Chrome packaged apps

Codelab at Google I/O 2013

Find this guide online at http://goo.gl/UHCS8

Instructor:
Renato Mangini, Developer Programs Engineer

Teaching Assistants:
Eiji Kitamura, Developer Advocate
Joe Marini, Developer Advocate
Mike Tsao, Software Engineer
Pete LePage, Developer Advocate

Introduction

Welcome to the Chrome packaged apps codelab! This is a self-paced codelab where you will exercise the basic concepts of a Chrome packaged app and also learn some of its many APIs. Most concepts covered in this tutorial are described in details in the platform's official documentation. In each Codelab step you will find links to the specific APIs covered there.

In the first chapter you will learn the basic building blocks required for a Chrome packaged application and how to run and debug it.

In the second chapter we will bootstrap from a well known open source web application, the ToDoMVC, and learn how to adapt it to run as a Chrome packaged app, removing unsupported features like localStorage and achieving CSP compliance.

From the third chapter onwards, we will add features to the ToDoMVC app using the Chrome packaged apps exclusive APIs.

At the end, you should have build an offline-enabled, installable and feature-rich version of the ToDoMVC app. Along the way, you should learn:

- The building blocks and the basic development flow for the Chrome platform
- How to package existing web apps and run it in the Chrome platform
- chrome.storage.local storage, an asynchronous local object store
- alarms and notifications APIs
- How to use the webview tag to show external, unrestricted web content
- How to load resources, like images, from external sources
- Using the extended FileSystem API to write to a file in the native filesystem

Prerequisites

This codelab assumes you are familiar with fundamental web programming. You should know the basics of HTML and CSS, and you should be familiar with JavaScript.

You should have Chrome Dev installed. Access chrome://version in your Chrome and check **if the "Google Chrome" is version 28** or later. If it is not, please follow the instructions in the link
https://www.google.com/intl/en/chrome/browser/index.html?extra=devchannel#eula

FOR PIXEL USERS: if you want to use your shiny new Pixel Chromebook (you should!), just follow the instructions at the beginning of Step 1.

If you really want to use Chrome 27, please be aware that the notifications on step 3 will not work.

How to use this material

Start fresh in a new directory.
Each step builds on top of the previous.
There is a recommended time for each step. If you get past this recommended time and decide to move over, every step has a "cheat" link where you can grab the reference code from the previous step.

Download the reference code for all steps from:

https://github.com/mangini/io13-codelab/archive/master.zip

Install and play with the app at http://goo.gl/qNNUX

Step 1 - Create and run a minimum app

Objectives: understand the development flow
Recommended time to complete this step: 10 minutes

Besides the code specific to your app, a Chrome Packaged App requires two files:

- A manifest[1], where you specify the meta information of your app
- An event page[2], also called background page, where you usually register listeners for specific app events, like a launch listener that opens the app's window

Want to use your shiny new Chromebook Pixel? Prepare it by following one of the flows below:

- **Option 1, RECOMMENDED for this Codelab**:
    a. Install a simple text editor app like this[3]
    b. Create a new folder under the Downloads directory
    c. Optional but **highly** recommended: switch to the newer, unstable ChromeOS. If you don't follow this step, you will have the stable version (ChromeOS 26) and some APIs covered in this Codelab may not work.
        i. Make sure you are logged in as the owner of the Chromebook (the first user to sign in)
        ii. Go to chrome://help/
        iii. Click in More info...
        iv. select "Dev - unstable" in the "Channel" listbox. If you don't see this listbox, you are not logged in as the Chromebook owner.
        v. wait until it updates. A restart is required at the end. If you want to restore the stable version after the Codelab, you will need to follow these procedures.
- **Option 2, for ADVANCED users only!** Follow the steps b and c from the Option 1. And then switch to Developer mode and use any Linux editor. **Switching into Developer Mode does a lot of stuff (erasing your stateful partition, disabling some of the checks that make sure you're running official code, etc).** If you still want to go this route after so many warnings, switch to developer mode, install crouton and use VIM or any Linux text editor.

Open your favorite code/text editor and create the following files in an empty directory:

manifest.json:

[1] http://developer.chrome.com/trunk/apps/manifest.html
[2] http://developer.chrome.com/trunk/apps/app_lifecycle.html#eventpage
[3] http://goo.gl/MjR7R

```
{
 "manifest_version": 2,
 "name": "I/O Codelab",
 "version": "1",
 "permissions": [],
 "app": {
   "background": {
     "scripts": ["background.js"]
   }
 },
 "minimum_chrome_version": "28"
}
```

background.js:

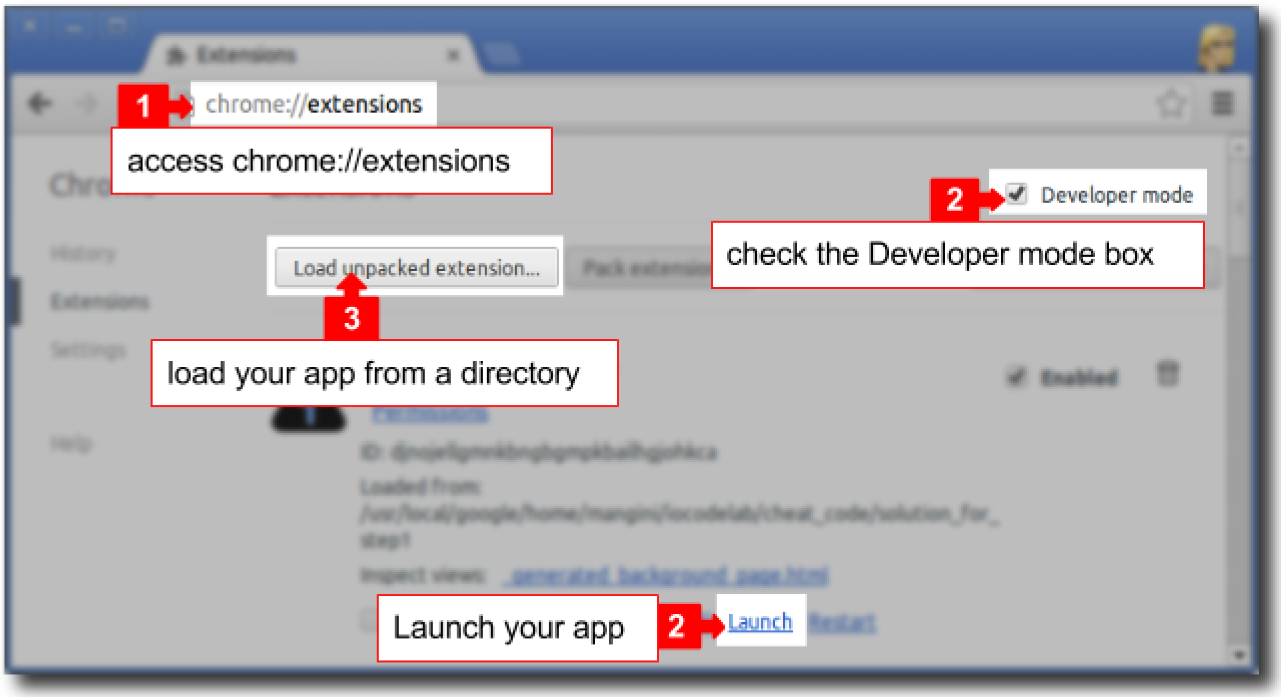
```
chrome.app.runtime.onLaunched.addListener(function() {
 chrome.app.window.create('index.html', {
   id: 'main',
   bounds: { width: 620, height: 500 }
 });
});
```

