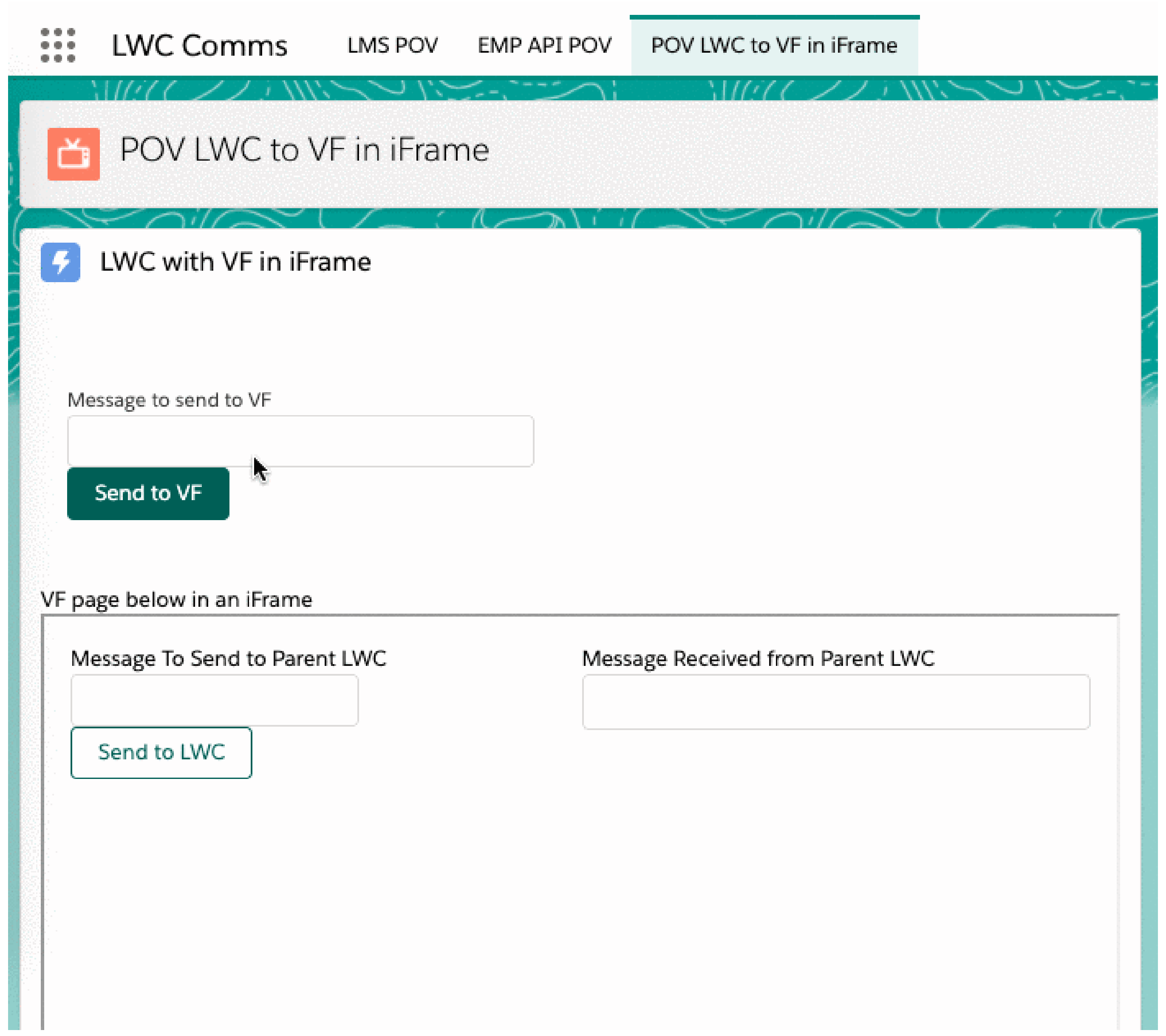
LWC with VF in Iframe – Bidirectional Communication

10TH AUG 2020 WSABEEL 1 COMMENT

This post will demonstrate the case where we have a Visualforce Page nested into a Lightning Web Component and how the communication between the two is established.



There are two important things you need to know about Visualforce pages running in Lightning Experience:

- **Different DOMs**. A Visualforce page hosted in Lightning Experience is loaded in an iframe. In other words, it's loaded in its own window object which is different from the main window object where the Lightning Components are loaded.
- **Different Origins.** Visualforce pages and Lightning Components are served from different domains.

