:recordEditForm`.
`force`	`recordView`	`record-view-form`	`force:recordView` has been replaced by `lightning:recordViewForm`.
`ltng`	`require`	`platformResourceLoader` module	In Aura components, add the `<ltng:require>` tag to your component markup to reference external static resources. In Lightning web components, import the `lightning/platformResourceLoader` module in your component's JavaScript.