chrome

# chrome.devtools.inspectedWindow

| | |
|---|---|
| Description: | Use the `chrome.devtools.inspectedWindow` API to interact with the inspected window: obtain the tab ID for the inspected page, evaluate the code in the context of the inspected window, reload the page, or obtain the list of resources within the page. |
| Availability: | Since Chrome 31. |

Use `chrome.devtools.inspectedWindow` to interact with the inspected window: obtain the tab ID for the inspected page, evaluate the code in the context of inspected window, reload the page, or obtain the list of resources within the page.

See **DevTools APIs summary** for general introduction to using Developer Tools APIs.

## Overview

The `tabId` property provides the tab identifier that you can use with the `chrome.tabs.*` API calls. However, please note that `chrome.tabs.*` API is not exposed to the Developer Tools extension pages due to security considerations — you will need to pass the tab ID to the background page and invoke the `chrome.tabs.*` API functions from there.

The `reload` method may be used to reload the inspected page. Additionally, the caller can specify an override for the user agent string, a script that will be injected early upon page load, or an option to force reload of cached resources.

Use the `getResources` call and the `onResourceContent` event to obtain the list of resources (documents, stylesheets, scripts, images etc) within the inspected page. The `getContent` and `setContent` methods of the `Resource` class along with the `onResourceContentCommitted` event may be used to support modification of the resource content, for example, by an external editor.

## Executing Code in the Inspected Window

The `eval` method provides the ability for extensions to execute JavaScript code in the context of the inspected page. This method is powerful when used in the right context and dangerous when used inappropriately. Use the `tabs.executeScript` method unless you need the specific functionality that the `eval` method provides.

Here are the main differences between the `eval` and `tabs.executeScript` methods:

- The `eval` method does not use an isolated world for the code being evaluated, so the JavaScript state of the inspected window is accessible to the code. Use this method when access to the JavaScript state of the inspected page is required.
- The execution context of the code being evaluated includes the **Developer Tools console API**. For example, the code can use `inspect` and `$0`.
- The evaluated code may return a value that is passed to the extension callback. The returned value has to be a valid JSON object (it may contain only primitive JavaScript types and acyclic references to other JSON objects). *Please observe extra care while processing the data received from the inspected page — the execution context is essentially controlled by the inspected page; a malicious page may affect the data being returned to the extension.*

> **Important:** Due to the security considerations explained above, the `tabs.executeScript` method is the preferred way for an extension to access DOM data of the inspected page in cases where the access to JavaScript state of the inspected page is not required.

Note that a page can include multiple different JavaScript execution contexts. Each frame has its own context, plus an additional context for each extension that has content scripts running in that frame.

By default, the `eval` method executes in the context of the main frame of the inspected page.

The `eval` method takes an optional second argument that you can use to specify the context in which the code is evaluated. This *options* object can contain one or more of the following keys:

`frameURL`
: Use to specify a frame other than the inspected page's main frame.

`contextSecurityOrigin`
: Use to select a context within the specified frame according to its **web origin**.

`useContentScriptContext`
: If true, execute the script in the same context as the extensions's content scripts. (Equivalent to specifying the extensions's own web orgin as the context security origin.) This can be used to exchange data with the content script.

## Examples

The following code checks for the version of jQuery used by the inspected page:

```
chrome.devtools.inspectedWindow.eval(
    "jQuery.fn.jquery",
     function(result, isException) {
       if (isException)
         console.log("the page is not using jQuery");
       else
         console.log("The page is using jQuery v" + result);
     }
);
```

You can find more examples that use Developer Tools APIs in **Samples**.

## Summary

| |
|---|
| **Types** |
| **Resource** |
| **Properties** |
| **tabId** |
| **Methods** |
| **eval** − `chrome.devtools.inspectedWindow.eval(string expression, object options, function callback)` |
| **reload** − `chrome.devtools.inspectedWindow.reload(object reloadOptions)` |
| **getResources** − `chrome.devtools.inspectedWindow.getResources(function callback)` |
| **Events** |

| |
|---|
| onResourceAdded |
| onResourceContentCommitted |

## Types

### Resource

A resource within the inspected page, such as a document, a script, or an image.

| properties | | |
|---|---|---|
| string | url | The URL of the resource. |

**methods**

**getContent**
```
Resource.getContent(function callback)
```

Gets the content of the resource.

| Parameters | | |
|---|---|---|
| function | callback | A function that receives resource content when the request completes.<br>The *callback* parameter should be a function that looks like this:<br>`function(string content, string encoding) {...};`<br><br>string \| content \| Content of the resource (potentially encoded).<br>string \| encoding \| Empty if the content is not encoded, encoding name otherwise. Currently, only base64 is supported. |

**setContent**
```
Resource.setContent(string content, boolean commit, function callback)
```

Sets the content of the resource.

| Parameters | | |
|---|---|---|
| string | content | New content of the resource. Only resources with the text type are currently supported. |
| boolean | commit | True if the user has finished editing the resource, and the new content of the resource should be persisted; false if this is a minor change sent in progress of the user editing the resource. |
| function | (optional) callback | A function called upon request completion.<br>If you specify the *callback* parameter, it should be a function that looks like this:<br>`function(object error) {...};` |

| | | | |
|---|---|---|---|
| | | object | (optional) error | Set to undefined if the resource content was set successfully; describes error otherwise. |

