



eBook Gratuit

APPRENEZ

# google-chrome- extension

eBook gratuit non affilié créé à partir des  
**contributeurs de Stack Overflow.**

#google-

chrome-

extension

# Table des matières

À propos.....	1
<b>Chapitre 1: Démarrer avec google-chrome-extension.....</b>	<b>2</b>
Remarques.....	2
TODO: Brève description des extensions Chrome.....	2
<b>Documentation officielle.....</b>	<b>2</b>
<b>Lectures complémentaires.....</b>	<b>2</b>
TODO: Remplir avec des liens vers des sujets importants de présentation.....	2
Exemples.....	2
Exemple minimum absolu.....	2
Page de fond.....	3
Scripts de contenu.....	4
<b>Voir également.....</b>	<b>4</b>
Page d'options.....	5
Version 2.....	5
Version 1 ( obsolète ).....	6
Espace de rangement.....	6
<b>Documentation officielle.....</b>	<b>6</b>
Créer un nouvel onglet.....	6
<b>Chapitre 2: Débogage des extensions Chrome.....</b>	<b>8</b>
Exemples.....	8
Utilisation des outils Developer pour déboguer votre extension.....	8
<b>Chapitre 3: Intégration de l'outil de développement.....</b>	<b>10</b>
Exemples.....	10
Point de repère programmatique.....	10
Débogage de la page d'arrière-plan / du script.....	10
Déboguer la fenêtre contextuelle.....	11
<b>Chapitre 4: manifest.json.....</b>	<b>12</b>
Remarques.....	12
<b>Documentation officielle.....</b>	<b>12</b>

<b>Format</b>	<b>12</b>
Examples	13
Manifest.json minimum absolu	13
Obtenir un manifeste à partir du code d'extension	13
<b>Chapitre 5: Message passant</b>	<b>14</b>
Remarques	14
<b>Documentation officielle</b>	<b>14</b>
Examples	14
Envoyer une réponse de manière asynchrone	14
<b>Chapitre 6: Pages de fond</b>	<b>16</b>
Examples	16
Déclarer la page de fond dans le manifeste	16
<b>Chapitre 7: Portage vers / depuis Firefox</b>	<b>17</b>
Remarques	17
Examples	17
Portage via WebExtensions	18
<b>Extensions compatibles basées sur WebExtension</b>	<b>18</b>
Une extension simple qui peut fonctionner dans Firefox et Google Chrome	18
<b>Si l'add-on actuel est basé sur le SDK complémentaire ou XUL</b>	<b>20</b>
<b>Chapitre 8: Scripts de contenu</b>	<b>22</b>
Remarques	22
<b>Documentation officielle</b>	<b>22</b>
Examples	22
Déclaration de scripts de contenu dans le manifeste	22
<b>Exemple minimal</b>	<b>22</b>
<b>Note importante</b>	<b>23</b>
Injection de scripts de contenu à partir d'une page d'extension	23
Exemple minimal	23
Code en ligne	23
Choisir l'onglet	23
<b>Autorisations</b>	<b>24</b>

<b>Vérification des erreurs</b> .....	<b>24</b>
Plusieurs scripts de contenu dans le manifeste.....	24
<b>Mêmes conditions, plusieurs scripts</b> .....	<b>24</b>
<b>Mêmes scripts, plusieurs sites</b> .....	<b>24</b>
<b>Différents scripts ou différents sites</b> .....	<b>25</b>
<b>Crédits</b> .....	<b>26</b>

---

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [google-chrome-extension](#)

It is an unofficial and free google-chrome-extension ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official google-chrome-extension.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Chapitre 1: Démarrer avec google-chrome-extension

## Remarques

**TODO: Brève description des extensions Chrome**

---

## Documentation officielle

- [Que sont les extensions?](#) (centre de documentation)
- [Didacticiel de mise en route](#) (didacticiel de base)
- [Vue d'ensemble](#)
- [API JavaScript](#) (liste complète des API `chrome.*` )

---

## Lectures complémentaires

**TODO: Remplir avec des liens vers des sujets importants de présentation**

## Exemples

### Exemple minimum absolu

Toute extension Chrome démarre en tant *qu'extension décompressée* : un dossier contenant les fichiers de l'extension.

Un fichier qu'il doit contenir est `manifest.json` , qui décrit les propriétés de base de l'extension. La plupart des propriétés de ce fichier sont facultatives, mais voici un fichier `manifest.json` minimum absolu:

```
{
  "manifest_version": 2,
  "name": "My Extension",
  "version": "1.0"
}
```

Créez un dossier (par exemple, `myExtension` ) quelque part, ajoutez `manifest.json` comme indiqué ci-dessus.

Ensuite, vous devez charger l'extension dans Chrome.

1. Ouvrez la page `chrome://extensions/` , accessible par **Menu> Plus d'outils> Extensions** .
2. Activez le **mode développeur** avec une case à cocher en haut à droite, si ce n'est déjà fait.
3. Cliquez sur le bouton **Charger une extension décompressée ...** et sélectionnez le dossier `myExtension` créé.

## Extensions

2  Developer mode

3

Load unpacked extension...

Pack extension...

Update extensions...

C'est tout! Votre première extension est chargée par Google Chrome:



My Extension 1.0

Enabled



[Details](#) [Reload \(Ctrl+R\)](#)

ID: `gdgijjhpbdlebnhblpfplpolomkjbmm`

Loaded from: `C:\Devel\myExtension`

Allow in incognito

Bien sûr, cela ne fait rien pour le moment, c'est donc un bon moment pour lire un [aperçu de l'architecture des extensions](#) pour commencer à ajouter des pièces dont vous avez besoin.

