Lightning Web Components Cheatsheet

Complete Reference for LWC Development in Salesforce

Component Structure

```
myComponent/
├── myComponent.html // Template
├── myComponent.js // JavaScript Controller
├── myComponent.js-meta.xml // Configuration
└── myComponent.css // Styles (optional)
```

Basic Component Example

```
<!-- myComponent.html -->
<template>
  <lightning-card title="Hello LWC">
    <div class="slds-m-around_medium">
      <p>{greeting}</p>
    </div>
  </lightning-card>
</template>

// myComponent.js
import { LightningElement } from 'lwc';

export default class MyComponent extends LightningElement {
  greeting = 'Hello, Salesforce!';
}
```

Component Decorators

Decorator	Purpose	Example
@api	Public property/method	@api recordId;
@track	Track object/array changes (legacy)	@track myArray = [];
@wire	Wire service to component	@wire(getRecord, {...})

Note: @track is no longer needed for primitive values. LWC automatically tracks them. Use @track only for objects and arrays when needed.

Data Binding in Template

Property Binding

```
<!-- One-way binding -->
<p>{propertyName}</p>

<!-- Attribute binding -->
```

```
<img src={imageUrl} alt="pic">
```

Event Binding

```
<!-- onclick handler -->
<button onclick={handleClick}>
Click Me
</button>

// In JS
handleClick(event) {
console.log('Clicked');
}
```

Conditional Rendering

```
<!-- if:true -->
<template if:true={isVisible}>
  <p>This is visible</p>
</template>

<!-- if:false -->
<template if:false={isVisible}>
  <p>This is hidden</p>
</template>

<!-- lwc:if, lwc:elseif, lwc:else (Modern syntax) -->
<template lwc:if={condition1}>
  <p>Condition 1</p>
</template>
<template lwc:elseif={condition2}>
  <p>Condition 2</p>
</template>
<template lwc:else>
  <p>Else</p>
</template>
```

Iterating Lists

```
<!-- for:each -->
<template for:each={contacts} for:item="contact" for:index="index">
  <div key={contact.Id}>
    {contact.Name}
  </div>
</template>

<!-- iterator -->
<template iterator:it={contacts}>
  <div key={it.value.Id}>
    {it.value.Name}
    <template if:true={it.first}>First</template>
    <template if:true={it.last}>Last</template>
  </div>
</template>
```

Lifecycle Hooks

Hook	When Called	Use Case
constructor()	Component created	Initialize properties
connectedCallback()	Inserted into DOM	Fetch data, setup
renderedCallback()	After render	DOM manipulation
disconnectedCallback()	Removed from DOM	Cleanup
errorCallback()	Error in lifecycle	Error handling

@wire Adapter

```
// Wire to property
@wire(getRecord, { recordId: '$recordId', fields: FIELDS })
record;

// Wire to function
@wire(getRecord, { recordId: '$recordId', fields: FIELDS })
wiredRecord({ error, data }) {
  if (data) {
    this.record = data;
  } else if (error) {
    console.error(error);
  }
}
```

Apex Wire Adapter

```
// Import Apex method
import getAccountList from '@salesforce/apex/AccountController.getAccountList';

// Wire to Apex
@wire(getAccountList, { searchKey: '$searchKey' })
accounts;

// Apex Class
public with sharing class AccountController {
  @AuraEnabled(cacheable=true)
  public static List<Account> getAccountList(String searchKey) {
    return [SELECT Id, Name FROM Account WHERE Name LIKE :searchKey LIMIT 10];
  }
}
```

Imperative Apex Call

```
import createAccount from '@salesforce/apex/AccountController.createAccount';

handleSave() {
  createAccount({ name: this.accountName })
    .then(result => {
      console.log('Success:', result);
    })
    .catch(error => {
```

```
      console.error('Error:', error);
    });
}
```

Component Communication

Parent to Child

```
<!-- Parent -->
<c-child message={msg}></c-child>

// Child
@api message;
```

Child to Parent

```
// Child
const event = new CustomEvent('select', {
detail: data
});
this.dispatchEvent(event);

<!-- Parent -->
<c-child onselect={handleSelect}></c-child>
```

Lightning Data Service (LDS)

```
// Import LDS modules
import { getRecord, updateRecord, createRecord, deleteRecord } from 'lightning/uiRecordApi';

// Get Record
@wire(getRecord, { recordId: '$recordId', fields: ['Account.Name'] })
account;

// Update Record
const fields = { Id: recordId, Name: 'New Name' };
updateRecord({ fields })
.then(() => console.log('Updated'))
.catch(error => console.error(error));
```

Common Base Components

Component	Purpose
lightning-card	Container with header and footer
lightning-button	Styled button
lightning-input	Input field
lightning-combobox	Dropdown select
lightning-datatable	Data table with sorting
lightning-record-form	Auto-form for record

lightning-record-edit-form	Custom edit form
lightning-spinner	Loading spinner

Navigation (lightning/navigation)

```
import { NavigationMixin } from 'lightning/navigation';

export default class MyComponent extends NavigationMixin(LightningElement) {
  navigateToRecord() {
    this[NavigationMixin.Navigate]({
      type: 'standard__recordPage',
      attributes: {
        recordId: this.recordId,
        actionName: 'view'
      }
    });
  }
}
```

Show Toast Notification

```
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

showToast() {
  const event = new ShowToastEvent({
    title: 'Success',
    message: 'Record saved successfully',
    variant: 'success' // success, error, warning, info
  });
  this.dispatchEvent(event);
}
```

Meta Configuration (js-meta.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>60.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
  <targetConfigs>
    <targetConfig targets="lightning__RecordPage">
      <objects>
        <object>Account</object>
      </objects>
    </targetConfig>
  </targetConfigs>
</LightningComponentBundle>
```