Opera Branding Guidelines - (OPG)
Editor : Chanuim
2020.04.03

extension -> 확장프로그램으로 번역했습니다.

필수 요소 학습

오페라 확장프로그램 작동 방법을 깔끔하게 적어놓은 개요가 필요하면 오페라 확장프로그램 구조(overview of the Opera extensions architecture)를 살펴보는 것부터 시작하면 됩니다.

오페라 확장프로그램을 어떻게 만드는지 배우고 싶으시면 첨 만들어보는 오페라 확장프로그램(making your first Opera extension) 아티클을 보시면 됩니다.
이 아티클을 읽음으로써 간단한 확장프로그램을 만드는 방법에 익숙해 질 수 있읍니다.

확장프로그램 어떻게 돌아가는 지에 대한 기본 지식을 가지고 있으면 이렇게 하면 됨
ㄴ 확장프로그램(사이드바, 북마크 참조 등등) 개발과 API 사용법에 관한 것들은 다양한 튜토리얼과 자세한 설명들이 적혀 있으니 그거 보면 됨.

테스트, 디버그 그리고 훑어보기

오페라 확장프로그램 테스팅과 디버깅 가이드(guide to testing and debugging Opera extensions) 읽고 문제정의를 하고 쉽게 해결 하셈

마참내 "그것"을 세상에 보여주는 거야!

확장프로그램 스토어에 관해 쓰인 확장프로그램 제출 가이드라인(guidelines on submitting extensions - Publishing Guidelines) 읽고 제출 과정을 쉽고 간단하게 만드셈.

너가 따라야 되는 합격기준을 써넣은 체크리스트(handy checklist of the acceptance criteria for you to follow)도 제공했음.

I - Architecture Overview

Introduction

오페라 확장프로그램의 구조와 기술적 세부사항들을 자세히 살펴보도록 핮아.

The CRX Format

오페라는 확장프로그램에 관한 CRX(Chromium eXtension)파일 포맷을 지원한다.

확장프로그램의 모든 파일과 폴더는 집파일로 패키징되고 .crx로 파일 확장자가 변경된다.
CRX포맷은 크로미움 확장프로그램의 주요 부분을 지원하고 오페라 전용 API도 지원함.

크로미움 프로젝트의 API는 chrome.*을 사용해 호출할 수 있지만, 오페라만의 특정 기능(사이드바 액션같은 경우)은 opr.* object로 사용해서 호출 할 수 있다.
(크로미움 이야기가 나온 이유는 오페라는 크로미움을 이용해서 만든 브라우저이기 때문)

확장프로그램이 오페라가 지원하는 chrome.* API를 사용하는 한, 어떤 확장프로그램(크로미움의 CRX포맷)이든 오페라가 실행하기때문에 위의 문장은 중요함

왼쪽 사이드바의 API 문서 섹션(API DOCS)은 현재 오페라가 지원하고 있는 것들을 제공해주고 있음.

확장프로그램 코드보고 싶으면 .zip포맷으로 바꾸고, 압축해제 프로그램으로 zip파일 압축해제 하셈ㅇㅇ

Types of extensions

현재 오페라는 네가지 유형의 확장프로그램이 있음.

1. 브라우저 액션(과 페이지 액션)을 포함시킨 확장프로그램

browser window로 UI element를 집어넣는 것으로 브라우저 액션과 페이지 액션을 사용할 수 있음.
브라우저 액션은 주소창 오른편에 UI element를 배치하는 것으로 사용됨.
UI element를 주소창 안쪽에 배치하는 페이지 액션과는 다름.

페이지 액션은 특정 기준에 맞는 일부 페이지들 혹은 그냥 한 페이지에만 특정 UI element를 넣어서 사용함

모든 페이지에 UI element 띄우고 싶으면 브라우저 액션 쓰셈.
사용할 수 있는 UI element는 버튼, 뱃지, 팝업임.
요놈들을 어케 만드는지는 버튼, 뱃지, 팝업 만들기(creating buttons, badges and popups) 읽으셈

브라우저 액션을 사용한 툴바는 한번에 최대 6개의 확장프로그램이 설치가능이며 페이지 액션은 4개까지만 설치가능임.(무슨 소리지?)