**Important:** lorsque vous apportez des modifications à votre extension, n'oubliez pas de revenir à `chrome://extensions/` et appuyez sur le lien **Recharger** de votre extension après avoir apporté des modifications. En cas de scripts de contenu, rechargez également la page cible.

## Page de fond

Les pages d'arrière-plan sont des pages implicites contenant des scripts d'arrière-plan. Un script d'arrière-plan est un script long et unique permettant de gérer une tâche ou un état. Il existe pour la durée de vie de votre extension et une seule instance à la fois est active.

Vous pouvez le déclarer comme ceci dans votre `manifest.json` :

```
"background": {
  "scripts": ["background.js"]
}
```

Une page d'arrière-plan sera générée par le système d'extension qui inclut chacun des fichiers répertoriés dans la propriété `scripts`.

Vous avez accès à toutes les API `chrome.*` Autorisées.

Il existe deux types de pages d'arrière-plan: **les pages d'arrière-plan persistantes** qui sont toujours ouvertes et les **pages d'événements** ouvertes et fermées selon les besoins.

Si vous voulez que votre page d'arrière-plan ne soit pas persistante, il vous suffit de définir le paramètre `-flag persistent` sur `false`:

```
"background": {
  "scripts": ["eventPage.js"],
  "persistent": false
}
```

Ce script d'arrière-plan n'est actif que si un événement est déclenché sur lequel un auditeur est enregistré. En général, vous utilisez un `addListener` pour l'enregistrement.

Exemple: L'application ou l'extension est d'abord installée.

```
chrome.runtime.onInstalled.addListener(function() {
  console.log("The Extension is installed!");
});
```

## Scripts de contenu

Un **script de contenu** est un code d'extension qui s'exécute à côté d'une page normale.

Ils ont un accès complet au DOM de la page Web (et sont, en fait, **la seule partie de l'extension pouvant accéder au DOM de la page**), mais le code JavaScript est isolé, un concept appelé [Isolated World](#). Chaque extension a son propre contexte JavaScript de script de contenu invisible pour les autres et la page, empêchant les conflits de code.

Exemple de définition dans [manifest.json](#):

```
"content_scripts": [
  {
    "matches": ["http://www.stackoverflow.com/*"],
    "css": ["style.css"],
    "js": ["jquery.js", "myscript.js"]
  }
]
```

Les attributs ont la signification suivante:

Attribut	La description
allumettes	Spécifie dans quelles pages ce script de contenu sera injecté. Suit le format du <a href="#">motif de correspondance</a> .
css	Liste des fichiers CSS à injecter dans les pages correspondantes.
js	Liste des fichiers JS à injecter dans les pages correspondantes. <b>Exécuté dans l'ordre listé.</b>

Les scripts de contenu peuvent également être injectés à la demande en utilisant `chrome.tabs.executeScript`, appelé [Injection programmatique](#).

## Voir également

- Documentation officielle: [Scripts de contenu](#)
- Documentation de débordement de pile: [scripts de contenu](#)

## Page d'options

Les pages d'options permettent à l'utilisateur de conserver les paramètres de votre extension.

## Version 2

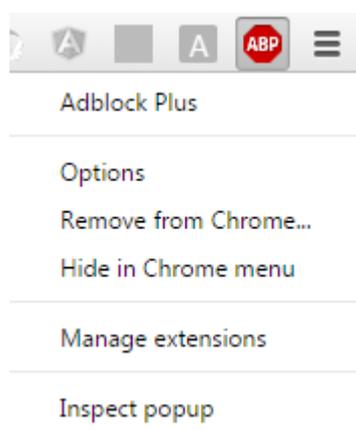
Depuis Chrome 40, il est possible d'avoir la page d'option comme dialogue prédéfini sur *les extensions chrome: //* .

La manière de définir une page d'option dans le `manifest.json` est la suivante:

```
"options_ui": {  
  "page": "options.html",  
  "chrome_style": true  
}
```

Cette page d'options se comportera comme un dialogue, elle s'ouvrira comme une fenêtre contextuelle, où les `options.html` seront affichées. `chrome_style` appliquera une feuille de style Chrome pour des raisons de cohérence de style à votre page d'options.

Les options seront automatiquement exposées via le menu contextuel du **bouton d'extension** ou la page `chrome: // extensions` .



Adblock Plus 1.12.1

Enabled



Used by over 50 million people, a free ad blocker that blocks ALL annoying ads, malware and tracking.

[Details](#) [Options](#)

ID: cfhdojbkjhnlbpbkdaibdccddilifddb

Inspect views: [background page](#)

Allow in incognito

Vous pouvez également [ouvrir la page d'options par programmation](#) , par exemple depuis une

interface utilisateur contextuelle:

```
chrome.runtime.openOptionsPage();
```

## Version 1 ( obsolète )

Exemple de définition dans `manifest.json` :

```
"options_page": "options.html"
```

Il est recommandé d'utiliser la version 2 car le comportement `options_ui` sera bientôt appliqué aux pages d'options de la version 1.

