

# Firefox OS アプリハンズオン

## お絵描きカメラアプリを作ろう

今回のハンズオンはお絵描きアプリを拡張して、カメラ機能を付け加えた後にサーバーへお絵描きした内容をアップロードする Firefox OS アプリケーションを作ります。

M.Sakamaki

2015/02/28

# Firefox OS アプリハンズオン

お絵描きカメラアプリを作ろう

## もくじ

環境構築をしよう .....	2
FIREFOX (DEVELOPER EDITION) のインストール .....	2
FIREFOX OS シミュレーターのインストール .....	3
USB ドライバーのインストール(WINDOWSのみ) .....	4
お絵描きアプリを動かしてみよう .....	5
FIREFOX OS シミュレーターを動かしてみよう .....	5
WEB IDE でアプリを開いてみよう .....	6
アプリケーションを動かしてみよう .....	7
アプリケーションの設定を編集しよう .....	10
アプリケーション名を変えてみよう .....	10
アイコンを変えてみよう .....	11
画面の回転を固定しよう .....	12
実機に接続してみよう .....	13
実機の設定変更/確認 .....	13
実機の接続 .....	14
カメラアプリにしてみよう .....	15
カメラの入力を画面に映そう .....	15
権限を設定しよう .....	15
機能を追加しよう .....	16
写真を取ってみよう .....	20
サーバーに接続してみよう .....	22
作者と写真のタイトルを入力できるようにしよう .....	22
サーバーに情報を送ってみよう .....	23
サーバへの送信処理を書く .....	23
その他の API 紹介 .....	24
写真を取ったときに振動させてみよう .....	24
画面に触らず写真を取ってみよう .....	24
明るさでペンの色を変更してみよう .....	25
アップロードした事を通知しよう .....	26
カメラ API に置き換えてみよう .....	27

# 環境構築をしよう

## Firefox (Developer Edition) のインストール



<https://www.mozilla.org/ja/firefox/developer/>

開発者のために作られた唯一のブラウザです。作成、テスト、サイズの変更といった、開発のワークフローで必要とされる機能をすべて持っています。

各 OS のお作法に従ってインストールしてください。

初回インストールの場合、以下のようなダイアログが出ますが後で(Not now)を選択してください。  
(次回から表示しないにチェックをつけると起動毎に確認されません。)



## Firefox OS シミュレーターのインストール

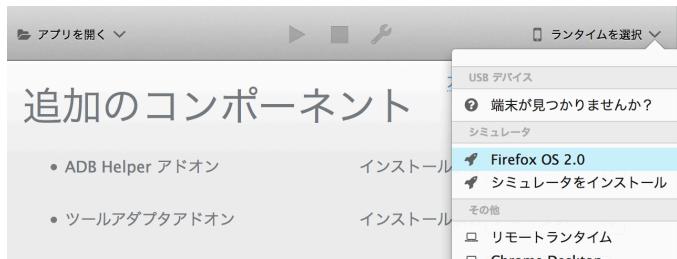
Firefox シミュレーターをインストールして行きます。  
(既にインストール済みの方は次の章に移動してください。)



次のような画面が開きます、これが Web IDE です。  
Web IDE の「ランタイムを選択」を選びます。



シミュレーターをインストール後、「ランタイムを選択」中に Firefox OS 2.0 が現れれば成功です。



## USB ドライバーのインストール(Windowsのみ)

Windowsを使用する場合は別途 USB ドライバーをインストールする必要があります。

[http://cds.w5v8t3u9.hwdcdn.net/Alcatel\\_USB\\_Driver\\_Q\\_4.0.0\\_2013\\_11\\_11\\_noinstall.zip](http://cds.w5v8t3u9.hwdcdn.net/Alcatel_USB_Driver_Q_4.0.0_2013_11_11_noinstall.zip)

