

Parent to Child Component Communication in LWC

Lightning Web Components (LWC) is a modern, standards-based framework for building web applications on the Salesforce platform. LWC promotes the development of modular, reusable components that can be easily integrated into your applications. One of the key aspects of component-based development is communication between components.

Component Composition:

Component composition in Lightning Web Components (LWC) is a hierarchical conception that involves creating a scale of components where one component can include other components. This allows you to build complex and reusable user interfaces by breaking them down into smaller, more manageable parts.

Parent Component: This is the higher-level component that includes and manages one or more child components. The parent component can pass data to its child components, listen for events from them, and control their behaviour.

Child Components: These are the lower, more-contained components that can be included within the parent component. Child components accept data from the parent component, handle their own logic, and send events that the parent component can attend to.

- **Including Child Components in a Parent Component:**

Syntax to add child into parent component :

Parent.html

```
<template>
  <c-child-component></c-child-component>
</template>
```

- **Why Component Communication is Essential**

In a complex web application, it's common to break the user interface into smaller, manageable components. These components need to work together to provide a seamless user experience.

Parent to Child component communication is particularly important when you want to pass data, trigger actions, or share information between a parent component and its child components.

- **Passing Data from Parent to Child**

There are several methods for passing data from a parent component to a child component in LWC. The most commonly used method is by using attributes. In order to implement this communication, properties in the child component need to be accessible at parent component. To expose properties at child component publicly, **@api** decorator is used. Using API decorator primitive as well as non-primitive data at child component gets accessible to parent component.

USE CASE I: Parent to child communication utilizing primitive data:
sampleParentComponent.html

```
<template>
  <lightning-card title="Parent Component">
    <lightning-input class="slds-m-around_small"
      type="text"
      label="Name"
      placeholder="Enter string here.."
      onchange={handleInput}>
    </lightning-input>
    <c-sample-child-component str={inputString}></c-sample-child-component>
  </lightning-card>
</template>
```

sampleParentComponent.js

```
import { LightningElement } from 'lwc';
import getResponse from
  '@salesforce/apex/ComponentCommunicationHandler.getResponse';
export default class SampleParentComponent extends LightningElement {
  inputstring;
  handleInput(event){
    //console.log('OUTPUT: ',event.target.value);
  }
  this.inputstring = event.target.value;
}
```

sampleChildComponent.html

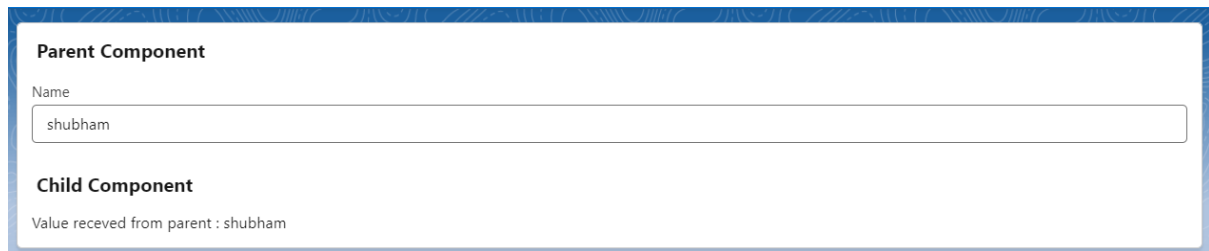
```
<template>
<lightning-card title="Child Component" >
  <p class="slds-m-around_small">Value received from parent : {str}</p>
</lightning-card>
</template>
```

sampleChildComponent.js

```
import { LightningElement, api } from 'lwc';
export default class SampleChildComponent extends LightningElement{
  @api str;
}
```

in this example, we are taking input using lightning input in parent component and passing that input to child component. As you can see, @api decorator is used to make the variable in child component accessible to parent component.

OUTPUT:



The screenshot shows a web interface with two components. The top component, titled 'Parent Component', contains a text input field labeled 'Name' with the value 'shubham'. Below it, the 'Child Component' displays the text 'Value received from parent : shubham', indicating that the input from the parent is being passed to the child.