## Espace de rangement

Normalement, les paramètres doivent persister, il est donc recommandé d'utiliser l'API

`chrome.storage` . Les autorisations peuvent être déclarées comme ceci dans le `manifest.json` :

```
"permissions": [  
  "storage"  
]
```

---

## Documentation officielle

- [Page d'options - Version 1](#)
- [Page d'options - Version 2](#)
- [API de stockage](#)

### Créer un nouvel onglet

Dans le code d'extension, vous pouvez utiliser n'importe quelle API `chrome.*` Si vous préférez les autorisations requises. En outre, certaines API ne fonctionnent qu'à partir de pages d'arrière-plan, et certaines API ne fonctionnent qu'à partir de scripts de contenu.

Vous pouvez utiliser la plupart des méthodes `chrome.tabs` déclarant des autorisations. Maintenant, nous nous concentrons sur `chrome.tabs.create`

Remarque: Le nouvel onglet sera ouvert sans avertissement `popup` .

```
chrome.tabs.create({  
  url:"http://stackoverflow.com",  
  selected:false // We open the tab in the background  
})
```

Vous pouvez en apprendre plus sur l'objet `tab`, dans le [développeur chrome officiel](#)

Lire Démarrer avec google-chrome-extension en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/787/demarrer-avec-google-chrome-extension>

---

# Chapitre 2: Débogage des extensions Chrome

## Exemples

### Utilisation des outils Developer pour déboguer votre extension

Une extension chromée est séparée en 4 parties maximum:

- la page de fond
- la page popup
- un ou plusieurs scripts de contenu
- la page d'options

Chaque partie, étant intrinsèquement séparée, nécessite un débogage individuel.

**Gardez à l'esprit que ces pages sont séparées, ce qui signifie que les variables ne sont pas directement partagées entre elles et qu'une `console.log()` dans l'une de ces pages ne sera pas visible dans les journaux des autres parties.**

#### ***En utilisant les devtools de chrome:***

Les extensions Chrome sont déboguées comme pour les autres applications Web et les pages Web. Le débogage se fait le plus souvent à l'aide de l'inspecteur devtools de chrome ouvert en utilisant respectivement le raccourci clavier pour windows et mac: `ctrl + shift + i` et `cmd + shift + i` ou en cliquant avec le bouton droit sur la page et en sélectionnant inspect.

De l'inspecteur, un développeur peut vérifier les éléments HTML et la manière dont css les affecte, ou utiliser la console pour inspecter les valeurs des variables javascript et lire les sorties de n'importe quelle `console.log()` définie par les développeurs.

Vous trouverez plus d'informations sur l'utilisation de l'inspecteur sur [Chrome Devtools](#) .

#### ***Inspection de la fenêtre contextuelle, de la page d'options et des autres pages accessibles à l'aide de chrome: `//....votreExtensionId.../`:***

La *page contextuelle* et la *page d' options* peuvent être consultées en les inspectant simplement lorsqu'elles sont ouvertes.

Les pages HTML supplémentaires faisant partie de l'extension, mais qui ne sont ni la fenêtre contextuelle ni la page d'options, sont également déboguées de la même manière.

#### ***Inspection de la page de fond:***

Pour accéder à votre *page d'arrière - plan*, vous devez d'abord accéder à la page d'extension chrome à l'adresse <chrome://extensions/> . Assurez-vous que la case «*Mode développeur*» est

activée.

## Extensions

Developer mode

Load unpacked extension...

Pack extension...

Update extensions now

Cliquez ensuite sur votre script d'arrière-plan en regard de «*Inspecter les vues*» pour inspecter votre page d'arrière-plan.



My Extension 2.0

Enabled



Do Stuff!

[Details](#) [Reload \(Ctrl+R\)](#)