index.html:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
</head>
<body>
   <h1>Hello, let's code!</h1>
</body>
</html>
```

Congratulations, you've just created a new Chrome packaged app. If you've copied from the cheats directory, make sure you also copy the icon_128.png because it is referred at manifest.json.

Now you can run it:

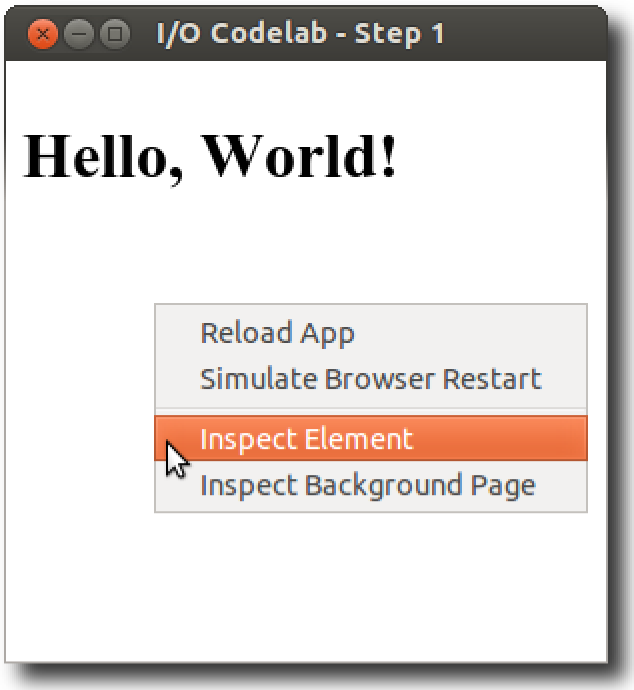

1
chrome://extensions
access chrome://extensions
2
Developer mode
check the Developer mode box
Load unpacked extension...
3
load your app from a directory
Launch your app
2
Launch
Restart

How to debug a Chrome packaged app

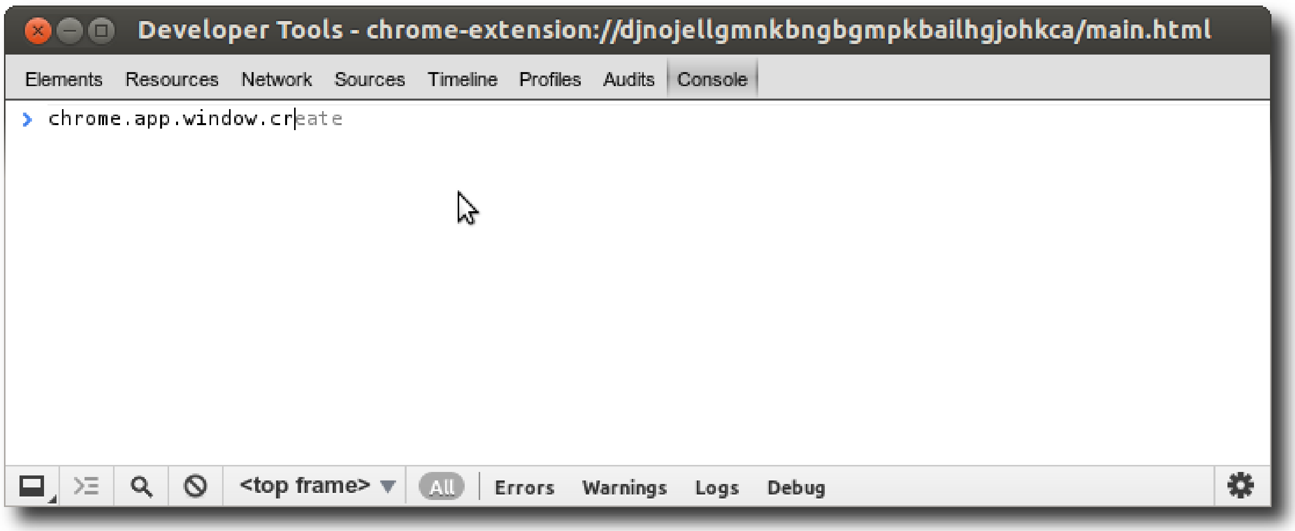
If you are familiar with the Chrome Developer Tools, you will like to know that you can use the Developer tools to inspect, debug, audit and test your app just like you do it on a regular web page.

The usual development cycle is:

- Write a piece of code
- Reload the app (right click, Reload App)
- Test
- Check DevTools console for any error
- Rinse and repeat.



The Developer Tools console has access to the same APIs available to your app. This way, you can easily test an API call before adding it to your code:



```
> chrome.app.window.create
```

Step 2 - Import and adapt ToDoMVC app

Want to start fresh from here? Get previous step's code in `solution_for_step1` subfolder!

Objectives:

- Learn how to adapt an existing application for the Chrome apps platform
- Learn the chrome.storage.local API[4]
- Prepare the basis for the next codelab steps.

Recommended time to complete this step: 20 minutes

We will start by importing an existing regular web app into our project, and as a starting point, we will use a common benchmark app, the ToDoMVC[5]:

1. We've included a version of the ToDoMVC app in the reference code zip. Copy all content, including subfolders, from the `todomvc` folder in the zip to your application folder. You should now have the following file structure in your application folder:

   `manifest.json` (from step 1)
   `background.js` (from step 1)
   `icon_128.png` (optional, from step 1)
   `index.html` (from todomvc)
   `bower.json` (from todomvc)
   `bower_components/` (from todomvc)
   `js/` (from todomvc)

2. Reload your app now. You should see the basic UI, but nothing else will work. If you open the console (right-click, Inspect Element, Console), there is an error like:

```
Refused to execute inline script because it violates the following Content Security Policy directive: "default-src 'self' chrome-extension-resource:". Note that 'script-src' was not explicitly set, so 'default-src' is used as a fallback.
 index.html:42
```

CSP Compliance

1. Let's fix this error by making the app CSP compliant[6]. One of the most common CSP non-compliances is caused by inline JavaScript, like event handlers as DOM attributes (`<button onclick=''>`) and `<script>` tags with content inside the HTML. The solution is simple: just move the inline content to a new file.
   a. edit index.html and move the inline JavaScript to a new file `js/bootstrap.js`:

[4] http://developer.chrome.com/trunk/apps/storage.html
[5] If you know the ToDoMVC web app (http://todomvc.com), we have copied its vanilla JavaScript version to be used as a starting point.
[6] http://developer.chrome.com/trunk/apps/app_csp.html

```
      ~~<script>~~
            ~~// Bootstrap app data~~
            ~~window.app = {};~~
      ~~</script>~~
      <script src="js/bootstrap.js"></script>
```

   b. create a `js/bootstrap.js` file with the following contents:

```
      // Bootstrap app data
      window.app = {};
```

3. Reload your app. Not working yet, right? But now if you open Console, the previous error should be gone, but there is another one:

```
Uncaught window.localStorage is not available in packaged apps. Use
chrome.storage.local instead. platformApp:16
```

4. This one takes more steps to fix...

Converting from localStorage to chrome.storage.local

LocalStorage is not supported in Chrome packaged apps. Why? LocalStorage is synchronous, and synchronous access to blocking resources (I/O) in a single threaded runtime in general is not a great idea. If your storage has high latency, your app might become unresponsive.

Chrome packaged apps has an asynchronous equivalent that can also store objects directly, avoiding the sometimes costly object->string->object serialization process.

To fix this in our app, we will need to convert localStorage to chrome.storage.local. There are many steps, but they are all small changes to the API calls or fixes on the ToDoMVC async support.

NOTE: changing all the places were localStorage is used in the ToDoMVC is time-consuming and error-prone, although required and important for your learning. To maximize your fun at this Codelab, we highly suggest that you:

   a. look at the code changes described below. Check if you understand them and ask teaching assistants any questions if may have
   b. **copy files from cheat_code/solution_for_step_2 and proceed to the next Codelab Step**. We've kept all the steps below for learning purposes.

1. In manifest.json, request the "storage" permission:

```
...
  "permissions": ["storage"],
...
```

2. In store.js, fix the constructor:

```
function Store(name, callback) {
  var data;
  var dbName;

  callback = callback || function () {};

  dbName = this._dbName = name;

~~  if (!localStorage[dbName]) {~~
~~    data = {~~
~~      todos: []~~
~~    };~~
~~    localStorage[dbName] = JSON.stringify(data);~~
~~  }~~

~~  callback.call(this, JSON.parse(localStorage[dbName]));~~

  chrome.storage.local.get(dbName, function(storage) {
    if ( dbName in storage ) {
      callback.call(this, storage[dbName].todos);
    } else {
      storage = {};
      storage[dbName] = { todos: [] };
      chrome.storage.local.set( storage, function() {
        callback.call(this, storage[dbName].todos);
      }.bind(this));
    }
  }.bind(this));

}
```

3. Fix the find method:

```
Store.prototype.find = function (query, callback) {
  if (!callback) {
    return;
  }

~~  var todos = JSON.parse(localStorage[this._dbName]).todos;~~

~~  callback.call(this, todos.filter(function (todo) {~~
~~    for (var q in query) {~~
~~      return query[q] === todo[q];~~
~~    }~~
~~  }));~~

  chrome.storage.local.get(this._dbName, function(storage) {
    var todos = storage[this._dbName].todos.filter(
      function (todo) {
        for (var q in query) {
          return query[q] === todo[q];
        }
      });
    callback.call(this, todos);
  }.bind(this));
};
```

4. Fix the findAll method:

```
Store.prototype.findAll = function (callback) {
  callback = callback || function () {};
~~  callback.call(this, JSON.parse(localStorage[this._dbName]).todos);~~

  chrome.storage.local.get(this._dbName, function(storage) {
    callback.call(this, storage[this._dbName].todos);
  }.bind(this));
};
```

5. The save method presents a new challenge: since it depends on two async operations (get and set) that operate on the whole monolithic JSON storage every time, any batch update on more than one ToDo, like "mark all ToDos as completed", will result in a data hazard known as Read-After-Write[7]. There are several ways to fix it, but we will use the chance to slightly refactor it by taking an array of ToDo id's to be updated at once. Notice that this issue wouldn't happen if we were using a more appropriate data storage, like IndexedDB. But we are trying to minimize the conversion effort:

```
Store.prototype.save = function (id, updateData, callback) {

  chrome.storage.local.get(this._dbName, function(storage) {

~~    var data = JSON.parse(localStorage[this._dbName]);~~
    var data = storage[this._dbName];

    var todos = data.todos;

    callback = callback || function () {};

```

[7] http://en.wikipedia.org/wiki/Hazard_(computer_architecture)#Read_After_Write_.28RAW.29

```
    // If an ID was actually given, find the item and update each
property
    if ( typeof id !== 'object'  || Array.isArray(id) ) {
      var ids = [].concat( id );
      ids.forEach(function(id) {
        for (var i = 0; i < todos.length; i++) {
          if (todos[i].id == id) {
            for (var x in updateData) {
              todos[i][x] = updateData[x];
            }
          }
        }
      });

~~      localStorage[this._dbName] = JSON.stringify(data);~~
~~      callback.call(this,~~
~~JSON.parse(localStorage[this._dbName]).todos);~~
      chrome.storage.local.set(storage, function() {
        chrome.storage.local.get(this._dbName, function(storage) {
          callback.call(this, storage[this._dbName].todos);
        }.bind(this));
      }.bind(this));

    } else {
      callback = updateData;

      updateData = id;

      // Generate an ID
      updateData.id = new Date().getTime();

      todos.push(updateData);
~~      localStorage[this._dbName] = JSON.stringify(data);~~
~~      callback.call(this, [updateData]);~~

      chrome.storage.local.set(storage, function() {
        callback.call(this, [updateData]);
      }.bind(this));

    }
  }.bind(this));
};
```

6. We also need to change the client of the save method, so it can include all IDs in one call.

a. Update method toggleComplete in controller.js:

```
Controller.prototype.toggleComplete = function (ids, checkbox,
silent) {
  var completed = checkbox.checked ? 1 : 0;

  this.model.update(ids, { completed: completed }, function () {
    if ( ids.constructor != Array ) {
      ids = [ ids ];
    }

    ids.forEach( function(id) {

      var listItem = $$('[data-id="' + id + '"]');

      if (!listItem) {
        return;
      }

      listItem.className = completed ? 'completed' : '';

      // In case it was toggled from an event and not by clicking
the checkbox
      listItem.querySelector('input').checked = completed;
    });

    if (!silent) {
      this._filter();
    }
  }.bind(this));
};
```

b. Update method toggleAll in controller.js:

```
Controller.prototype.toggleAll = function (e) {
  var completed = e.target.checked ? 1 : 0;
  var query = 0;

  if (completed === 0) {
    query = 1;
  }

  this.model.read({ completed: query }, function (data) {
    var ids = [];
    data.forEach(function (item) {
      ids.push(item.id);
      ~~this.toggleComplete(item.id, e.target, true);~~
    }.bind(this));
    this.toggleComplete(ids, e.target, false);
  }.bind(this));

  this._filter();
};
```

7. Let's now fix two minor bugs in the ToDoMVC code that only show up when actual async storage is used:
   a. In controller.js, method removeItem, move the _filter call to inside the callback:

```
  Controller.prototype.removeItem = function (id) {
    this.model.remove(id, function () {
      this.$todoList.removeChild($$('[data-id="' + id + '"]'));
      this._filter();
    }.bind(this));

~~    this._filter();~~
  };
```

   b. Also in controller.js, make _updateCount method async:

```
Controller.prototype._updateCount = function () {
~~  var todos = this.model.getCount();~~
  this.model.getCount(function(todos) {

    this.$todoItemCounter.innerHTML =
this.view.itemCounter(todos.active);

    this.$clearCompleted.innerHTML =
this.view.clearCompletedButton(todos.completed);
    this.$clearCompleted.style.display = todos.completed > 0 ?
'block' : 'none';

    this.$toggleAll.checked = todos.completed === todos.total;

    this._toggleFrame(todos);
  }.bind(this));
};
```

   c. And the corresponding method getCount in model.js now needs to accept a callback:

```
Model.prototype.getCount = function (callback) {
  var todos = {
    active: 0,
    completed: 0,
    total: 0
  };

  this.storage.findAll(function (data) {
    data.each(function (todo) {
      if (todo.completed === 1) {
        todos.completed++;
      } else {
        todos.active++;
      }

      todos.total++;
    });
    if (callback) callback(todos);
  });

~~  return todos;~~
};
```

8. We are almost there. If you reload the app now, you will be able to insert new Todo's, but you won't be able to remove them. Let's now fix the remove method in store.js. The same data hazard issue mentioned in the save method is also present here, so we need to change the remove method to allow for batch operations:
   a. Fix the remove method in store.js:

```
Store.prototype.remove = function (id, callback) {
  chrome.storage.local.get(this._dbName, function(storage) {
~~    var data = JSON.parse(localStorage[this._dbName]);~~
    var data = storage[this._dbName];
    var todos = data.todos;

    var ids = [].concat(id);
    ids.forEach( function(id) {
      for (var i = 0; i < todos.length; i++) {
        if (todos[i].id == id) {
          todos.splice(i, 1);
          break;
        }
      }
    });

~~  localStorage[this._dbName] = JSON.stringify(data);~~
~~  callback.call(this, JSON.parse(localStorage[this._dbName]).todos);~~
```

```
    chrome.storage.local.set(storage, function() {
      callback.call(this, todos);
    }.bind(this));
  }.bind(this));
};
```

b. Now change the removeCompletedItems in controller.js to make it call removeItem on all ids at once:

```
Controller.prototype.removeCompletedItems = function () {
  this.model.read({ completed: 1 }, function (data) {
    var ids = [];
    data.forEach(function (item) {
      ids.push(item.id);
      ~~this.removeItem(item.id);~~
    }.bind(this));
    this.removeItem(ids);
  }.bind(this));

  this._filter();
};
```

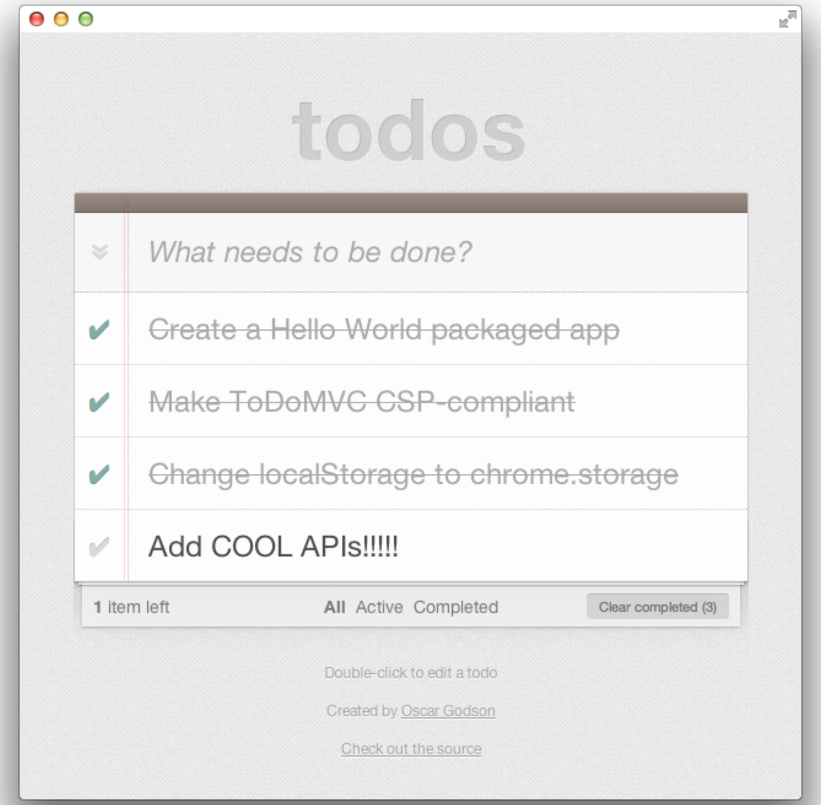
c. And finally change the remoteItem in controller.js to support removing multiple items from DOM at once:

```
Controller.prototype.removeItem = function (id) {
  this.model.remove(id, function () {
    var ids = [].concat(id);
    ids.forEach( function(id) {
      this.$todoList.removeChild($$('[data-id="' + id + '"]'));
    }.bind(this));
    this._filter();
  }.bind(this));
};
```

9. You are done. Reload your app (right-click, Reload App) and enjoy!

There is another method in store.js using localStorage: drop. But since it is not being used anywhere in the project, we will leave it as an exercise for you to fix it later.

You should now have a cool working Chrome packaged version of the ToDoMVC as in the screenshot below:



Got a problem?

Remember to always have the console of Developer Tools open to see JavaScript error messages:

- Open the Developer Tools (right-click on the app's window, click on Inspect element)
- Select the Console tab
- You should see any runtime error messages there

Step 3 - Alarms and notifications: remind yourself of open To Do's

Want to start fresh from here? Get previous step's code in `solution_for_step2` subfolder!

Objectives:

- learn how to wake your app at specified intervals using alarms
- learn how to use notifications to draw user attention to something important

Recommended time to complete this step: 20 minutes

Now we will change the app to remind you if you have open To Do's, even when the app is closed. The app will use the Alarms API to set a wake up interval and, as long as Chrome is running, the alarm listener will be called at approximately the interval set.

Part 1 - Alarms:

1. In manifest.json, request the "alarms" permission:

```
...
  "permissions": ["storage", "alarms"],
...
```

2. In background.js, add a onAlarm listener that, for now, just send a log message to the console whenever there is a To Do item in the storage:

```
...
chrome.app.runtime.onLaunched.addListener(function() {
 chrome.app.window.create(index.html', {
   id: 'main',
   bounds: { width: 620, height: 500 }
 });
});

chrome.alarms.onAlarm.addListener(function( alarm ) {
 console.log("Got an alarm!", alarm);
});
```

3. In index.html, add an "Activate alarm" button and import the script we will create in the upcoming step :

```
...
  <footer id="info">
    <button id="toggleAlarm">Activate alarm</button>
    <p>Double-click to edit a todo</p>
```

```
...
  <script src="js/store.js"></script>
  <script src="js/model.js"></script>
  <script src="js/view.js"></script>
  <script src="js/controller.js"></script>
  <script src="js/app.js"></script>
  <script src="js/alarms.js"></script>
</body>
</html>
```

4. Create a new script js/alarms.js:
    - add checkAlarm, createAlarm, cancelAlarm and toggleAlarm methods:

```
(function () {
  'use strict';
   var alarmName = 'remindme';

   function checkAlarm(callback) {
     chrome.alarms.getAll(function(alarms) {

       var hasAlarm = alarms.some(function(a) {
         return a.name == alarmName;
       });

       var newLabel;
       if (hasAlarm) {
         newLabel = 'Cancel alarm';
       } else {
         newLabel = 'Activate alarm';
       }
       document.getElementById('toggleAlarm').innerText = newLabel;

       if (callback) callback(hasAlarm);
     })
   }

   function createAlarm() {
     chrome.alarms.create(alarmName, {
       delayInMinutes: 0.1, periodInMinutes: 0.1});
   }

   function cancelAlarm() {
     chrome.alarms.clear(alarmName);
   }

```

```
    function doToggleAlarm() {
      checkAlarm( function(hasAlarm) {
        if (hasAlarm) {
          cancelAlarm();
        } else {
          createAlarm();
        }
        checkAlarm();
      });
    }

   $$('#toggleAlarm').addEventListener('click', doToggleAlarm);

   checkAlarm();

})();
```

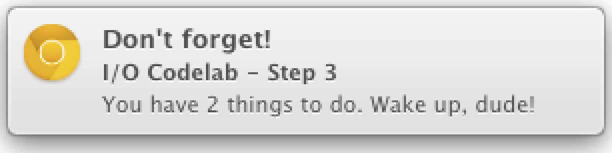
NOTES:

- Observe the parameters in `chrome.alarms.create` call. These small values (0.1 of a minute) are for testing only. In a published app, these values are not accepted and are rounded to approximately 1 minute. In your test environment, a simple warning is issued to the console - you can ignore it.
- Since the log message is being sent to the console in the event (background) page (see step 2 above), you need to inspect that background page to see the log messages:

  Open the Developer Tools (right-click on the app's window, click on **Inspect Background page**, select the Console tab). Whenever you have the alarm activated, you should see log messages being printed in the console every time the alarm "rings".

Part 2 - Notifications:

Now let's change the alarm notification to something the user can easily notice. For that purpose, we will use the chrome.notifications API. We will show a desktop notification like the one below and, when the user clicks on the notification, we will open or raise the To Do window to the top.


Don't forget!
I/O Codelab - Step 3
You have 2 things to do. Wake up, dude!

1. In manifest.json, request the "notifications" permission:

```
...
  "permissions": ["storage", "alarms", "notifications"],
...
```

2. In background.js:
    - move the chrome.app.window.create call to a method so we can reuse it:

```
...
function launch() {
 chrome.app.window.create('index.html', {
   id: 'main',
   bounds: { width: 620, height: 500 }
 });
}

~~chrome.app.runtime.onLaunched.addListener(function() {~~
~~  chrome.app.window.create('index.html', {~~
~~    id: 'main',~~
~~    bounds: { width: 620, height: 500 }~~
~~  });~~
~~});~~

chrome.app.runtime.onLaunched.addListener(launch);
...
```

    - create a showNotification method (notice that the sample code below refers to an `icon_128.png`. If you want your notification to have an icon, remember to also copy it from the cheat code or create your own icon):

```
...
var dbName = 'todos-vanillajs';

function showNotification(storedData) {

  var openTodos = 0;

  if ( storedData[dbName].todos ) {
    storedData[dbName].todos.forEach(function(todo) {
      if ( !todo.completed ) {
        openTodos++;
      }
    });
  }

  if (openTodos>0) {
    // Now create the notification
    chrome.notifications.create('reminder', {
        type: 'basic',
        iconUrl: 'icon_128.png',
        title: 'Don\'t forget!',
        message: 'You have '+openTodos+
                 ' things to do. Wake up, dude!'
     }, function(notificationId) {})
  }
}

// When the user clicks on the notification,
// we will open the To Do list and dismiss the notification
chrome.notifications.onClicked.addListener(
  function( notificationId ) {
    launch();
    chrome.notifications.clear(notificationId, function() {});
  }
);
...
```

- change the onAlarm listener to, instead of simply logging a new alarm to console, get stored data and call the showNotification:

```
...
chrome.alarms.onAlarm.addListener(function( alarm ) {
 ~~console.log("Got an alarm!", alarm);~~
 chrome.storage.local.get(dbName, showNotification);
});
...
```

Now reload your app and spend a few minutes playing with it. You should notice that:

- Even when you close the app window, the alarms will keep coming
- If you close all Chrome windows, alarms won't arrive (on platforms other than ChromeOS)
- When your app is closed, if you click on the notification the window will open

Got a problem?

If notifications are not showing up, check if your Chrome version is 28. The chrome.notifications API were introduced in Chrome 28, so you might want to change your code to use the standard Web notifications API instead. We will leave that as a challenge for you. The W3C specifications are here[8]

If you have Chrome 28 and notifications still don't show up, check error messages in console on both the main window (right click -> Inspect element) and the background page (right click -> Inspect background page).

Step 4 - Parse URLs and open then in a Webview

Want to start fresh from here? Get previous step's code in `solution_for_step3` subfolder!

Objectives:

- learn that it is possible to load and show most external content inside your app using a webview tag in a secure and sandboxed way

Recommended time to complete this step: 10 minutes

Some applications need to present external web content directly to the user, but keep him inside the application experience. For example, a news aggregator might want to embed the news from external sites with all the formatting, images and behavior of the original site. For these and other usages, Chrome packaged apps have a custom HTML tag called webview[9]. It is a very powerful component, but its simplest usage is actually pretty straightforward, as you may see below.

We will now change our sample to search for URLs in the To Do content and, when found, add a side link. The link, when clicked, will open a new app window (not a browser tab) with a webview presenting the content.

1. In manifest.json, request the "webview" permission:

```
...
  "permissions": ["storage", "alarms", "notifications", "webview"],
...
```

2. Create a new file, webview.html with a simple <webview> tag:

```
<!DOCTYPE html>
```

[8] http://www.w3.org/TR/notifications/
[9] http://developer.chrome.com/trunk/apps/webview_tag.html

```
<html>
<head>
   <meta charset="utf-8">
</head>
<body>
  <webview style="width: 100%; height: 100%;"></webview>
</body>
</html>
```

3. In controller.js:
   - add a method to parse URLs from the To Do content. Whenever a URL is found, an anchor replaces it:

```
  Controller.prototype._parseForURLs = function (text) {
    var re = /(https?:\/\/[^\s"<>,]+)/g;
    return text.replace(re, '<a href="$1" data-src="$1">$1</a>');
  };
```

   - add a method to open a new window with webview.html and set a URL in webview.src:

```
Controller.prototype._doShowUrl = function(e) {
  // only applies to elements with data-src attributes
  if (!e.target.hasAttribute('data-src')) {
    return;
  }
  e.preventDefault();

  var url = e.target.getAttribute('data-src');
  chrome.app.window.create(
   'webview.html',
   {hidden: true},   // only show window when webview is configured
   function(appWin) {
     appWin.contentWindow.addEventListener('DOMContentLoaded',
       function(e) {
         // when window is loaded, set webview source
         var webview = appWin.contentWindow.
              document.querySelector('webview');
         webview.src = url;
         // now we can show it:
         appWin.show();
       }
     );
   });
};
```

○ Parse for links whenever items are shown:

```
/**
 * An event to fire on load. Will get all items and display them in
the
 * todo-list
 */
Controller.prototype.showAll = function () {
  this.model.read(function (data) {
    this.$todoList.innerHTML =
       this._parseForURLs(this.view.show(data));
  }.bind(this));
};

