Lightning Message Channel

1)Create a message channel folder in force.app/main/default

2)Create file in message channel folder

3)Give message channel file name like

Sample_MessageChannel.messageChannel-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningMessageChannel xmlns="https://soap.sforce.com/2006/04/metadata">
    <description>This is a sample Lightning Message Channel.</description>
    <isExposed>true</isExposed>
    <lightningMessageFields>
        <description>Variable 1</description>
        <fieldName>variable1</fieldName>
    </lightningMessageFields>
    <masterLabel>Sample Message Channel</masterLabel>
</LightningMessageChannel>
```

Pubsub Model example

1)Create one Lwc component with the name pubsub

2) copy all content in this Pubsub.js

```
// A basic pub-sub mechanism for sibling component communication



const events = {};
```

```
const samePageRef = (pageRef1, pageRef2) => {

    const obj1 = pageRef1.attributes;

    const obj2 = pageRef2.attributes;

    return Object.keys(obj1)

        .concat(Object.keys(obj2))

        .every(key => {

            return obj1[key] === obj2[key];

        });

};

/**

 * Registers a callback for an event

 * @param {string} eventName - Name of the event to listen for.

 * @param {function} callback - Function to invoke when said event is
fired.

 * @param {object} thisArg - The value to be passed as the this parameter
to the callback function is bound.
```

```
 */

const registerListener = (eventName, callback, thisArg) => {

    // Checking that the listener has a pageRef property. We rely on that
property for filtering purpose in fireEvent()

    if (!thisArg.pageRef) {

        throw new Error(

            'pubsub listeners need a "@wire(CurrentPageReference) pageRef"
property'

        );

    }

    if (!events[eventName]) {

        events[eventName] = [];

    }

    const duplicate = events[eventName].find(listener => {

        return listener.callback === callback && listener.thisArg ===
thisArg;

    });
```

```
    if (!duplicate) {
        events[eventName].push({ callback, thisArg });
    }
};

/**
 * Unregisters a callback for an event
 * @param {string} eventName - Name of the event to unregister from.
 * @param {function} callback - Function to unregister.
 * @param {object} thisArg - The value to be passed as the this parameter
to the callback function is bound.
 */
const unregisterListener = (eventName, callback, thisArg) => {
    if (events[eventName]) {
        events[eventName] = events[eventName].filter(
            listener =>
                listener.callback !== callback || listener.thisArg !==
thisArg
```

```
        );

    }

};

/**

 * Unregisters all event listeners bound to an object.

 * @param {object} thisArg - All the callbacks bound to this object will
be removed.

 */

const unregisterAllListeners = thisArg => {

    Object.keys(events).forEach(eventName => {

        events[eventName] = events[eventName].filter(

            listener => listener.thisArg !== thisArg

        );

    });

};
```

```
/**

 * Fires an event to listeners.

 * @param {object} pageRef - Reference of the page that represents the
event scope.

 * @param {string} eventName - Name of the event to fire.

 * @param {*} payload - Payload of the event to fire.

 */

const fireEvent = (pageRef, eventName, payload) => {

    if (events[eventName]) {

        const listeners = events[eventName];

        listeners.forEach(listener => {

            if (samePageRef(pageRef, listener.thisArg.pageRef)) {

                try {

                    listener.callback.call(listener.thisArg, payload);

                } catch (error) {

                    // fail silently
```

```
                }

            }

        });

    }

};

export {

    registerListener,

    unregisterListener,

    unregisterAllListeners,

    fireEvent

};
```

.....

Next step once u created this file pub sub include Two components program and use this file

Pubcom

```
<template>

    <lightning-card title="Publish Component">
```

```
        <lightning-input class="slds-p-around_medium" label="Enter Text to
Send: " value={strText}
            onchange={changeName}></lightning-input>
        <br/>
        <lightning-button class="slds-p-around_medium" label="Publish"
variant="brand"
            onclick={publishEvent}></lightning-button>

    </lightning-card>
</template>
import { LightningElement, wire } from 'lwc';
import { fireEvent } from 'c/pubsub';
import { CurrentPageReference } from 'lightning/navigation';

export default class PublishCmp extends LightningElement {
    strText = '';
    @wire(CurrentPageReference) objpageReference;

    changeName(event){
        this.strText = event.target.value;
    }

    // This method will fire the event and pass strText as a payload.
    publishEvent(){
        fireEvent(this.objpageReference, 'sendNameEvent', this.strText);
    }
}
```

Subcomp

```
<template>
```

```
    <lightning-card  title="Subscribe Component">
        <p class="slds-p-around_medium" style="font-size:25px">
            Received Text: {strCapturedText}
        </p>
    </lightning-card>

</template>
import { LightningElement, wire } from 'lwc';
import { CurrentPageReference } from 'lightning/navigation';
import { registerListener, unregisterAllListeners } from 'c/pubsub';

export default class SubscribeCmp extends LightningElement {
    strCapturedText = '';
    @wire(CurrentPageReference) pageRef;

    // This method will run once the component is rendered on DOM and will
add the listener.
    connectedCallback(){
        registerListener('sendNameEvent', this.setCaptureText, this);
    }

    // This method will run once the component is removed from DOM.
    disconnectedCallback(){
        unregisterAllListeners(this);
    }

    // This method will update the value once event is captured.
    setCaptureText(objPayload){
        this.strCapturedText = objPayload;
    }
```

```
}
```