## Properties

| | | |
|---|---|---|
| integer | `chrome.devtools.inspectedWindow.tabId` | The ID of the tab being inspected. This ID may be used with chrome.tabs.* API. |

## Methods

### eval

```
chrome.devtools.inspectedWindow.eval(string expression, object options, function callback)
```

Evaluates a JavaScript expression in the context of the main frame of the inspected page. The expression must evaluate to a JSON-compliant object, otherwise an exception is thrown. The eval function can report either a DevTools-side error or a JavaScript exception that occurs during evaluation. In either case, the `result` parameter of the callback is `undefined`. In the case of a DevTools-side error, the `isException` parameter is non-null and has `isError` set to true and `code` set to an error code. In the case of a JavaScript error, `isException` is set to true and `value` is set to the string value of thrown object.

**Parameters**

| | | |
|---|---|---|
| string | expression | An expression to evaluate. |
| object | (optional) options | Since Chrome 38. The options parameter can contain one or more options. |

| | | |
|---|---|---|
| string | (optional) frameURL | If specified, the expression is evaluated on the iframe whose URL matches the one specified. By default, the expression is evaluated in the top frame of the inspected page. |
| boolean | (optional) useContentScriptContext | Evaluate the expression in the context of the content script of the calling extension, provided that the content script is already injected into the inspected page. If not, the expression is not evaluated and the callback is invoked with the exception parameter set to an object that has the `isError` field set to true and the `code` field set to `E_NOTFOUND`. |
| string | (optional) contextSecurityOrigin | Evaluate the expression in the context of a content script of an extension that matches the specified origin. If given, contextSecurityOrigin overrides the 'true' setting on userContentScriptContext. |

|  |  |  |
|---|---|---|
| function | (optional) callback | A function called when evaluation completes.<br>If you specify the *callback* parameter, it should be a function that looks like this:<br>`function(object result, object exceptionInfo) {...};` |

|  |  |  |
|---|---|---|
| object | result | The result of evaluation. |
| object | exceptionInfo | An object providing details if an exception occurred while evaluating the expression. |

|  |  |  |
|---|---|---|
| boolean | isError | Set if the error occurred on the DevTools side before the expression is evaluated. |
| string | code | Set if the error occurred on the DevTools side before the expression is evaluated. |
| string | description | Set if the error occurred on the DevTools side before the expression is evaluated. |
| array of any | details | Set if the error occurred on the DevTools side before the expression is evaluated, contains the array of the values that may be substituted into the description string to provide more information about the cause of the error. |
| boolean | isException | Set if the evaluated code produces an unhandled exception. |
| string | value | Set if the evaluated code produces an unhandled exception. |

## reload

```
chrome.devtools.inspectedWindow.reload(object reloadOptions)
```

Reloads the inspected page.

| **Parameters** |  |  |
|---|---|---|
| object | (optional) reloadOptions |  |

|  |  |  |
|---|---|---|
| boolean | (optional) ignoreCache | When true, the loader will bypass the cache for all inspected page resources loaded before the `load` event is fired. The effect is similar to pressing Ctrl+Shift+R in the inspected window or within the Developer Tools window. |
| string | (optional) userAgent | If specified, the string will override the value of the `User-Agent` HTTP header that's sent while loading the resources of the inspected page. The string will also override the value of the `navigator.userAgent` property that's returned to any scripts that are |

| | | | | |
|---|---|---|---|---|
| | | | | running within the inspected page. |
| | | string | (optional) injectedScript | If specified, the script will be injected into every frame of the inspected page immediately upon load, before any of the frame's scripts. The script will not be injected after subsequent reloads—for example, if the user presses Ctrl+R. |

### getResources

```
chrome.devtools.inspectedWindow.getResources(function callback)
```

Retrieves the list of resources from the inspected page.

| Parameters | | |
|---|---|---|
| function | callback | A function that receives the list of resources when the request completes.<br>The *callback* parameter should be a function that looks like this:<br>`function(array of Resource resources) {...};` |

| | | |
|---|---|---|
| array of **Resource** | resources | The resources within the page. |

## Events

### onResourceAdded

Fired when a new resource is added to the inspected page.

**addListener**

```
chrome.devtools.inspectedWindow.onResourceAdded.addListener(function callback)
```

| Parameters | | |
|---|---|---|
| function | callback | The *callback* parameter should be a function that looks like this:<br>`function( Resource resource) {...};` |

| | | |
|---|---|---|
| **Resource** | resource | |

### onResourceContentCommitted

Fired when a new revision of the resource is committed (e.g. user saves an edited version of the resource in the Developer Tools).

**addListener**

```
chrome.devtools.inspectedWindow.onResourceContentCommitted.addListener(function callback)
```

| Parameters |
|---|

| | | |
|---|---|---|
| function | callback | The *callback* parameter should be a function that looks like this:<br>`function( Resource resource, string content) {...};`<br><br>**Resource** \| resource \| <br>string \| content \| New content of the resource. |