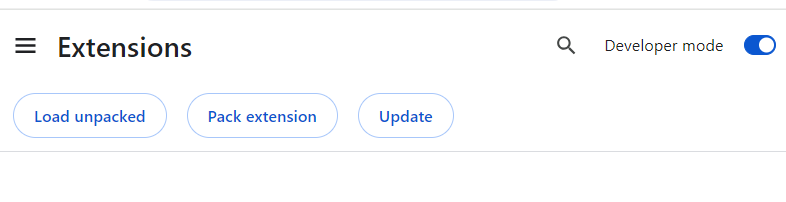
**Documentation for Multi-Monitor Extension**

**Steps to install extension in your browser :**

**[Developer Mode]**

1. On Chrome or Microsoft Edge, tap on three dots in the top right corner.
2. Click on Extensions > Manage Extension.
3. Turn on developer mode. As soon as you turn it on it expands the features to upload extension from your system.

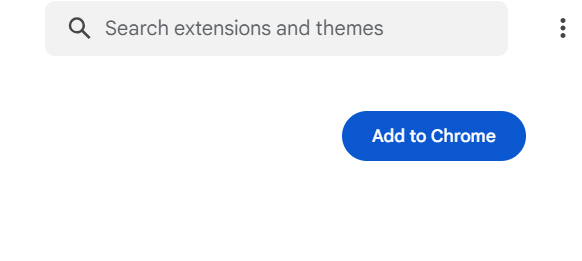


1. Click on Load unpacked and select the folder that contains your extension, the extension will be uploaded.

**[Production Mode]**

**After the extension is published, the user can directly access it from chrome web store.**

1. On Chrome or Microsoft Edge, tap on three dots in the top right corner.
2. Click on Extensions > Visit Chrome Web Store.
3. Search for the name of the extension and Add it to Chrome or Microsoft Edge



1. **The extension will be added to** Chrome or Microsoft Edge.

**Changes require in website :**

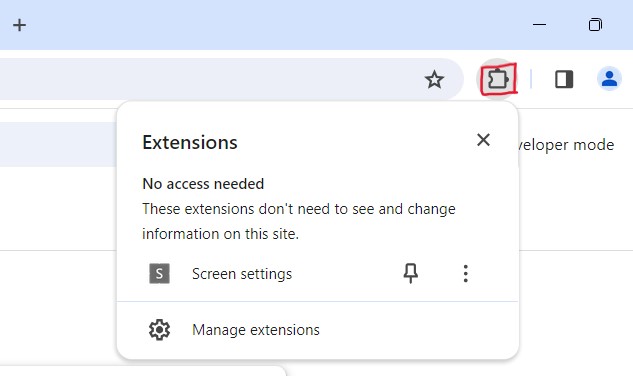
* Please add **action-secondary-screen="true"** to your anchor tag where you want your screen setting functionality to be applicable. This will allow screen setting extension to detect the tag on which click, screen setting function to be applicable

Example:

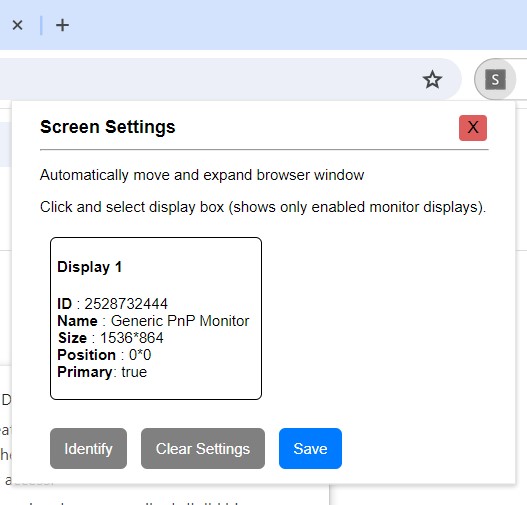
<a href=" https://www.google.com/" action-secondary-screen="true" target="\_blank"> Navigate to another page</a>

**Steps to use extension in your browser :**

**The Extension is now added to your browser click on the highlighted icon**



1. Now click on Screen Settings extension. It will open a popup displaying the monitors that are connected.



1. To use the extension, and choose to display the result on Screen 1 or Screen 2 or in both Screen 1 and Screen 2 (expanded format), you need to add an attribute to your button which opens the result in the screens as selected in the extension.