In case of Developer Edition, Sandboxes and Production,
Lightning Components are loaded from a domain that looks like this:
yourdomain.lightning.force.com

Visualforce pages are loaded from a domain that looks like this:
yourdomain–c.visualforce.com
(in Trailhead Playground Org, I witnessed the domain is in the form of –
yourdomain-dev-ed–c.na35.visual.force.com)

The browser's same-origin policy prevents a page from accessing content or code in another page loaded from a different origin (protocol + port + host).

In our case, that means that a Visualforce page can't use the parent window reference to access content or execute code in the Lightning Component wrapper. Similarly, the Lightning component can't use the iframe's *contentWindow* reference to access content or execute code in the Visualforce page it wraps.

These restrictions are enforced for very good reasons. But fortunately, there is also an API (otherWindow.postMessage()) that provides a secure approach (when used properly) to exchange messages between different window objects with content loaded from different origins.

window.postMessage() is a standard.

In the scenario below, we will look at different examples illustrating how **postMessage**() can be used to communicate between LWC and Visualforce page

Let's Jump Into Code

The Installation URL or the Deployment process of the complete App **LWC Comms** is available in our Github Repo – https://github.com/sfwiseguys/LWCComms

LWC to VF –

We have an LWC that wraps a VF page using the iframe tag, and we want the LWC to send messages to the wrapped VF page.

The first argument of **postMessage**() is the data you want to pass to the other window. It can be a primitive data type or an object.

The second argument of postMessage() is the **origin** (protocol + port + host) of the window you send the message to (vfWindow in this case). The event will not be sent if the page in vfWindow at the time postMessage() is called wasn't loaded from vfOrigin.

LWC HTML :

```html
<template>
    <lightning-card title="LWC with VF in iFrame" icon-name="custom:custom9">
        <lightning-layout multiple-rows>
            <lightning-layout-item size="12" padding="around-small">
            </lightning-layout-item>
            <lightning-layout-item size="6" padding="around-small">
                <div class="slds-m-around_medium">
                    <lightning-input label="Message to send to VF" type="text" value={msg} onchange={handleChange}>
                    </lightning-input>
                    <lightning-button label="Send to VF" variant="brand" onclick={handleFiretoVF}></lightning-button>
                </div>

            </lightning-layout-item>
            <lightning-layout-item size="6" padding="around-small">
                <template if:true={receivedMessage}>
                    <p> Message Received from VF = </p>
                    <div class="slds-box">
                        <lightning-formatted-text value={receivedMessage}></lightning-formatted-text>
                    </div>
                </template>
            </lightning-layout-item>

            <lightning-layout-item size="12" padding="around-small">
                <p>VF page below in an iFrame</p>
                <iframe height="400px" width="100%" src="/apex/POV_VFiframe"></iframe>
            </lightning-layout-item>

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

LWC JS :

```javascript
import { LightningElement,wire} from 'lwc';

import getVFOrigin from '@salesforce/apex/POV_Controller.getVFOrigin';

export default class pov_lwc_vfiframe extends LightningElement {
    msg = '';
    receivedMessage = '';
    error;

    // Wire getVFOrigin Apex method to a Property
    @wire(getVFOrigin)
    vfOrigin;

    /*****Called on LOAD of LWC  *****/
    connectedCallback() {
        // Binding EventListener here when Data received from VF
        window.addEventListener("message", this.handleVFResponse.bind(this));
    }

    handleVFResponse(message) {
        if (message.origin === this.vfOrigin.data) {
            this.receivedMessage = message.data;
        }
    }

    handleChange(event) {
        this.msg = event.detail.value;
    }

    handleFiretoVF() {
        let message = this.msg;
        //Firing an event to send data to VF
        this.template.querySelector("iframe").contentWindow.postMessage(message, this.vfOrigin.data);
    }
```

In here, We use an Apex method to get us the Dynamic Origin URL (vfOrigin) as shown below –

```
@AuraEnabled(cacheable=true)
    public static string getVFOrigin() {
      string vfOrigin = '';
    string baseURL = URL.getOrgDomainUrl().toExternalForm(); // Expected Format = https://domain.my.salesforce.com