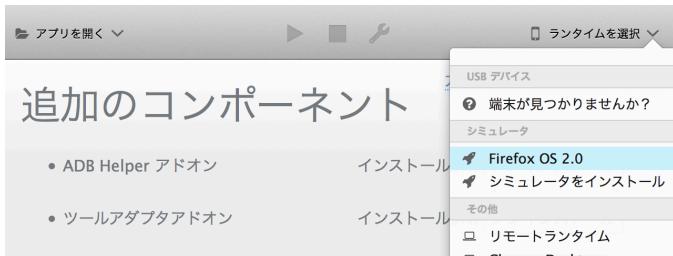
ドライバをインストールするには、ZIP ファイルを展開してできる  
the Alcatel\_USB\_Driver\_Q\_4.0.0\_2013\_11\_11\_noinstall フォルダを開き、  
DriverInstaller.exe の実行ファイルをダブルクリックします。  
この時に不明な発行元の警告が出るかもしれません。  
その場合、はいを選んで実行ファイルを起動します。

ファイルが開いたら Install ボタンをクリックしてドライバをインストールします。

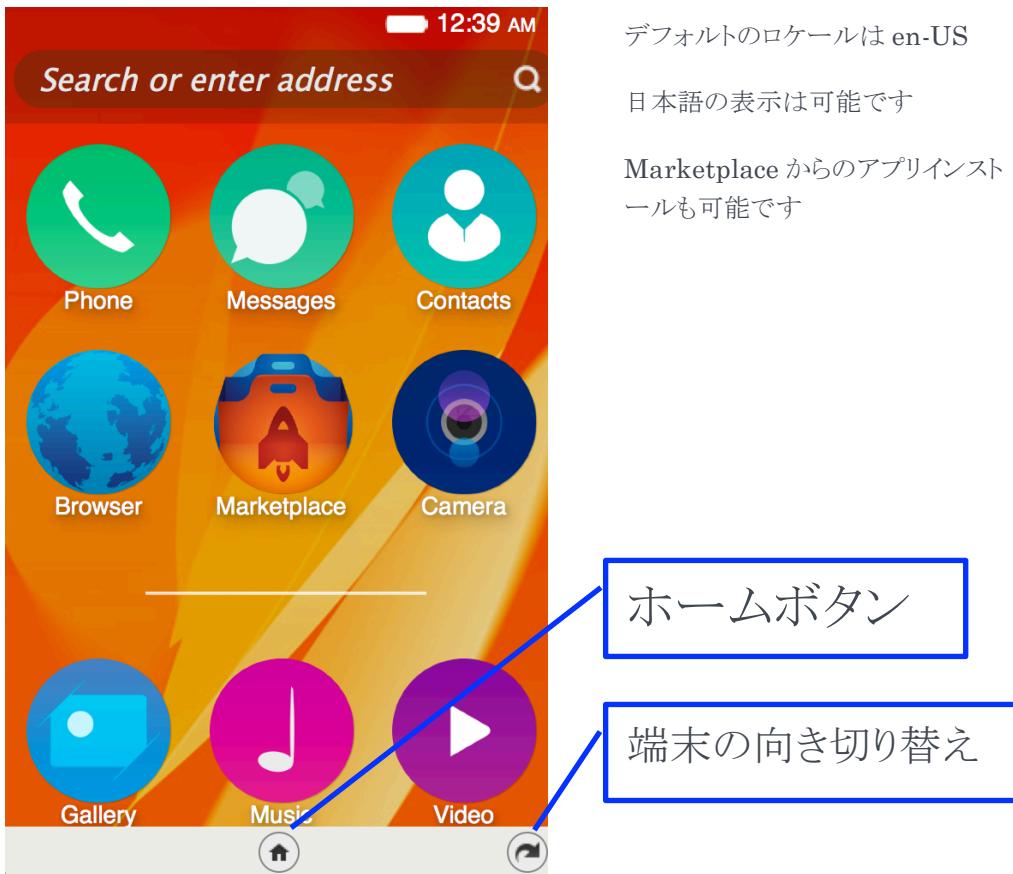
# お絵描きアプリを動かしてみよう

## Firefox OS シミュレーターを動かしてみよう

ではさっそく Firefox OS シミュレーターを起動してみましょう。  