/**
 * Renders all active tasks
 */
Controller.prototype.showActive = function () {
  this.model.read({ completed: 0 }, function (data) {
    this.$todoList.innerHTML =
       this._parseForURLs(this.view.show(data));
  }.bind(this));
};

/**
 * Renders all completed tasks
 */
Controller.prototype.showCompleted = function () {
  this.model.read({ completed: 1 }, function (data) {
    this.$todoList.innerHTML =
        this._parseForURLs(this.view.show(data));
  }.bind(this));
};
```

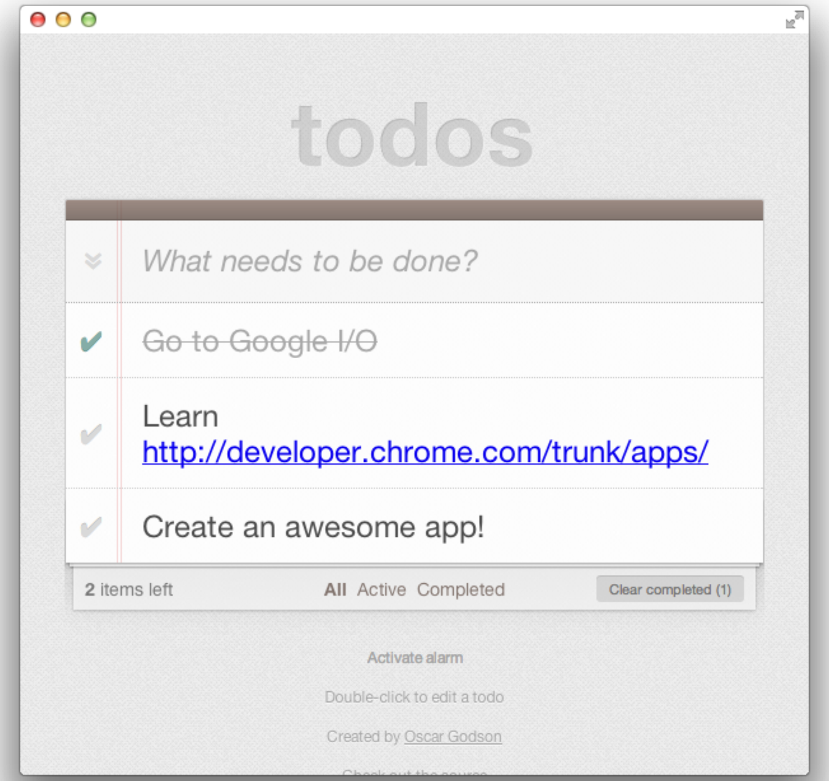
○ Parse for links in the editItem. Also, fix the code so that it uses the innerText of the input element instead of its innerHTML:

```
  Controller.prototype.editItem = function (id, label) {
    ...

    var onSaveHandler = function () {
      ...
        // Instead of re-rendering the whole view just update
        // this piece of it
        ~~label.innerHTML = value;~~
        label.innerHTML = this._parseForURLs(value);
    ...

    // Get the innerHTML of the label instead of requesting the data
from the
    // ORM. If this were a real DB this would save a lot of time and
would avoid
    // a spinner gif.
~~    input.value = label.innerHTML;~~
    input.value = label.innerText;
    ...
```

- Finally, add a click listener in the Controller constructor to call the doShowUrl method when user clicks in a link:

```
  function Controller(model, view) {
    this.model = model;
    this.view = view;

    this.ENTER_KEY = 13;
    this.ESCAPE_KEY = 27;

    this.$main = $$('#main');
    this.$toggleAll = $$('#toggle-all');
    this.$todoList = $$('#todo-list');
    this.$todoItemCounter = $$('#todo-count');
    this.$clearCompleted = $$('#clear-completed');
    this.$footer = $$('#footer');

    this.router = new Router();
    this.router.init();

    this.$todoList.addEventListener('click', this._doShowUrl);
    ...
```

Now, if you reload your app, you should have something like:


todos

What needs to be done?

~~Go to Google I/O~~

Learn http://developer.chrome.com/trunk/apps/

Create an awesome app!

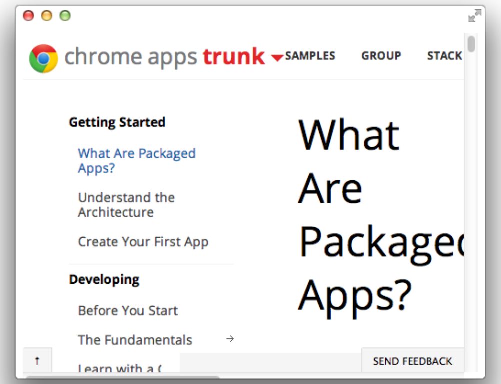
2 items left All Active Completed Clear completed (1)

Activate alarm

Double-click to edit a todo

Created by Oscar Godson

And after clicking on the link:



What Are Packaged Apps?

Note: a webview is a sandboxed process. You can only interact with it using its API. The embedder app (your app) cannot easily have direct access to the webview DOM, for example.

Advanced: If you are ahead of time, check the webview documentation and play a little bit with its methods, showing a small status message or indicator while the webview is loading.

Step 5 - Add images from the web

Want to start fresh from here? Get previous step's code in `solution_for_step4` subfolder!

Objectives:

- learn how to load resources from outside your app and add them to the DOM through XHR and ObjectURLs

Recommended time to complete this step: 20 minutes

Chrome packaged apps platform force your app to be fully compliant with Content Security Policies. This includes not being able to directly load DOM resources, like images, fonts and CSS's from outside of your app. If you want to show an external image in your app, you need to request it via XHR, transform it into a Blob and create an ObjectURL. This ObjectURL can then be added to the DOM, because it refers to an in-memory item in the context of the app.

Let's change our app to look for image URLs in the To Do content. If the URL looks like an image (ends with .png, .jpg, .svg or .gif), we will download the image and show it on the side of the anchor as a thumbnail.

1. In manifest.json, request the "<all_urls>" permission. In a Chrome packaged app you can make XMLHttpRequest calls to any URL, as long as you whitelist its domain in the manifest. Instead of listing specific domains, we will ask permission to make requests to "<all_urls>" because we don't know beforehand what image URL the user of our app will type in the To Do content:

```
...
  "permissions": ["storage", "alarms",
                  "notifications", "webview", "<all_urls>"],
...
```

2. In js/controller.js:
    a. Add a method to create ObjectURLs from a Blob. ObjectURLs hold memory and you need to revoke when you no longer need them, so add also a clear method:

```
  Controller.prototype._clearObjectURL = function() {
    if (this.objectURLs) {
      this.objectURLs.forEach(function(objURL) {
        URL.revokeObjectURL(objURL);
      });
      this.objectURLs = null;
    }
  };
  Controller.prototype._createObjectURL = function(blob) {
    var objURL = URL.createObjectURL(blob);
    this.objectURLs = this.objectURLs || [];
    this.objectURLs.push(objURL);
    return objURL;
  };
```

b. Add a method to execute a XMLHttpRequest on a URL, create an ObjectURL from the XHR's response and add an <img> element with this ObjectURL to the DOM:

```
Controller.prototype._requestRemoteImageAndAppend =
  function(imageUrl, element) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', imageUrl);
    xhr.responseType = 'blob';
    xhr.onload = function() {
      var img = document.createElement('img');
      img.setAttribute('data-src', imageUrl);
      img.className = 'icon';
      var objURL = this._createObjectURL(xhr.response);
      img.setAttribute('src', objURL);
      element.appendChild(img);
    }.bind(this);
    xhr.send();
  };
```

c. Now add a method that finds all links not yet processed and check them. For each URL that looks like an image, execute _requestRemoteImageAndAppend:

```
Controller.prototype._parseForImageURLs = function () {
  // remove old blobs to avoid memory leak:
  this._clearObjectURL();

  var links = this.$todoList.
            querySelectorAll('a[data-src]:not(.thumbnail)');
  var re = /\.(png|jpg|jpeg|svg|gif)$/;

  for (var i = 0; i<links.length; i++) {
    var url = links[i].getAttribute('data-src');
    if (re.test(url)) {
      links[i].classList.add('thumbnail');
      this._requestRemoteImageAndAppend(url, links[i]);
    }
  }
};
```

d. And call it appropriately in showAll, showActive, showCompleted and editItem:

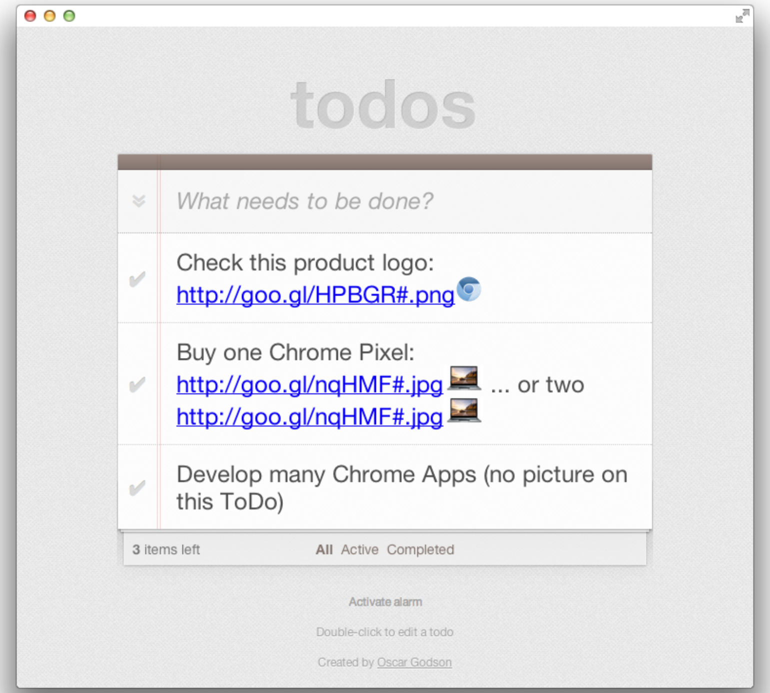
```
Controller.prototype.showAll = function () {
  this.model.read(function (data) {
    this.$todoList.innerHTML =
      this._parseForURLs(this.view.show(data));
    this._parseForImageURLs();
  }.bind(this));
};

/**
 * Renders all active tasks
 */
Controller.prototype.showActive = function () {
  this.model.read({ completed: 0 }, function (data) {
    this.$todoList.innerHTML =
      this._parseForURLs(this.view.show(data));
    this._parseForImageURLs();
  }.bind(this));
};

/**
 * Renders all completed tasks
 */
Controller.prototype.showCompleted = function () {
  this.model.read({ completed: 1 }, function (data) {
    this.$todoList.innerHTML =
      this._parseForURLs(this.view.show(data));
    this._parseForImageURLs();
  }.bind(this));
};

...

Controller.prototype.editItem = function (id, label) {
  var li =  label;

  // This finds the <label>'s parent <li>
  while (li.nodeName !== 'LI') {
    li = li.parentNode;
  }
  var onSaveHandler = function () {
    var value = input.value.trim();
    var discarding = input.dataset.discard;

    if (value.length && !discarding) {
      this.model.update(id, { title: input.value });

      // Instead of re-rendering the whole view just update
      // this piece of it
      label.innerHTML = this._parseForURLs(value);
      this._parseForImageURLs();
...
```

3. Finally, in bower_components/todomvc-common/base.css, add a CSS rule to limit the size of the image:

```
.thumbnail img[data-src] {
  max-width: 100px;
  max-height: 28px;
}
```

Now reload your app, go to Google Image Search, find some image URLs and add them in ToDo contents. Some examples:
http://goo.gl/nqHMF#.jpg
http://goo.gl/HPBGR#.png

And this is how it should look:


todos

What needs to be done?

Check this product logo: http://goo.gl/HPBGR#.png

Buy one Chrome Pixel: http://goo.gl/nqHMF#.jpg ... or two http://goo.gl/nqHMF#.jpg

Develop many Chrome Apps (no picture on this ToDo)

3 items left

Tip: For real-world situations, when you need to control offline cache and dozens of simultaneous resource downloads, we have created a helper library[10] to handle some common use cases.

[10] https://github.com/GoogleChrome/apps-resource-loader#readme

Step 6 - Export ToDo's to the filesystem

Want to start fresh from here? Get previous step's code in `solution_for_step5` subfolder!

Objectives:

- Learn how to get a reference to a file in the external filesystem and write to this file during the lifetime of the app using the FileSystem API

Recommended time to complete this step: 20 minutes

In this step, we will add an export button to the app. When clicked, the current To Do items will be saved to a text file selected by the user. If the file exists, it will be replaced. Otherwise, a new file will be created. Observe that the user only needs to select the file once during the lifetime of the object representing the file entry. In our example, we tied it to the app window - so as long as the user keeps the window open, the JavaScript code can write to the selected file without any user interaction.

1. In manifest.json, request the `{fileSystem: [ "write" ] }` permission. Notice that the syntax of this permission is more complex than the others, as we will want to not only access the external fileSystem, but also write to it:

```
...
  "permissions": ["storage", "alarms", "notifications", "webview",
                  "<all_urls>", { "fileSystem": ["write"] } ],
...
```

2. In index.html, add an "Export to disk" button and a div where we will show a status message. Also, load the script we will create next:

```
...
  <footer id="info">
    <button id="toggleAlarm">Activate alarm</button>
    <button id="exportToDisk">Export to disk</button>
    <div id="status"></div>
    <p>Double-click to edit a todo</p>
    <p>Created by <a href="http://twitter.com/oscargodson">Oscar
Godson</a></p>
  </footer>
  <script src="bower_components/todomvc-common/base.js"></script>
  <script
src="bower_components/director/build/director.js"></script>
  <script src="js/bootstrap.js"></script>
  <script src="js/helpers.js"></script>
  <script src="js/store.js"></script>
  <script src="js/model.js"></script>
  <script src="js/view.js"></script>
  <script src="js/controller.js"></script>
  <script src="js/app.js"></script>
  <script src="js/alarms.js"></script>
  <script src="js/export.js"></script>
</body>
</html>
```

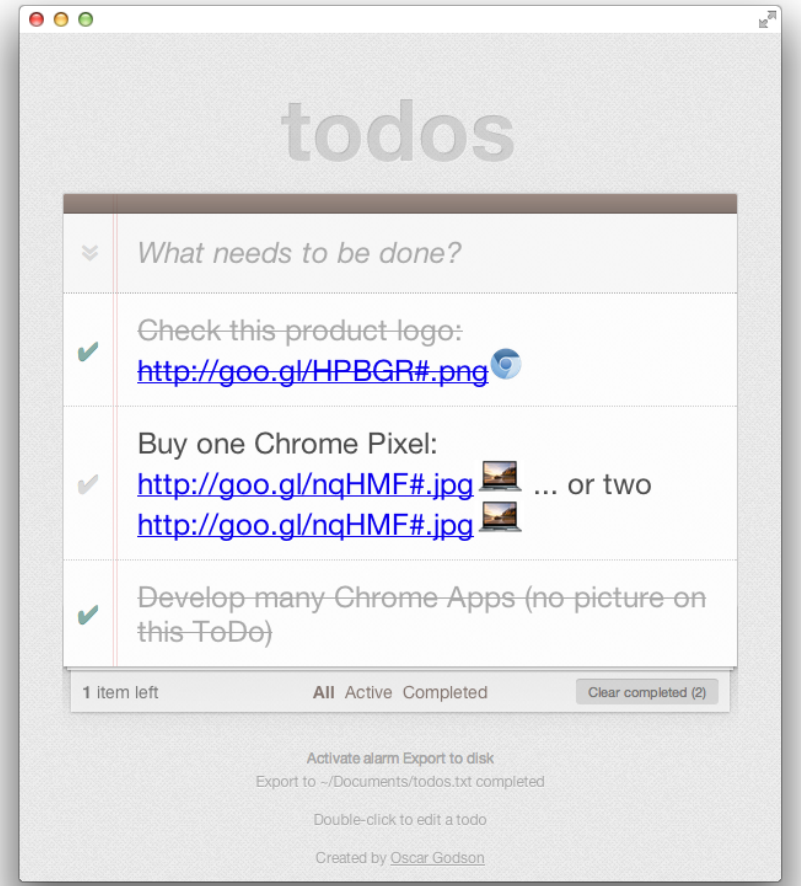
3. Create a new JavaScript, js/export.js, with the following content:
    - a method getTodosAsText that reads ToDos from chrome.storage.local and generate a textual representation of them;
    - a method exportToFileEntry that, given a FileEntry, will save the To Do's as text to that file;
    - a method doExportToDisk that executes the exportToFileEntry method added above if we already have a saved FileEntry or ask the user to choose one instead;
    - a click listener on the "Export to disk" button

```
(function() {

  var dbName = 'todos-vanillajs';

  var savedFileEntry, fileDisplayPath;

  function getTodosAsText(callback) {
    chrome.storage.local.get(dbName, function(storedData) {
      var text = '';

      if ( storedData[dbName].todos ) {
        storedData[dbName].todos.forEach(function(todo) {
            text += '- ';
            if ( todo.completed ) {
              text += '[DONE] ';
            }
            text += todo.title;
            text += '\n';
          }, '');
      }

      callback(text);

    }.bind(this));
  }

  // Given a FileEntry,
  function exportToFileEntry(fileEntry) {
    savedFileEntry = fileEntry;

    var status = document.getElementById('status');

    // Use this to get a pretty name appropriate for displaying
    chrome.fileSystem.getDisplayPath(fileEntry, function(path) {
      fileDisplayPath = path;
      status.innerText = 'Exporting to '+path;
    });

    getTodosAsText( function(contents) {

      fileEntry.createWriter(function(fileWriter) {

        fileWriter.onwriteend = function(e) {
          status.innerText = 'Export to '+
               fileDisplayPath+' completed';
        };

        fileWriter.onerror = function(e) {
          status.innerText = 'Export failed: '+e.toString();
        };

        var blob = new Blob([contents]);
        fileWriter.write(blob);

        // You need to explicitly set the file size to truncate
        // any content that might was there before
        fileWriter.truncate(blob.size);

      });
    });

  }

  function doExportToDisk() {

    if (savedFileEntry) {

      exportToFileEntry(savedFileEntry);

    } else {

      chrome.fileSystem.chooseEntry( {
        type: 'saveFile',
        suggestedName: 'todos.txt',
        accepts: [ { description: 'Text files (*.txt)',
                     extensions: ['txt']} ],
        acceptsAllTypes: true
      }, exportToFileEntry);

    }
  }

 document.getElementById('exportToDisk').
   addEventListener('click', doExportToDisk);

})()
```

Advanced: FileEntries cannot be persisted indefinitely, which means your app will need to ask the user to choose a file every time the app is launched. However, there is an option[11] to recover a FileEntry if your app was forced to restart (runtime crash, app update or runtime update for example). If you are ahead of time, try to play with it by saving the FileEntry ID and restoring it on app restart (tip: add a listener to the onRestarted event in the background page)

CONGRATS! If you did it all, you should have a complete ToDoMVC packaged app like the one below:

[11] http://developer.chrome.com/trunk/apps/fileSystem.html#method-getEntryId



todos

What needs to be done?

✓ ~~Check this product logo: http://goo.gl/HPBGR#.png~~

Buy one Chrome Pixel: http://goo.gl/nqHMF#.jpg ... or two http://goo.gl/nqHMF#.jpg

✓ ~~Develop many Chrome Apps (no picture on this ToDo)~~

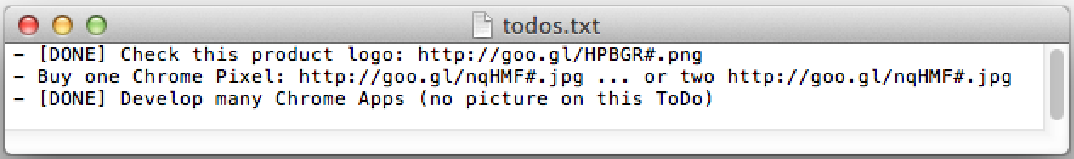
1 item left All Active Completed Clear completed (2)

Activate alarm Export to disk

Export to ~/Documents/todos.txt completed

Double-click to edit a todo

Created by Oscar Godson



todos.txt

```
- [DONE] Check this product logo: http://goo.gl/HPBGR#.png
- Buy one Chrome Pixel: http://goo.gl/nqHMF#.jpg ... or two http://goo.gl/nqHMF#.jpg
- [DONE] Develop many Chrome Apps (no picture on this ToDo)
```

Advanced bonus challenge: Add voice commands

Want to start fresh from here? Get previous step's code in `solution_for_step6` subfolder!

If you got so far and still have time, you might want to try this very advanced challenge: what about being able to add ToDos by saying them to your app? Use the HTML5 WebSpeech API[12] and follow the high-level steps below:

- request permission for "audioCapture" in the manifest.json
- start audio recognition when the user presses an activation button
- get any text after an "add note" command and before an "end note" command and save as a To Do item

There is no cheat code for this challenge, so go and hack yourself!

[12] http://www.google.com/intl/en/chrome/demos/speech.html