2. Context Menu extensions

이름이 의미하는 바와 같이, 페이지에 맞게끔 컨텍스트 메뉴가 확장 함.
궁금하면 컨텍스트 메뉴 확장프로그램 만들기(how to create context menu extensions) 아티클 읽으셈 ㅎㅅㅎ

3. Extensions with no UI

아무런 UI 컴포넌트가 없는 확장프로그램도 만들 수 있음!
물론 님이 오페라 이전 버전 아님 Greasemonkey 스크립트에 삽입된 스크립트를 알고 있다면 ㅇㅇ

이에 관한 예는 이거임
키보드 인풋을 기다림 그리고 유저가 키보드 숏컷을 입력하면 어떤 행동(새 텝에서 특정 페이지 열기)을 수행함

이런 확장프로그램은 이 글 담 부분에서 다룰 스크립트의 일부분으로 쓰일거임

II - Different parts of an extension

The Extension manifest

모든 확장프로그램은 반-드시 manifest 파일이 들어있읍니다.
manifest 파일은 확장프로그램의 이름, 작성자, 확장프로그램이 어떤 API에 접근하길 원하는지(permissions 필드)에 관한 중요한 정보 등등을 제공해줍니다.

manifest파일이 정확히 정의되지 않으면 확장프로그램은 실행조차 되지 않을거에오
다른 또 한가지 중요한 건 developer 필드인데 이는 이름을 적을 수 있쓰어오~

manifest에 대해 더 알고 싶으면 read the API doc을 갈 것

The Background Process

특정 작업 혹은 특정 상태를 유지하기 위해 조정을 해야되는 경우, 백그라운드에서 프로세스를 실행해야 됩니다.

이를 위해 두 가지 대안이 있읍니다. — Background 페이지 or Event 페이지

HTML 페이지를 사용해 script태그에 자바스크립트를 꼴박할 수 있지만, .js파일 만들고
manifest파일에서 그거 참조하는게 더 나음. 브라우저는 자동으로 그에 맞는 페이지를 생성해줄거임.

ex) manifest file code
“background” : {
	“scripts” : [“backgorund.js”]
}

manifest file에서 event 페이지를 명시하기 위해서, persistent 필드를 false로 설정해야 될 필요가 있읍니다.
ex) manifest file code
“background” : {
	“scripts” : [“backgorund.js”],
	“persistent” : false
}

Difference of Background Page and Event Page.
(본문에 없는 소제목이지만 이해를 위해 붙였음)

Background 페이지(혹은 background 스크립트)는 유저 인터페이스에 필쑤입니다.
브라우저에 UI 아이템으로 추가하기 위한 코드들은 여기에 define되어 있어야 합니다.
또한, state의 변화를 알아 차리고 그에 맞춰 UI를 변경해야 되는 책임이 있습니다.

Event 페이지는 필요할 때에만 로드 된다는 점에서, background 페이지와 정확히 같습니다.
이벤트 페이지가 로드 되지 않았다는 말은 시스템 메모리와 리소스가 전혀 사용되지 않았다는 뜻으로 따라서, 더 나은 퍼포먼스를 보입니다.

확장프로그램을 만든다면 가능하다면 event 페이지를 사용하는걸 권장합니다.

이벤트 페이지가 로드 될 때?

1) 확장프로그램이 설치됐거나 다시 시작하거나 시작하거나 혹은 새로운 버전으로 업데이트 됐거나
2) listening 되고 있는 이벤트 페이지에다가 이벤트를 보낼 때
    a) listening 되고 있다는 말은 특정 이벤트를 설정했다는 뜻
3) 확장프로그램의 다른 부분(e.g. popup)이 이벤트 페이지를 호출했을 때
4) 확장프로그램의 다른 부분이 메시지를 보낼때(runtime.sendMessage()혹은 long-lived connections을 사용할 때?.)

