

## 1. Parent → Child Communication (Parameters)

### Purpose

Used when a **parent component passes data to a child component**.

### Mechanism

- [Parameter]
- [Parameter] with EventCallback (for interaction)

## 2. Child → Parent Communication (EventCallback)

### Purpose

Used when a **child component needs to notify the parent** (e.g., button click, form submit).

### Mechanism

- EventCallback
- EventCallback<T>

### Example

ParentComponent.razor

```
<ChildComponent OnSave="HandleSave" />

<p>Status: @status</p>

@code {
    string status;

    void HandleSave(string message)
    {
        status = message;
    }
}
```

### ChildComponent.razor

razor

```
<button @onclick="Save">Save</button>
```

```
@code {
```

```
    [Parameter] public EventCallback<string> OnSave { get; set; }
```

```
    async Task Save()
```

```
    {
```

```
        await OnSave.InvokeAsync("Saved Successfully");
```

```
    }
```

```
}
```

### Key Points

- Asynchronous by design
- Keeps components loosely coupled

### 3. Parent ↔ Child (Two-Way Binding)

#### Purpose

Used when **both parent and child need to stay in sync.**

#### Mechanism

- @bind-Value
- Value + ValueChanged

## Example

### ParentComponent.razor

```
razor

<ChildComponent @bind-Value="name" />
<p>Name: @name</p>

@code {
    string name = "San";
}
```

### ChildComponent.razor

```
razor

<input value="@Value"
        @oninput="e => ValueChanged.InvokeAsync(e.Value.ToString())" />

@code {
    [Parameter] public string Value { get; set; }
    [Parameter] public EventCallback<string> ValueChanged { get; set; }
}
```



## Key Points

- Convention-based (Value / ValueChanged)
- Common in form controls

## 4. Cascading Parameters (Implicit Parent → Many Children)

### Purpose

Used to **share common data across many nested components** (e.g., user info, theme, culture).

### Mechanism

- <CascadingValue>
- [CascadingParameter]

## Example

#### App.razor or ParentComponent.razor

```
razor

<CascadingValue Value="currentUser">
    <ChildComponent />
</CascadingValue>

@code {
    string currentUser = "Admin";
}
```

#### ChildComponent.razor

```
razor

<p>User: @User</p>

@code {
    [CascadingParameter] public string User { get; set; }
}
```

### Key Points

- No explicit parameter passing
- Ideal for global data
- Avoid overuse (can reduce clarity)

## 5. Sibling Components Communication (State Container / Service)

### Purpose

Used when **components do not have a direct parent-child relationship**.

### Mechanism

- Shared service (State Container)
- Dependency Injection
- Events / NotifyStateChanged

### Example

#### StateService.cs

```

public class AppState
{
    public string Message { get; private set; }

    public event Action OnChange;

    public void SetMessage(string message)
    {
        Message = message;
        OnChange?.Invoke();
    }
}

```

#### Program.cs

```

csharp

builder.Services.AddScoped<AppState>();

```

#### ReceiverComponent.razor

```

razor

<p>@State.Message</p>

@Inject AppState State

@code {
    protected override void OnInitialized()
    {
        State.OnChange += StateHasChanged;
    }
}

```

#### Key Points

- 🔗 Best for large applications
- 🔗 Enables decoupled communication

### 6. Component Reference (@ref) – Direct Method Calls

#### Purpose

Used when a parent needs to call a child's public method.

#### Example

## ParentComponent.razor

```
ParentComponent.razor

razor

<ChildComponent @ref="child" />
<button @onclick="CallChild">Call Child</button>

@code {
    ChildComponent child;

    void CallChild()
    {
        child.ShowMessage();
    }
}
```

## ChildComponent.razor

```
ChildComponent.razor

razor

@code {
    public void ShowMessage()
    {
        Console.WriteLine("Called from Parent");
    }
}
```

## Key Points

- Tight coupling
- Avoid for state management
- Useful for UI control (dialogs, focus)

## 7. Choosing the Right Approach

Scenario	Recommended Pattern
Parent → Child	[Parameter]
Child → Parent	EventCallback
Two-way binding	@bind

Deep hierarchy      Cascading Parameters

Siblings / Unrelated      State Container

Direct control @ref