The attribute you need to add in your button is:

**action-secondary-screen="true"**

Example:

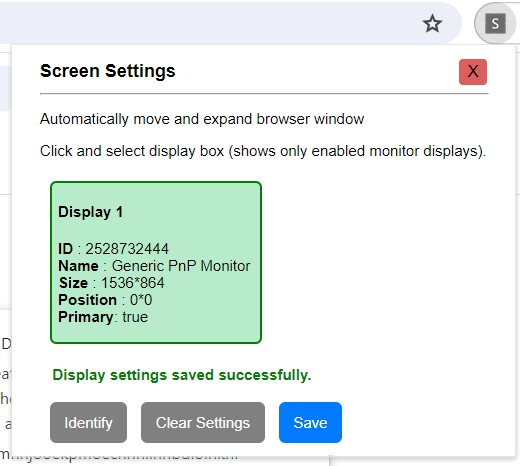
<a href=" https://www.google.com/" action-secondary-screen="true" target="\_blank"> Navigate to another page</a>

1. By clicking on **“Navigate to another page”** it will show your url on eitheron Screen 1 or Screen 2 or in both Screen 1 and Screen 2 (expanded format) as selected by the user.

**Extension:**

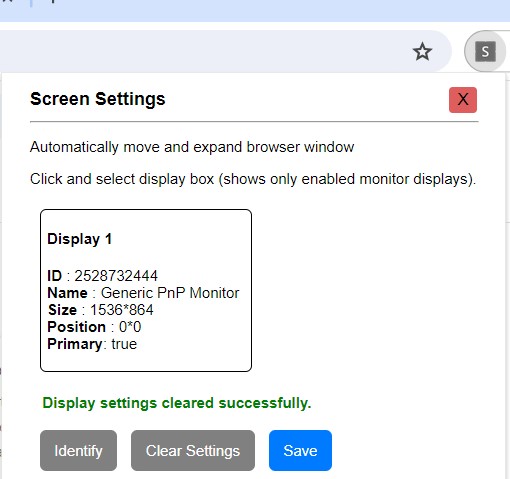
* Choosing the Display 1, and click on **Save** button, it will save your Display 1 and your result will be displayed on monitor 1.

It displays the message: “Display settings saved successfully.”

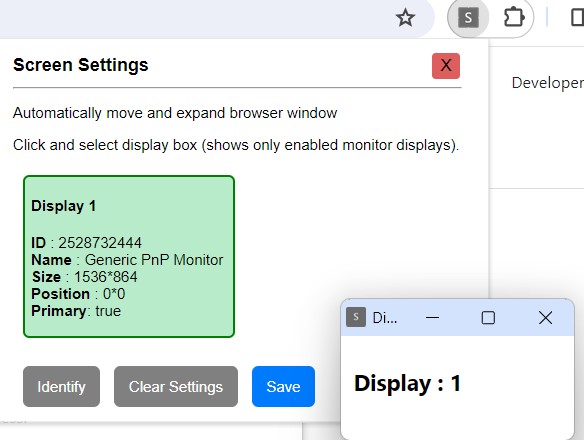


* Click on the **Clear Settings** button to clear all the selected displays.

It displays the message: “Display settings cleared successfully.”



* Click on the **Identify** button to open a popup about display identification.



**Extension Workflow:**

**[Content.js]**

As the extension is installed in the Chrome/Microsoft Edge, it gets activated.

Whenever a website is loaded a function **addEventForScreenSetting()** is called.

Inside this function there is a constant that gets all the **‘a’** tags. The **addEventForScreenSetting** function iterates through all the **<a>** (anchor) elements in the document**.**

If an anchor element has the attribute **action-secondary-screen** set to "true" and does not have the attribute **data-event-listener-added**, it adds a click event listener to that anchor element.