주 차이점은 event 페이지는 이벤트만을 다루도록 만들어졌다는 것이다.
그래서 eventpage.js에 이벤트 리스너들을 register시켜야 됨
그리고 브라우저는 저장 방식을 최적화 할 것이고, 런타임때 이 이벤트들을 실행함.

이벤트 리스너에 감싸이지 않은 내용들(리스너안에 포함되지 않은 코드들)은 로드될 때 다뤄지며,
이벤트 리스너안에서 어떤 방법으로든 변수와 함수를 참조할 때만 엔진에 의해 유지 될거임.
(life-cycle에 대한 얘기)

결론은, 브라우저에 의해 자원 소비 감소로 인한 성능이 좋아지기 때문에 되도록 이면 이벤트 페이지를 사용해라.

The Content Script

웹 페이지 자체를 변경하고 싶으면, content script를 사용해야 됨.
content script는 웹 페이지의 dom에 접근 할 수 있지만, 변수와 함수에 대한 접근권은 dom자체에만 국한 된다.

이는 다른 content스크립트에도 적용되며 background스크립트에도 적용됨 그리고 API접근도 그렇다.

하지만 message passing을 사용해 확장프로그램의 다른 부분들(background script, popup)과 communicate할 수 있음.
(background 스크립트에 있는 함수를 불러다가 content 스크립트에서 호스트 페이지의 dom을 포함한 특정 작업을 할 수 있음)

더 자세한걸 보고 싶다면 content scripts에 관한 아티클을 읽어보세요.

The Popup Page

때로는 확장프로그램을 클릭하면 팝업이 나타날 수 있읍니다. HTML페이지에 의해 define되고 manifest에 명시할 필요가 있읍니다.
buttons, badges and popups 아티클을 눌러서 더 보세요ㅎㅎ

The Options Page

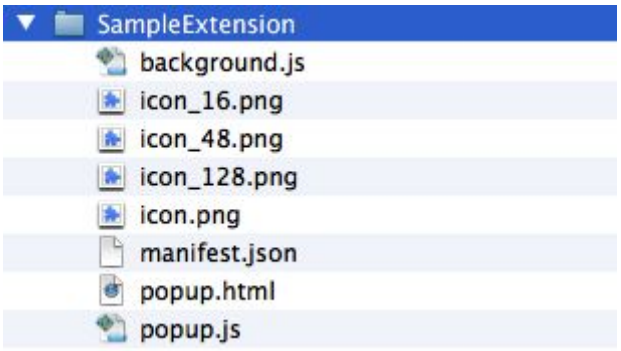
사용자 설정을 저장할 공간이 필요할 때, options page를 만들어서 해결할 수 있습니다.
options 페이지를 define했다면, 유저가 들어갈 수 있는 확장프로그램 관리 페이지가 링크로 주어집니다.
options 페이지도 마찬가지로 manifest에 선언할 필요가 있습니다.
확장프로그램의 사용자 설정을 저장하기 위해 Web Storage API를 정의한 localStorage를 사용할 수 있습니다.

Icons and other files

확장프로그램은 아이콘이 필요함(128x128은 설치중에 그리고 확장프로그램의 페이지에 사용 됨. 48x48은 확장프로그램의 관리페이지, 16x16은 확장프로그램의 페이지의 favicon, 19x19는 브라우저 혹은 페이지 액션으로 아이콘을 집어 넣어야 될 때 사용됨)

아이콘 외에도 이미지, 폰트, 팝업과 옵션 페이지를 꾸미기 위한 css와 js파일등이 필요 할 수도 있습니다.

이런 것들은 확장프로그램 패키지의 모든 공간에 있을 수 있음.
(경로가 딱히 정해지지 않아도 상관 없다는 뜻)



위의 스크린샷은 전형적인 확장프로그램의 폴더 구조를 나타낸 것이다.
더욱 구조화 시킨다면 media폴더를 만들고 이미지, 폰트, 다른 미디어 파일을 넣을 수 있고, css폴더 만들어서 css를, 스크립트 폴더를 만들어서 js를 만들 수 있다.