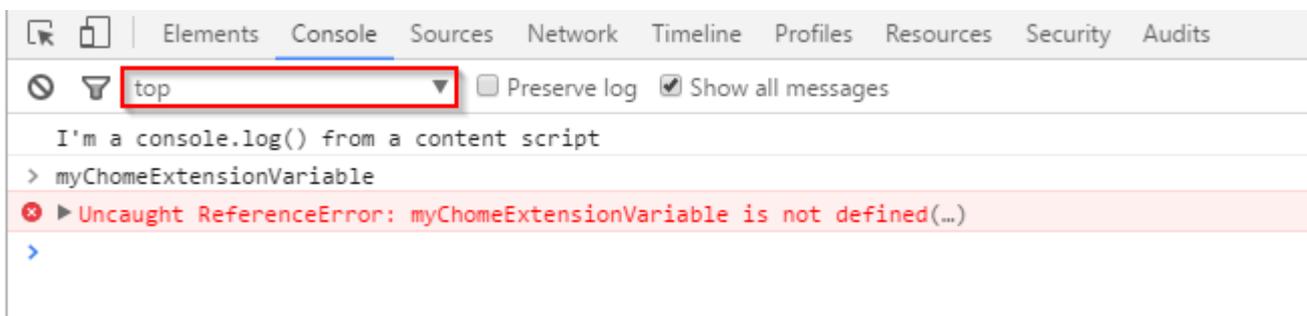
ID: abglhdcgbjkbhalmfachdimiapkfedl

Loaded from: C:\make\_page\_red

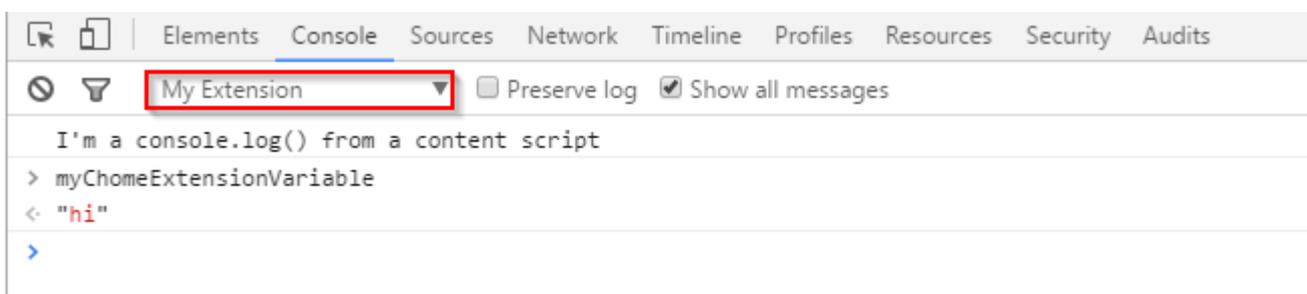
Inspect views: **background page**

### Inspection des scripts de contenu:

Les scripts de contenu sont exécutés le long des sites Web dans lesquels ils ont été insérés. Vous pouvez inspecter le script de contenu en inspectant d'abord le site Web où le script de contenu est inséré. Dans la console, vous pourrez voir tous les `console.log()` par votre extension, mais vous ne pourrez pas modifier ou inspecter les variables du script de contenu.



Pour résoudre ce problème, vous devez cliquer sur le menu déroulant généralement défini sur 'top' et sélectionner votre extension dans la liste des extensions.



De là, vous aurez accès aux variables de votre extension.

Lire Débogage des extensions Chrome en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/5730/debogage-des-extensions-chrome>

# Chapitre 3: Intégration de l'outil de développement

## Exemples

### Point de repère programmatique

Ajoutez l'instruction de débogueur dans votre script de contenu

```
var foo = 1;
debugger;

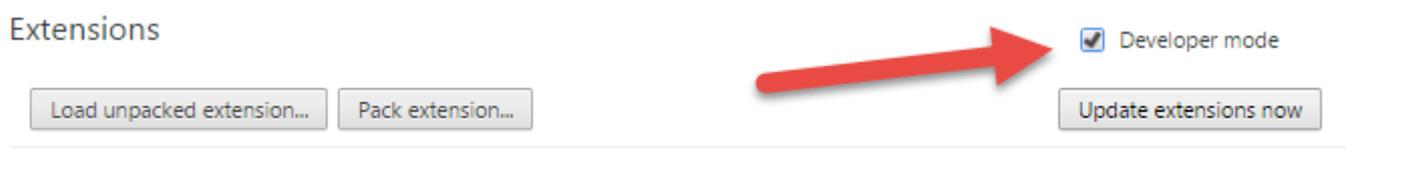
foo = 2;
```

Ouvrez l'outil de développement sur la page Web où votre script de contenu est injecté pour voir la pause d'exécution du code sur ces lignes.

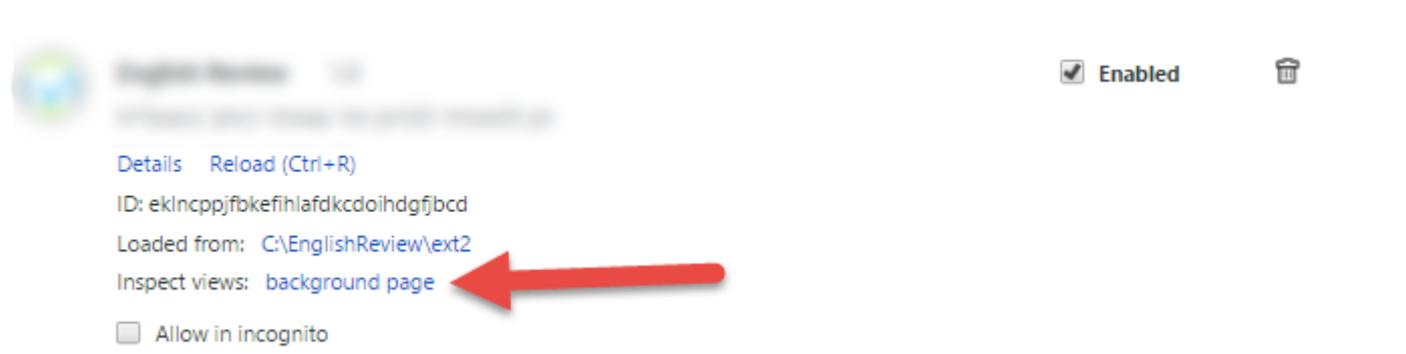
### Débogage de la page d'arrière-plan / du script

Le script d'arrière-plan est comme tout autre code JavaScript. Vous pouvez le déboguer en utilisant les mêmes outils que vous déboguez le code JavaScript dans Chrome.

Pour ouvrir les outils de développement Chrome, accédez à `chrome://extensions` et activez le **mode développeur** :



Vous pouvez maintenant déboguer toute extension ayant une page d'arrière-plan ou un script. Faites défiler jusqu'à l'extension que vous souhaitez déboguer et cliquez sur le lien de la **page d'arrière - plan** pour l'inspecter.



**Conseil:** Pour recharger l'extension, vous pouvez appuyer sur `F5` dans la fenêtre des outils de

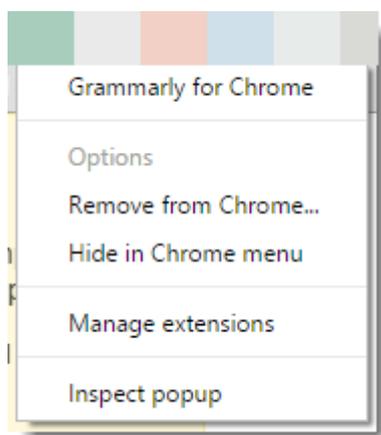
développement. Vous pouvez placer des points d'arrêt dans le code d'initialisation avant de recharger.