The event listener prevents the default click behavior, and instead, it sends a message to the Chrome extension using **chrome.runtime.sendMessage** with information about the clicked link and screen dimensions.

This callback sends message consisting link that needs to be opened on button click, and the information about screen like height, width etc.

**addEventForScreenSetting()** method is called twice

* + - 1. Firstly when the website is loaded
      2. If there is any change in the DOM.

It uses the MutationObserver API to detect changes in the DOM (Document Object Model). When a change is detected, it calls the **executeOnDOMChange** function, which in turn calls the **addEventForScreenSetting** function.

Chrome APIs are used to send message and receive message.

executeOnDOMChange

Window.onLoad

addEventForScreenSettings()

* Get screenInfo
* Iterates <a> element
* Adds event listener on click

‘a’ has attribute **action-secondary-screen**

Adds an event listener which therefore sends message using chrome.runtime.sendMessage API

As content.js uses **chrome.runtime.sendMessage** To send message and background.js uses **chrome.runtime.onMessage.addListener** To listen for the messages. This is how they are linked.

**[Background.js]**

There is a Chrome extension background script that listens for messages using **chrome.runtime.onMessage.addListener**. When a message is received, it checks for a specific property (**customChangeScreen**) in the message.

It also uses the **chrome.storage.sync.get**  api to get the storage data from “**selectedDisplays**” i.e. the displays that are selected in the extension popup.

If this property is present, it triggers the **OpenScreen** function, which opens a URL in a new tab or window based on the availability of secondary screens.

The **OpenScreen** function is a part of a Chrome extension background script, and its purpose is to handle the opening of a specified URL on secondary screens based on user preferences and available displays.

* The function starts by checking for the availability of secondary screens based on user-selected displays.
* It sorts the selected displays by their index and maps them to obtain an array of display IDs (displayIds).
* It then uses the **chrome.system.display.getInfo** API to get information about all available displays.
* The function checks various conditions to determine the appropriate action:

1. If there is only one selected display, it checks if the source screen overlaps with the display horizontally. If yes, it opens the URL in a new tab; otherwise, it opens a new window on the selected display.
2. If there are multiple selected displays, it opens a new window on the first selected display and adjusts the width based on the total width of selected displays.
3. If there are no selected displays or only one display, it opens the URL in a new tab.

* The function utilizes the “**CheckSourceAndDisplayIsOne”** function to check for horizontal overlap between the source screen and the display.
* The Chrome extension APIs **chrome.tabs.create** and **chrome.windows.create** are used to create new tabs or windows, and **chrome.windows.update** is used to update the properties of the created windows.

**[Popup.js]**

It is Chrome extension popup script. A tool for managing and configuring display settings.

**monitorArray**: This array is used to store information about selected displays. It is initially an empty array.

It contains an object consisting {

DisplayIndex [id of the display]

Indexid [index to maintain sequence of selected monitors]

}

The code ensures that users select displays in a sequential order and checks if selected displays have the same resolution. The UI provides visual feedback on the selected displays

The **SetDataChromeStorage** function is responsible for storing the **monitorArray** data, which contains information about selected displays, in the Chrome storage using the **chrome.storage.sync.set** method. This function is a part of a Chrome extension and is typically triggered when the user performs an action that should persist data across different instances of the extension.

It takes an object as an argument, where the keys are the names of the items to store, and the values are the data to be stored. In this scase, it's storing the **monitorArray** under the key “**selectedDisplays”**.

displayInfo has more than 1 element and displayIds is not null or undefinded and displayIds contains more than one element

Chrome.system.display.getInfo displayInfo

selectedDisplays Sort and Map DisplayIds

selectedDisplays is not undefined and selecteDisplays is not null

Window.left is less than display.bounds.left

Update total width by adding display.bounds.width

* Find display
* Create a new window in First Display
* Find display.
* CheckSourceAndDisplayIsOne()

displayIds has only one element

Update new window Maximize and adjust width Update window position and width final state

Update totalWidth

Create new window and open url in selected screen

selectedDisplays found

Create tab in the same screen as source screen

CheckSourceAndDisplayisOne is true

Create tab in the same screen and open in new tab.

OpenScreen()