USE CASE II: Parent to child communication using non-primitive data:
sampleParentComponent.html

```
<template>
  <lightning-card title="Parent Component">
    <c-sample-child-component color-data-details={dataDetails}>
    </c-sample-child-component>
  </lightning-card>
</template>
```

sampleParentComponent.js

```
import { LightningElement } from 'lwc';
export default class SampleParentComponent extends LightningElement {
  dataDetails =[
    {
      "id": 1,
      "name": "cerulean"
    },
    {
      "id": 2,
      "name": "fuchsia rose"
    },
    {
      "id": 3,
      "name": "true red"
    },
    {
      "id": 4,
      "name": "aqua sky"
    },
  ],
```

```

        {
            "id": 5,
            "name": "aqua"
        },
        {
            "id": 6,
            "name": "blue turquoise"
        }
    ]
}

```

sampleChildComponent.js

```

import { LightningElement, api } from 'lwc';
export default class SampleChildComponentTwo extends LightningElement {
    @api colorDataDetails;
}

```

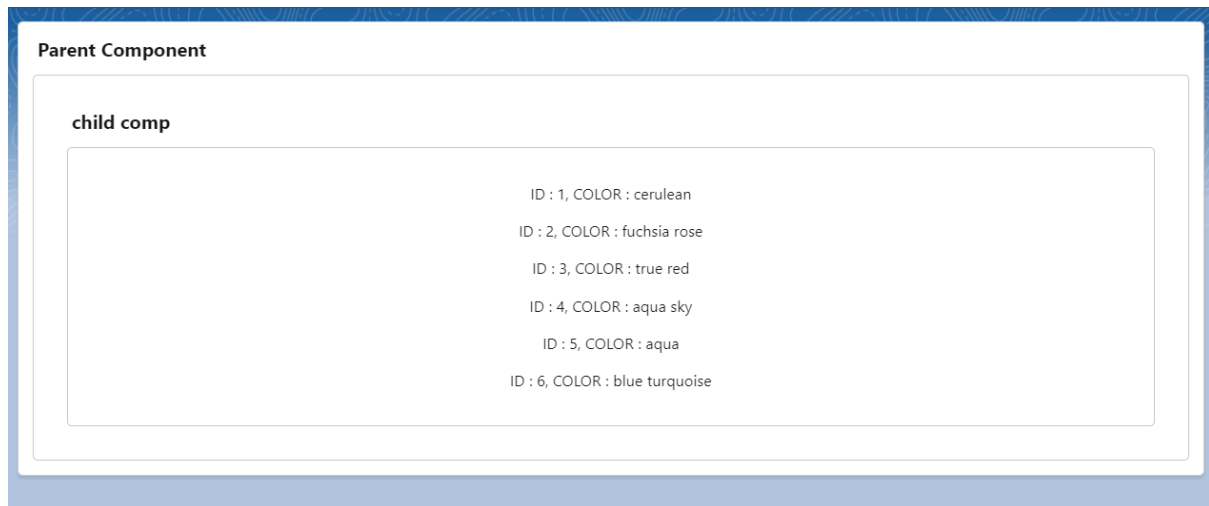
sampleChildComponent.html

```

<template>
    <lightning-card title="child comp 2">
        <p class="slds-text-align_center slds-box
slds-m-around_small">
            Child component - 2:
        </p>
        <template for:each = {colorDataDetails} for:item = 'item'>
            <div class="slds-text-align_center
slds-m-around_small"key = {item.id}>
                ID: {item.id}, COLOR : {item.name}
            </div>
        </template>
    </lightning-card>
</template>

```

OUTPUT :



In this example , we have passed a list From parent component to child component . As you can see the property in child component is decorated using `@api`. While passing values from parent to child , parameter needs to be mentioned in kebab case.

USE CASE III: Parent to child communication using child component methods.