**Conseil:** Si vous cliquez avec le bouton droit de la souris sur le bouton d'action de l'extension et sélectionnez "Gérer les extensions", la page `chrome://extensions` défilera jusqu'à cette extension.

## Déboguer la fenêtre contextuelle

Vous avez deux façons de déboguer la fenêtre contextuelle. Les deux méthodes sont en utilisant les Chrome DevTools.

**Option 1:** cliquez avec le bouton droit sur le bouton d'action de l'extension et choisissez **Inspecter la fenêtre contextuelle**.



**Option 2:** Ouvrez la fenêtre contextuelle directement dans votre navigateur sous forme d'onglet.

Par exemple, si votre identifiant d'extension est `abcdefghijklmnop` et que votre fichier html popup est `popup.html`. Allez à l'adresse et accédez à:

```
chrome-extension://abcdefghijklmnop/popup.html
```

Maintenant, vous voyez le groupe en onglet régulier. Et vous pouvez appuyer sur `F12` pour ouvrir les outils de développement.

Lire [Intégration de l'outil de développement en ligne](https://riptutorial.com/fr/google-chrome-extension/topic/5938/integration-de-l-outil-de-developpement): <https://riptutorial.com/fr/google-chrome-extension/topic/5938/integration-de-l-outil-de-developpement>

---

# Chapitre 4: manifest.json

## Remarques

---

## Documentation officielle

[Format de fichier manifeste](#)

---

## Format

Le fichier manifeste est écrit au format **JSON** (JavaScript Object Notation).

Ce format diffère des règles plus souples d'écriture des littéraux d'objet dans le code JavaScript. Parmi les différences importantes:

- Chaque nom de clé et littéral de chaîne **doit être entre guillemets** .
  - Correct: `"key": "value"`
  - Wrong: `key: "value"`, `'key': 'value'`
- **Aucun commentaire** n'est autorisé par le format.
  - Wrong: `"key": "value" // This controls feature foo`
- Règles strictes de virgule: **éléments séparés par des virgules, pas de virgule** .
  - Correct:

```
{
  "foo": "bar",
  "baz": "qux"
}
```

- Mauvais (virgule manquant):

```
{
  "foo": "bar"
  "baz": "qux"
}
```

- Mauvais (virgule):

```
{
  "foo": "bar",
  "baz": "qux",
}
```

```
}
```

## Exemples

### Manifest.json minimum absolu

`manifest.json` fournit des informations sur l'extension, telles que les fichiers les plus importants et les fonctionnalités pouvant être utilisées par l'extension. Parmi les champs de manifeste pris en charge pour les extensions, les **trois** suivants sont requis.

```
{  
  "manifest_version": 2,  
  "name": "My Extension",  
  "version": "1.0"  
}
```

### Obtenir un manifeste à partir du code d'extension

`chrome.runtime.getManifest()` renvoie le manifeste de l'extension sous la forme d'un objet analysé.

Cette méthode fonctionne à la fois sur les scripts de contenu et sur toutes les pages d'extension, elle ne nécessite aucune autorisation,

Exemple, obtention de la chaîne de version de l'extension:

```
var version = chrome.runtime.getManifest().version;
```

Lire `manifest.json` en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/948/manifest-json>

---

# Chapitre 5: Message passant

## Remarques

---

## Documentation officielle

- [Message passant](#)
- [Messagerie native](#)
- [API `chrome.runtime`](#) (la plupart des fonctions de messagerie et tous les événements de messagerie)

## Exemples

### Envoyer une réponse de manière asynchrone

En essayant d'envoyer une réponse de manière asynchrone à partir `chrome.runtime.onMessage` rappel `chrome.runtime.onMessage` nous pourrions essayer ce **mauvais code** :

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
  $.ajax({
    url: 'https://www.google.com',
    method: 'GET',
    success: function(data) {
      // data won't be sent
      sendResponse(data);
    },
  },
  });
});
```

Cependant, nous trouverions que les `data` ne sont jamais envoyées. Cela se produit parce que nous avons placé `sendResponse` dans un appel asynchrone `ajax`, lorsque la méthode de `success` est exécutée, le canal de message a été fermé.

**La solution serait simple**, à condition que nous retournions explicitement `return true;` à la fin du rappel, qui indique que nous souhaitons envoyer une réponse de manière asynchrone, le canal de messages restera ouvert à l'autre extrémité (appelant) jusqu'à `sendResponse` que `sendResponse` soit exécuté.

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
  $.ajax({
    url: 'https://www.google.com',
    method: 'GET',
    success: function(data) {
      // data would be sent successfully
      sendResponse(data);
    },
  },
  true);
});
```

```
    return true; // keeps the message channel open until `sendResponse` is executed
  });
```

Bien sûr, cela s'applique également à un `return` explicite du rappel `onMessage`:

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
  if (request.action == 'get') {
    $.ajax({
      url: 'https://www.google.com',
      method: 'GET',
      success: function(data) {
        // data would be sent successfully
        sendResponse(data);
      },
    });

    return true; // keeps the message channel open until `sendResponse` is executed
  }

  // do something synchronous, use sendResponse

  // normal exit closes the message channel
});
```

Lire Message passant en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/2185/message-passant>

---

# Chapitre 6: Pages de fond

## Exemples

### Déclarer la page de fond dans le manifeste

Il existe deux manières d'enregistrer une page d'arrière-plan dans le manifeste d'extension.

#### 1. La propriété `scripts`

Dans le cas habituel, une page d'arrière-plan ne nécessite aucun balisage HTML. Nous pouvons enregistrer ces types de pages de fond en utilisant la propriété `scripts`.

Dans ce cas, une page d'arrière-plan sera générée par le système d'extension qui inclut chacun des fichiers répertoriés dans la propriété `scripts`.