    // Expected Format for DE, Sandbox & Production ORgs = https://domain--c.visualforce.com
    vfOrigin = baseURL.split('.my.')[0] + '--c.' + 'visualforce.com';

    /* ********* Below Odd Discrepancy was found while implementing this in a Trailhead Playground ***********
    Organization oOrg = [SELECT InstanceName, IsSandbox, OrganizationType FROM Organization LIMIT 1];
    if(oOrg.OrganizationType == 'Developer Edition'){
      // Expected Format for Trailhead Playground DE Org = https://domain--c.ap4.visual.force.com
      vfOrigin = baseURL.split('.my.')[0]+'--c.'+oOrg.InstanceName.toLowercase()+'.visual.force.com';

    } else {
      // Expected Format for personal DE, Sandbox & Production Orgs = https://domain--c.visualforce.com
      vfOrigin = baseURL.split('.my.')[0]+'--c.'+'visualforce.com';
    }  */

    return vfOrigin;
    }
```

VF to LWC-

VF Page –

```
<apex:page lightningStylesheets="true" controller="POV_Controller">
    <apex:slds />
    <div class="slds-grid slds-gutters slds-p-around_medium">
        <div class="slds-col">
            <p>Message To Send to Parent LWC</p>
            <input type="text" id="vfMessage" />
            <br/>
            <button class="slds-button slds-button_outline-brand" onclick="firetoLWC()">Send to LWC</button>
            <br/>
        </div>
        <div class="slds-col">
            <p>Message Received from Parent LWC</p>
            <div id="output" class="slds-box" />
        </div>
    </div>

    <script>
         // Obtaining LEX origin URL from Apex to fire to parent & match the source upon receiving message
         var lexOrigin = '{!lexOrigin}';

        /*** EventListener to GET response from LWC  ***/
        window.addEventListener("message", function (event) {
            if (event.origin === lexOrigin) {
                var receivedfromLWC = event.data;
                var output = document.querySelector("#output");
                output.innerHTML = receivedfromLWC;
            }
        });

        /*** Method to Fire Event to LWC ***/
        function firetoLWC() {
            var message = document.getElementById('vfMessage').value;
            window.parent.postMessage(message, lexOrigin);
        }
    </script>
</apex:page>
```

This also uses an Apex class getter method to dynamically get the LWC origin (**lexOrigin**) as shown below –

```
public string lexOrigin {get{
return URL.getOrgDomainUrl().toExternalForm().split('.my.')[0]+'.lightning.force.com';
} set;}
  // Expected Format = https://domain.lightning.salesforce.com

```

event.origin is the actual origin of the window that sent the message at the time **postMessage**() was called. You should always verify that the actual origin and the expected origin match, and reject the message if they don't.

event.data is the message sent from the other window

When you send a message from a Lightning component to the iframe it wraps using **contentWindow.postMessage()**, there can only be one Visualforce page loaded in that **contentWindow**. In other words, that Visualforce page is the only place where you can set up a message event listener and get the messages sent by the Lightning component in that fashion. This is a **one-to-one** messaging scheme.

When you send a message from an iframed Visualforce page to its Lightning component wrapper using **parent.postMessage()**, **parent** is a reference to your main **window** in Lightning Experience where other Lightning components may be loaded. If other Lightning components loaded in the same window object set up a message event listener, they will receive the Visualforce messages as well. This is a **one-to-many** messaging scheme, and it's something to account for both when you send and receive messages. For example, you could name messages to allow Lightning components to filter incoming messages and only handle messages they are interested in.

Important Security Consideration – window.postMessage() is a standard web API that is not aware of the Lightning and Locker service namespace isolation level. As a result, there is no way to send a message to a specific namespace or to check which namespace a message is coming from. Therefore, messages sent using postMessage() should be limited to non sensitive data and should not include sensitive data such as user data or cryptographic secrets.

Using this secure and standard-based approach, you can tightly integrate Visualforce pages in Lightning Experience and support all your communication requirements: Lightning to Visualforce, Visualforce to Lightning, and even Visualforce to Visualforce inside Lightning Experience.

An Important highlight to share here is the Dynamic generation of Lightning Origin and VF Origin using Apex, which takes separate values for a sandbox, production and developer edition.

This communication is preferred when VF pages are present in an **iFrame** inside a Lightning Component.
If VF Pages and Lightning Components are separately present on a Lightning page, then LMS should be the preferred mode of communication.