This can be achieved using `template.Queryselector('child-component').methodName();`

In this case, the method which is being called in parent component needs to be decorated using `@api`.

sampleParentComponent.html

```
<template>
  <lightning-card title="PARENT COMPONENT">
    <div class="slds-box slds-m-around_small">
      <lightning-button
        class="slds-m-around_small"
        variant="neutral"
        label="Change comp 4 background to aqua"
        onclick={handleBackgroundChangeClick}>
      </lightning-button>
      <c-sample-child-component-four>
      </c-sample-child-component-four>
    </div>
  </lightning-card>
</template>
```

sampleParentComponent.js

```
import { LightningElement } from 'lwc';
export default class SampleParentComponent extends LightningElement {
    backgroundColor = 'background-blue';
    handleBackgroundChangeClick(){
    this.template.querySelector('c-sample-child-component-four').changeBackground
    (this.backgroundColor);
    }
}
```

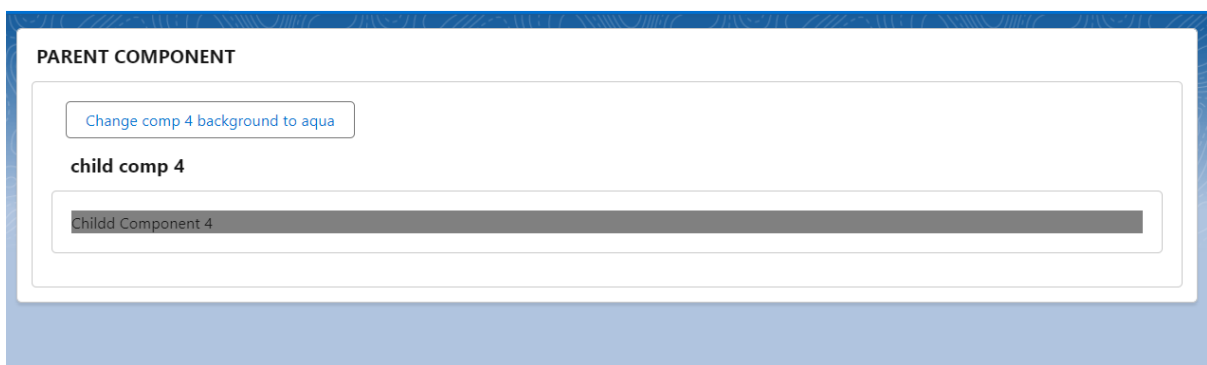
sampleChildComponentFour.js

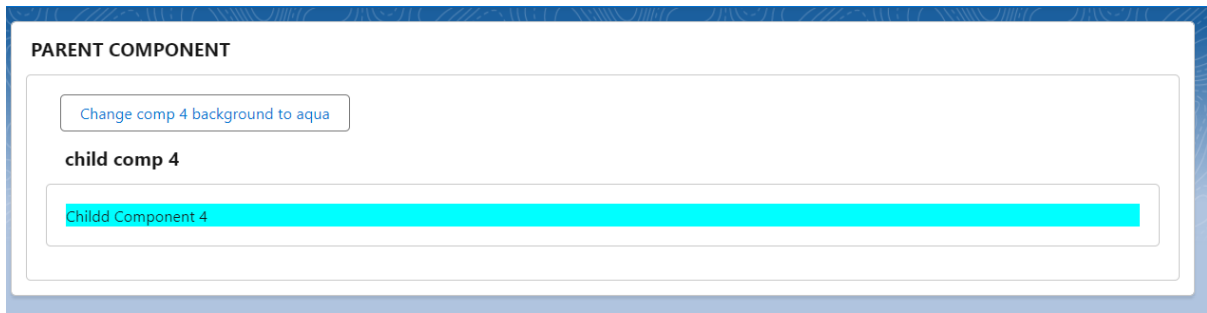
```
import { LightningElement, api } from 'lwc';
export default class SampleChildComponentFour extends LightningElement {
    className = 'background-grey';
    @api changeBackground(bgColor){
        this.className = bgColor;
    }
}
```

sampleChildComponentFour.css

```
.background-grey{
    background-color: grey;
}
.background-blue{
    background-color: aqua;
}
```

OUTPUT:





- **Conclusion**

Effective component communication is essential for building modular and maintainable applications in LWC. By using attributes, properties, custom events, and shared JavaScript modules, you can establish robust Parent to Child communication. Understanding these techniques will empower you to create more interactive and cohesive Lightning Web Components for your Salesforce applications.