```
{
  ...
  "background": {
    "scripts": ["background1.js", "background2.js"],
    "persistent": true
  },
  ...
}
```

#### 2. La propriété de la `page`

Dans certains cas, nous pouvons vouloir spécifier HTML dans la page d'arrière-plan, nous pouvons obtenir cela en utilisant la propriété de la `page`.

```
{
  ...
  "background": {
    "page": "background.html",
    "persistent": true
  },
  ...
}
```

---

`scripts` VS `page`

Il est difficile de dire lequel est le meilleur. nous pourrions utiliser la propriété `page` et avoir des éléments déclarés dans la page HTML pour une utilisation future. Nous pourrions également créer dynamiquement de tels éléments dans les scripts sans déclarer explicitement la page HTML. Tout dépend des besoins réels.

Lire Pages de fond en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/4066/pages-de-fond>

---

# Chapitre 7: Portage vers / depuis Firefox

## Remarques

Si vous utilisez une version de *Firefox* antérieure à 48, vous aurez également besoin d'une clé supplémentaire dans `manifest.json` appelée `applications`:

```
"applications": {
  "gecko": {
    "id": "borderify@example.com",
    "strict_min_version": "42.0",
    "strict_max_version": "50.*",
    "update_url": "https://example.com/updates.json"
  }
}
```

### [applications](#)

---

Remarque:

#### [Signature d'extension](#) :

Avec la sortie de Firefox 48, la signature de l'extension ne peut plus être désactivée dans les versions de la version et du canal bêta en utilisant une préférence. Comme indiqué lors de l'annonce de la signature de l'extension, nous publions des versions spécialisées qui prennent en charge cette préférence afin que les développeurs puissent continuer à tester le code généré par les versions bêta et par publication.

Statut de `WebExtensions` :

`WebExtensions` sont actuellement dans un état alpha expérimental. À partir de Firefox 46, vous pouvez publier `WebExtensions` sur les utilisateurs de Firefox, comme tout autre module complémentaire. Nous visons une première version stable dans Firefox 48.

**UPD** : *Firefox* 48 publié le 02.08.2016.

---

Liens:

[État de la prise en charge de l'API](#) - Liste des API et leur statut.

[Incompatibilités Chrome](#)

[WebExtensions](#) - API JavaScript, clés de `manifest.json`, didacticiels, etc.

## Exemples

## Portage via WebExtensions

Avant de parler du portage des extensions *Firefox* depuis / vers, il faut savoir ce que sont les `WebExtensions`.

`WebExtensions` - est une plate-forme qui représente une API pour créer des extensions *Firefox*.

Il utilise la même architecture d'extension que *Chromium*. Par conséquent, cette API est compatible à bien des égards avec l'API de *Google Chrome* et *Opera* (Opera basé sur Chrome). Dans de nombreux cas, les extensions développées pour ces navigateurs fonctionneront dans *Firefox* avec quelques modifications ou même sans elles.

MDN [recommande](#) d'utiliser `WebExtension` pour les nouvelles extensions:

À l'avenir, `WebExtensions` sera la méthode recommandée pour développer des modules complémentaires pour *Firefox*, et les autres systèmes seront obsolètes.

Compte tenu de ce qui précède, si vous souhaitez porter des extensions sur *Firefox*, vous devez savoir comment l'extension a été écrite.

Les extensions pour *Firefox* peuvent être basées sur `WebExtension`, `Add-on SDK` ou `XUL`.

---

## Extensions compatibles basées sur WebExtension

Lorsque vous utilisez `WebExtension`, vous `WebExtension` parcourir la liste des [incompatibilités](#), car certaines fonctions sont prises en charge totalement ou partiellement, c'est-à-dire que vous `WebExtension manifest.json`.

Il permet également d'utiliser le même [espace de noms](#) :

À l'heure actuelle, toutes les API sont accessibles via l'espace de noms `chrome`. \*. Lorsque nous commençons à ajouter nos propres API, nous nous attendons à les ajouter au navigateur. \* Namespace. Les développeurs pourront utiliser la détection des fonctionnalités pour déterminer si une API est disponible dans le navigateur. \*.

## Une extension simple qui peut fonctionner dans Firefox et Google Chrome

`manifest.json` :

```
{
  "manifest_version": 2,
  "name": "StackMirror",
```

```

"version": "1.0",

"description": "Mirror reflection of StackOverflow sites",

"icons": {
  "48": "icon/myIcon-48.png"
},

"page_action": {
  "default_icon": "icon/myIcon-48.png"
},

"background": {
  "scripts" : ["js/background/script.js"],
  "persistent": false
},

"permissions": ["tabs", "*://*.stackoverflow.com/*"]
}

```

**script de background :**

```

function startScript(tabId, changeInfo, tab) {

  if (tab.url.indexOf("stackoverflow.com") > -1) {

    chrome.tabs.executeScript(tabId,

      {code: 'document.body.style.transform = "scaleX(-1)";'}, function () {

        if (!chrome.runtime.lastError) {

          chrome.pageAction.show(tabId);