상대 경로로 확장프로그램에 있는 파일을 참조 할 수 있음.
<img src="media/myimage.png">
물론 절대경로로도 할 수 있지만 귀찮아 진다.
chrome-extension://<extensionID>/<pathToFile>

III - Permissions and privileges

Permissions are necessary

각 확장프로그램에는 브라우저 API 사용이 허용되고, 확장프로그램이 실행 될 수 있는 도메인 집합에 대한 접속을 관리하는 manifest 파일이 제공됩니다.

Separation of privileges

content 스크립트와 확장프로그램의 나머지부분은 역할과 특권이 분리 되어 있습니다.
content 스크립트는 웹페이지를 수-정할 수 있지만, UI레이어를 수정할 특권은 없음.
하지만 확장프로그램의 나머지부분(except of content)은 그걸 할 수 있지만 웹페이지를 수정할 수 없습니다.

Content scripts works in isolated worlds

content 스크립트는 페이지의 dom에 접근함으로써 웹 페이지를 수정할 수 있지만, 웹 페이지가 갖고 있는 변수와 함수에는 접근할 수 없습니다.
background 프로세스가 정의한 변수와 함수에 접근 할 수 없으며 반대의 경우도 이와 마찬가지임.
(message passing을 통해 둘이 전달 할 수 있긴함)
이 말은 Content스크립트는 확장프로그램 API에 접근하지 못 한다는 말이 됨. - background와 event페이지만 확장프로그램 api에 접근할 수 있으며 각각의 content스크립트는 지들만의 세상에서 산다. -
확장프로그램에 있는 다른 content script의 변수와 함수에 접근하지 못한다는 뜻 엌ㅋㅋㅋㅋ

Content security policy

콘텐츠 보안 정책(content security policy)은 확장프로그램의 manifest에 정의 되어 있음

ex)
“content_security_policy” : “[WRITE YOUR POLICY STRING HERE]”

이에 관한 문법과 csp에 넣을 값들을 알고 싶으면 specification에 가라우.
기본적으로 확장프로그램에 대한 policy string은 script-src ‘self’; object-src ‘self’임
그리고, 확장프로그램의 manifest에 정책을 정의하지 않으면 이 정책이 가정됨.
이 정책에 따르면 다음과 같은 사항에 유의해야 됨.

1) eval()과 관련된 함수들은 허용되지 않음
    a) eval()과 같은 것들은 cross-site scripting attack에 사용되므로 비활성화 합니더
        i) eval()
        ii) setTimeout()
        iii) setInterval()
        iv) new Function(String)
    b) unsafe-eval 문자열을 넣음으로써 eval()과 연관된 함수를 사용할 수 있음.
    c) 하지만 오피셜에 따르면 사용하지 말라고 강력하게 권고하고 있음.
2) 인라인 자바스크립트는 실행되지 않음

a) 인라인 js는 cross-site scripting attack의 공격 매개체로 사용될 수 있으므로 인라인 js는 비활성화 됨
b) script 블록과 같은 인라인 이벤트 핸들러는 허용되지 않는다 이말이야.
    i) <a onclick=”something()”>
c) 페이지 혹은 팝업을 갖고 있고, 상호작용을 할 일이 생겼다면, js파일에 적고 html페이지에서 그걸 참조하세요!
d) unsafe-inline 적어도 아무런 일도 일어나지 않으니 무모한 저항은 그만둬라.

3) 스크립트와 리소스가 네트워크가 아닌 확장프로그램의 파일로만 로드 됨
a) 확장프로그램 패키지에 있는 스크립트와 오브젝트만 로드할 수 있읍니다 ㅎㅅㅎ
b) 아 물론 화이트리스트를 통해서 로컬서버는 제외할 수 있음 ㅎㅎ
c) 화이트리스트에 크롬-익스텐션, 크롬-익스텐션-리소스도 추가 가능
d) ajax 통신하는 것에는 영향을 미치지 않으니 안심하라구~