ランタイムを選択で、Firefox OS 2.0 をクリックしてみてください。



起動すると、以下のようなウィンドウが立ち上がります。  
これが Firefox OS シミュレーターです。



## Web IDE でアプリを開いてみよう

開発の準備が完了したので、アプリケーションを WebIDE で開いてみましょう。

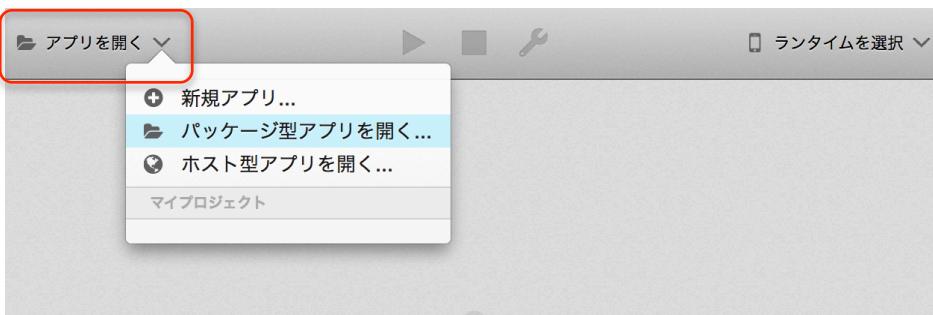
アプリケーションの雛形を以下の URL からダウンロードしてください。

URL: <https://github.com/MSakamaki/AzurePhotoShare/archive/master.zip>

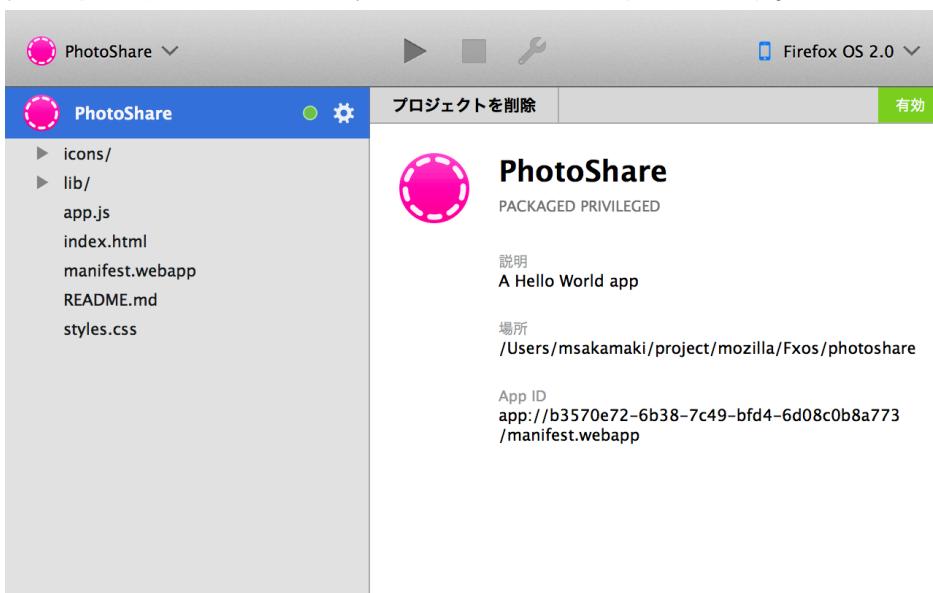
※ 接続状態が悪い方は講師、チューターに一声かけてください。USB 経由でお渡します。