        }

      });

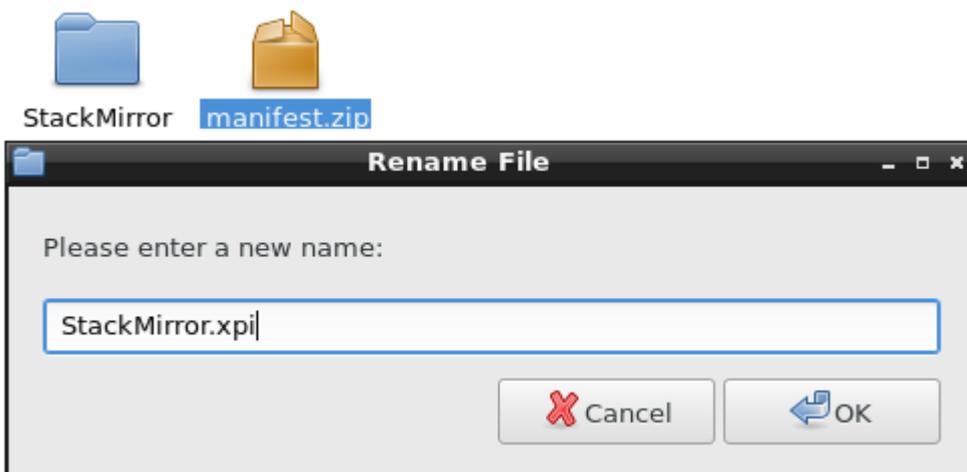
  }

}

chrome.tabs.onUpdated.addListener(startScript);

```

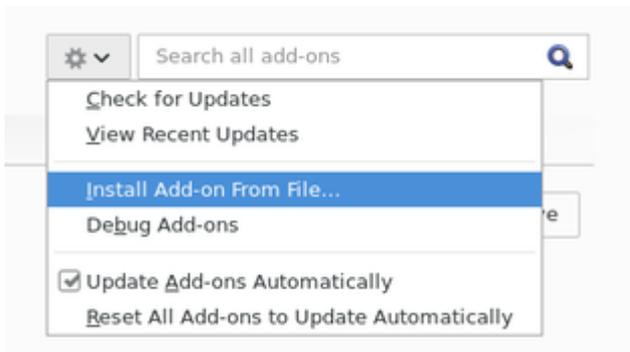
Projet pack en fichier zip standard, mais avec les extensions .xpi .



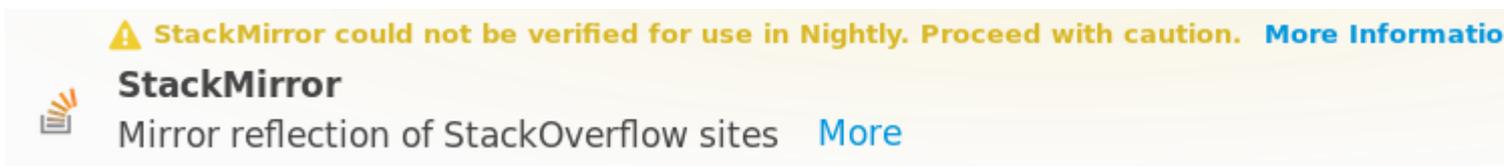
Ensuite, vous devez charger l'extension dans *Firefox* .

Ouvrez la page `about:addons` , accessible via **Menu > Add-ons** .

Cliquez sur le bouton **Outils pour tous les modules complémentaires** .



Lorsque l'extension est chargée, la page à `about:addons` ressemblera à ceci:



Les instructions sur le chargement de l'extension dans Google Chrome se trouvent dans une autre rubrique - [Prise en main de Chrome Extensions](#) .

Le résultat de l'opération d'extension sera le même dans les deux navigateurs ( *Firefox / Google Chrome* ):



## Si l'add-on actuel est basé sur le SDK complémentaire ou XUL

Lorsque l'extension en cours de portage est basée sur un `Add-on SDK WebExtensions` , il est `WebExtensions` parcourir le tableau de comparaison des `Add-on SDK WebExtensions => WebExtensions` , car ces technologies présentent des fonctionnalités similaires, mais leur implémentation est

différente. Chaque section du tableau décrit l'équivalent du `Add-on SDK WebExtension` pour `WebExtension`.

### Comparaison avec le SDK complémentaire

Une approche similaire et pour les extensions XUL.

### Comparaison avec les extensions XUL / XPCOM

Lire Portage vers / depuis Firefox en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/5731/portage-vers---depuis-firefox>

---

# Chapitre 8: Scripts de contenu

## Remarques

---

## Documentation officielle

- [Scripts de contenu](#)
- [Politique de sécurité du contenu](#) > [Scripts de contenu](#)

## Exemples

### Déclaration de scripts de contenu dans le manifeste

Les scripts de contenu peuvent être déclarés dans `manifest.json` pour être toujours injectés dans des pages correspondant à un ensemble de [modèles d'URL](#) .

---

## Exemple minimal

```
"content_scripts" : [  
  {  
    "js": ["content.js"],  
    "css": ["content.css"]  
    "matches": ["http://example.com/*"]  
  }  
]
```

Cette entrée de manifeste demande à Chrome d'injecter un script de contenu `content.js` , ainsi que le fichier CSS `content.css` , après toute navigation vers une page correspondant au [modèle de correspondance](#) `http://example.com/*`

Les clés `js` et `css` sont facultatives: vous ne pouvez en avoir qu'une ou les deux en fonction de vos besoins.

`content_scripts` clé `content_scripts` est un tableau et vous pouvez déclarer plusieurs définitions de script de contenu:

```
"content_scripts" : [  
  {  
    "js": ["content.js"],  
    "matches": ["http://*.example.com/*"]  
  },  
  {  
    "js": ["something_else.js"],  
    "matches": ["http://*.example.org/*"]  
  }  
]
```

Notez que `js` et les `matches` sont des tableaux, même si vous n'avez qu'une entrée.

Plus d'options sont disponibles dans la [documentation officielle](#) et d'autres exemples.

---

## Note importante

Les scripts de contenu déclarés dans le manifeste **ne seront injectés sur les nouvelles navigations qu'après le chargement de l'extension**. Ils ne seront pas injectés dans les onglets existants. Cela s'applique également aux rechargements d'extension pendant le développement et aux mises à jour d'extension après la publication.

Si vous devez vous assurer que les onglets actuellement ouverts sont couverts, envisagez également d'injecter par programmation au démarrage.

### Injection de scripts de contenu à partir d'une page d'extension

Si, au lieu d'avoir toujours un script de contenu injecté en fonction de l'URL, vous souhaitez contrôler directement l'injection d'un script de contenu, vous pouvez utiliser l'[injection programmatique](#).

## Exemple minimal

- JavaScript

```
chrome.tabs.executeScript({file: "content.js"});
```

- CSS

```
chrome.tabs.insertCSS({file: "content.css"});
```

Appelé depuis une page d'extension (par exemple, arrière-plan ou popup), et en supposant que vous ayez la permission d'injecter, ceci exécutera `content.js` ou insérera `content.css` tant que script de contenu dans le cadre supérieur de l'onglet actuel.

## Code en ligne

Vous pouvez exécuter du code en ligne au lieu d'un fichier en tant que script de contenu:

```
var code = "console.log('This code will execute as a content script');";
chrome.tabs.executeScript({code: code});
```

## Choisir l'onglet

Vous pouvez fournir un ID de tabulation (généralement à partir d'autres méthodes `chrome.tabs` ou de messagerie) à exécuter dans un onglet autre que celui actuellement actif.

```
chrome.tabs.executeScript({
  tabId: tabId,
  file: "content.js"
});
```

D'autres options sont disponibles dans la [documentation chrome.tabs.executeScript\(\)](#) et dans d'autres exemples.

---

## Autorisations

L'utilisation de `chrome.tabs.executeScript()` ne nécessite pas de permission `"tabs"` , mais nécessite [des autorisations d'hôte](#) pour l'URL de la page.

---

## Vérification des erreurs

Si l'injection du script échoue, on peut l'attraper dans le rappel facultatif:

```
chrome.tabs.executeScript({file: "content.js"}, function() {
  if(chrome.runtime.lastError) {
    console.error("Script injection failed: " + chrome.runtime.lastError.message);
  }
});
```

### Plusieurs scripts de contenu dans le manifeste

---

## Mêmes conditions, plusieurs scripts

Si vous devez injecter plusieurs fichiers avec toutes les autres conditions identiques, par exemple pour inclure une bibliothèque, vous pouvez tous les répertorier dans le tableau `"js"` :

```
"content_scripts" : [
  {
    "js": ["library.js", "content.js"],
    "matches": ["http://*.example.com/*"]
  }
]
```

**L'ordre compte:** `library.js` sera exécuté avant `content.js` .

---

## Mêmes scripts, plusieurs sites

Si vous devez injecter les mêmes fichiers dans plusieurs sites, vous pouvez fournir plusieurs modèles de correspondance:

```
"matches": ["http://example.com/*", "http://example.org/*"]
```

Si vous avez besoin d'injecter dans pratiquement chaque page, vous pouvez utiliser des modèles de correspondance larges tels que `*://*/*` (correspond à chaque page HTTP (S)) ou `<all_urls>` (correspond à chaque [page prise en charge](#)).

---

## Différents scripts ou différents sites

`content_scripts` est également un tableau, on peut donc définir plusieurs blocs de script de contenu:

```
"content_scripts" : [
  {
    "js": ["content.js"],
    "matches": ["http://*.example.com/*"]
  },
  {
    "js": ["something_else.js"],
    "matches": ["http://*.example.org/*"]
  }
]
```

Lire Scripts de contenu en ligne: <https://riptutorial.com/fr/google-chrome-extension/topic/2850/scripts-de-contenu>

# Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec google-chrome-extension	<a href="#">Aminadav</a> , <a href="#">Community</a> , <a href="#">Deliaz</a> , <a href="#">Haibara Ai</a> , <a href="#">ScientiaEtVeritas</a> , <a href="#">Xan</a>
2	Débogage des extensions Chrome	<a href="#">Marc Guiselin</a>
3	Intégration de l'outil de développement	<a href="#">Aminadav</a> , <a href="#">Paul Sweatte</a> , <a href="#">Xan</a>
4	manifest.json	<a href="#">Haibara Ai</a> , <a href="#">Xan</a>
5	Message passant	<a href="#">Haibara Ai</a> , <a href="#">wOxxOm</a> , <a href="#">Xan</a>
6	Pages de fond	<a href="#">Haibara Ai</a> , <a href="#">Noam Hacker</a>
7	Portage vers / depuis Firefox	<a href="#">Deliaz</a> , <a href="#">UserName</a>
8	Scripts de contenu	<a href="#">Haibara Ai</a> , <a href="#">Xan</a>