**Manifest - Web Accessible Resources in Chrome Extensions**

**Web-accessible resources** allow certain files in a Chrome extension to be accessed by web pages or other extensions. These resources can include images, scripts, stylesheets, or other assets that web pages may need.

**Why Are Web-Accessible Resources Needed?**

1. **Embedding Images and Stylesheets** – Websites may want to use images or stylesheets from an extension.
2. **Content Script Communication** – Sometimes, content scripts need to load external scripts or assets from the extension itself.
3. **Interacting with Other Extensions** – One extension may need to access resources from another installed extension.

**Security Considerations**

* By default, no resources are web accessible to prevent security issues like **fingerprinting** or **cross-site scripting (XSS) attacks**.
* Only declared resources are exposed, and access is granted only to specific websites or other extensions.

**Manifest Declaration**

To declare web-accessible resources, use the web\_accessible\_resources key in the manifest.json file.

**Basic Example**

{

"name": "Web Accessible Resources Example",

"version": "1.0",

"manifest\_version": 3,

"web\_accessible\_resources": [

{

"resources": [ "images/logo.png", "styles/style.css" ],

"matches": [ "https://example.com/\*" ]

}

]

}

**Explanation:**

* "resources" – Declares which files are web accessible (logo.png and style.css).
* "matches" – Specifies that only pages on example.com can access these files.

**Advanced Example**

{

"name": "Advanced Web Accessible Resources",

"version": "1.0",

"manifest\_version": 3,

"web\_accessible\_resources": [

{

"resources": [ "scripts/injected.js" ],

"matches": [ "https://mywebsite.com/\*" ]

},

{

"resources": [ "icons/\*.png" ],

"extension\_ids": [ "abcdefghijklmnoabcdefghijklmno" ],

"use\_dynamic\_url": true

}

]

}

**Explanation:**

1. "resources": [ "scripts/injected.js" ] – Allows a JavaScript file to be loaded from https://mywebsite.com/.
2. "resources": [ "icons/\*.png" ] – Allows all PNG files in the icons/ folder to be accessed, but **only by another extension** with ID abcdefghijklmnoabcdefghijklmno.
3. "use\_dynamic\_url": true – Uses a **dynamic session-based URL**, meaning the URL changes every time the extension is reloaded.

**Using Web-Accessible Resources in Web Pages**

Once declared, these resources can be accessed from a web page using a standard <img>, <script>, or <link> tag.

**Example 1: Using an Image from the Extension**

<img src="chrome-extension://<EXTENSION\_ID>/images/logo.png">

Where <EXTENSION\_ID> is the unique ID of the extension. You can find it in chrome://extensions/.

**Example 2: Loading an Injected Script**

<script src="chrome-extension://<EXTENSION\_ID>/scripts/injected.js"></script>

**Loading a Web-Accessible Script Dynamically**

**Example: Injecting a Script from the Extension**

Sometimes, a content script needs to inject another script into the web page. You can do this using JavaScript.

**content.js (Content Script)**

const script = document.createElement("script");

script.src = chrome.runtime.getURL("scripts/injected.js");

document.documentElement.appendChild(script);

**injected.js**

console.log("Injected script is running!");

**Manifest**

"web\_accessible\_resources": [

{

"resources": [ "scripts/injected.js" ],

"matches": [ "<all\_urls>" ]

}

]

**Explanation:**

* The content.js script is executed as part of the extension’s content scripts.
* It injects injected.js into the webpage.
* chrome.runtime.getURL("scripts/injected.js") returns the full extension URL for injected.js.

**Restricting Access to Specific Extensions**

If you only want certain extensions to access your resources, use "extension\_ids" instead of "matches".

**Example: Allowing Only Certain Extensions**

"web\_accessible\_resources": [

{

"resources": [ "data.json" ],

"extension\_ids": [ "abcdefghijklmnopqrstuvwxyza" ]

}

]

**Explanation:**

* Only the extension with the ID "abcdefghijklmnopqrstuvwxyza" can access data.json.
* Other websites and extensions **cannot** access it.

**Dynamic URLs for Enhanced Security**

Using "use\_dynamic\_url": true makes resources available **only for the current browser session**.

**Example**

"web\_accessible\_resources": [

{

"resources": [ "icons/\*.png" ],

"matches": [ "https://mysite.com/\*" ],

"use\_dynamic\_url": true

}

]

**Why Use This?**

* Prevents other websites from **hardcoding** your extension’s resource URLs.
* Makes it harder for attackers to exploit resources in case of vulnerabilities.

**Common Mistakes and How to Fix Them**

| **Mistake** | **Problem** | **Solution** |
| --- | --- | --- |
| **Forgetting to declare web\_accessible\_resources** | Resources cannot be accessed by websites | Add web\_accessible\_resources in manifest.json |
| **Using incorrect matches patterns** | Invalid match pattern error | Ensure patterns only use /\* after domain (e.g., https://example.com/\*) |
| **Accessing files using an incorrect URL format** | 404 Not Found error | Use chrome-extension://<EXTENSION\_ID>/path/to/resource |
| **Using "use\_dynamic\_url": true but expecting static URLs** | Resources become inaccessible after reload | Avoid dynamic URLs unless necessary |

**Conclusion**

Web-accessible resources are essential for extensions that need to expose assets to web pages or other extensions. However, they should be used carefully to avoid security risks.

**Key Takeaways:**

1. **Declare web-accessible resources explicitly** in manifest.json.
2. **Restrict access** to specific websites or extensions using "matches" or "extension\_ids".
3. **Use chrome.runtime.getURL()** to dynamically fetch extension resources.
4. **Consider use\_dynamic\_url: true** for added security if resources don't need a static URL.

Would you like a practical demo Chrome extension to test this? 🚀

  chrome.tabs.query({}, function (tabs) {

                    console.log(tabs);

                    tabs.forEach((tab) => {

                        console.log(tab);

                        console.log(tab.id);

                        chrome.tabs.sendMessage(tab.id, { action: "pause" }, function (response) {

                            if (!chrome.runtime.lastError) {

                                console.log(response)

                            } else {

                                console.log(chrome.runtime.lastError)

                            }

                        })

                    })

                })

   chrome.tabs.query({ currentWindow: true, active: true }, function (tabs) {

                var activeTab = tabs[0];

                chrome.tabs.sendMessage(activeTab.id, { "message": "start", startTime: startTime }).catch(console.log);

            });

*chrome.runtime.onMessage.addListener(async function (request) {*

*console.log(request);*

*if (request.message == 'start') {*

*// alert('start')*

*start()*

*// location.reload()*

*}*

        });