次に、Web IDE の左上にある「アプリを開く」から、パッケージ型アプリを開くを選択します。

ダウンロードして解凍されたフォルダを選んでください。



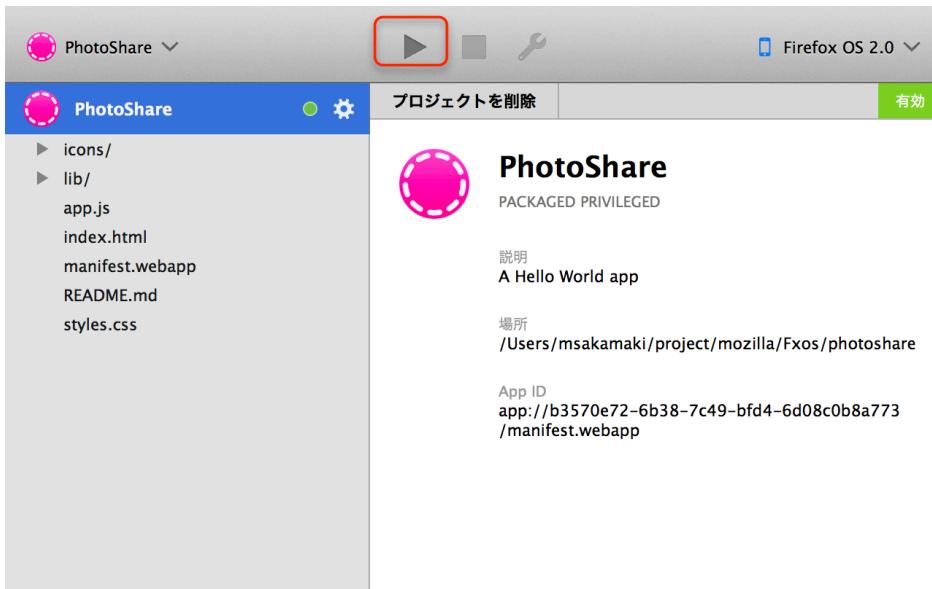
次のような画面が表示されれば、無事アプリケーションが開けています。



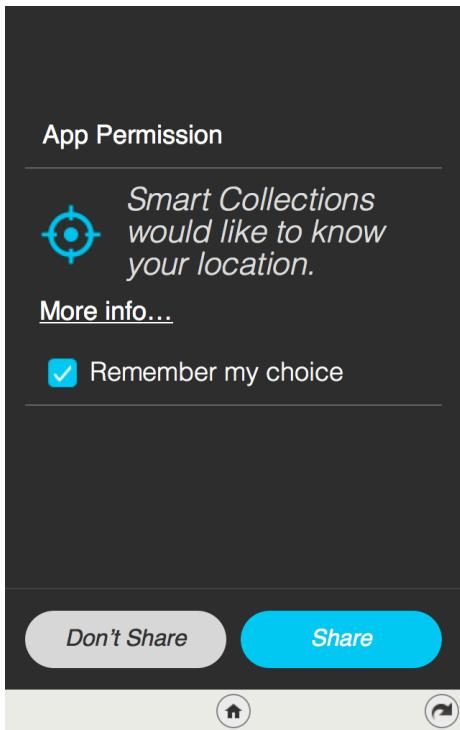
## アプリケーションを動かしてみよう

開いたアプリケーションを、Firefox OS シミュレーターで起動してみましょう。

中央上にある「▶」ボタンを押すとアプリがシミュレーターに転送されます。



初回起動時には以下ののような画面が表示されますが、「Share」を選択してください



スマートコレクションが位置情報を使用するのを許可するかどうかを確認するダイアログが表示されます。

これは初めてアプリをインストールするときにだけ表示されます。

2回目以降は表示されません。

起動すると、以下のような簡単なお絵描きアプリが表示されます。

※ 画像のパンダはサンプルです、起動時には表示されません。



左から、ペンの色を選択、操作を一つ戻す、操作を一つ進める。

## アプリケーションを更新、デバッグしてみよう



アプリケーション実行後、Web IDE の画面中央上には、3つのアイコンが表示されます。



編集したアプリケーションをシミュレータまたは実機に転送します。



実行中のアプリケーションを終了します。



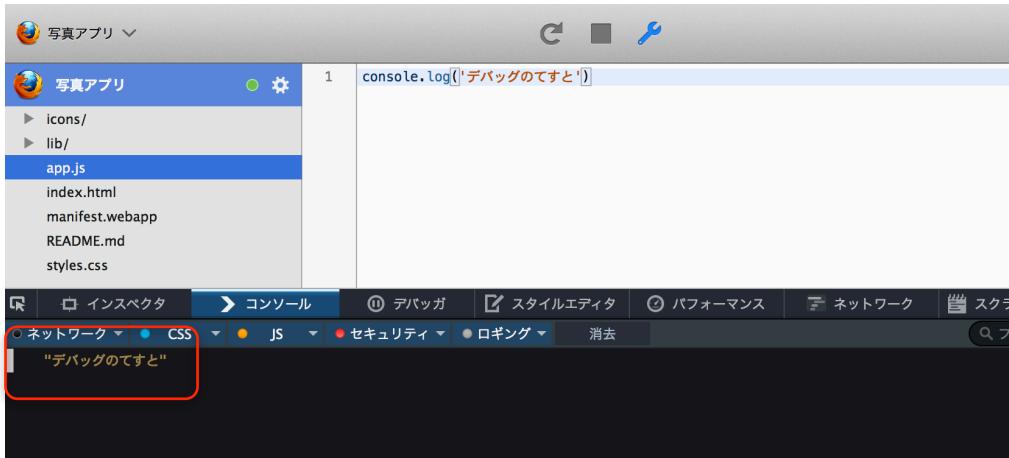
アプリケーションのデバッグを行います。

では、試しにデバッグ機能を使ってみましょう。



ファイル「app.js」に「`console.log('デバッグのてすと')`」と書いた後ファイルを保存、を押してアプリケーションを更新し、を押してみましょう。

以下のように、「>コンソール」が表示されて、デバッグルog(console.log)の内容が表示されるようになります。



他にも便利なツールがありますが、割愛します。

どのような機能があるかは、以下のサイトをご覧ください。

<https://www.mozilla.org/ja/firefox/developer/>

## アプリケーションの設定を編集しよう

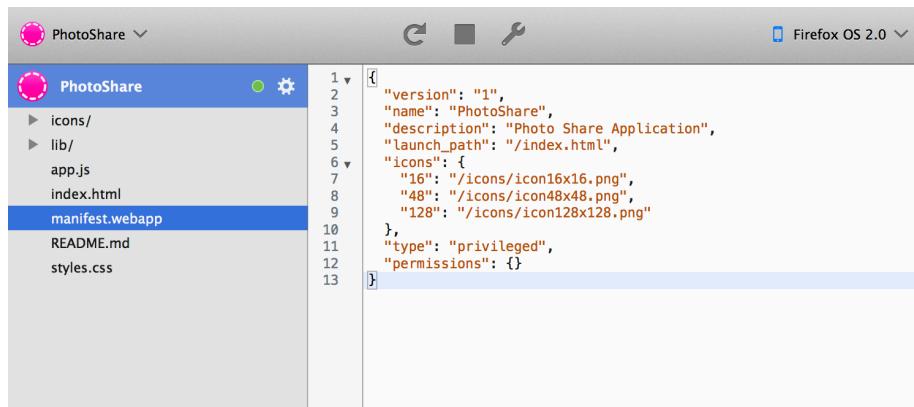
Firefox OS アプリケーションでは「manifest.webapp」と言うアプリマニフェストを作成する必要があります。

アプリマニフェストは、アプリに関する情報（名称、作者、アイコン、説明など）を、ユーザとアーリストアの双方が利用できるシンプルなドキュメントとして提供されています。

最も重要なことは、アプリが必要とする Web API のリストが含まれるという点です。

それにより、ユーザはアプリに関する十分な情報を得た上でそれをインストールすることが可能となります。

これは、Open Web App を Web サイトと区別する重要なポイントのひとつと言えるでしょう。



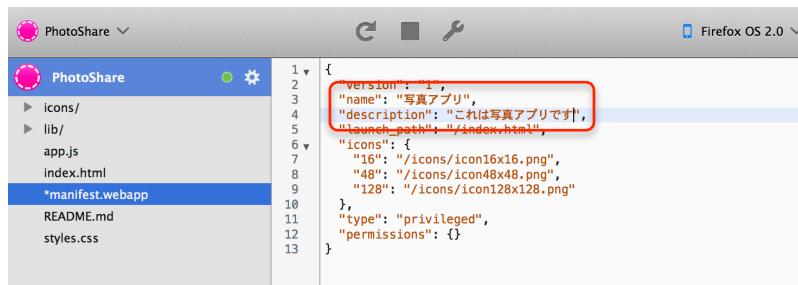
```

{
  "version": "1",
  "name": "PhotoShare",
  "description": "Photo Share Application",
  "launch_path": "/index.html",
  "icons": {
    "16": "/icons/icon16x16.png",
    "48": "/icons/icon48x48.png",
    "128": "/icons/icon128x128.png"
  },
  "type": "privileged",
  "permissions": {}
}

```

## アプリケーション名を変えてみよう

name がアプリケーション名、description がアプリケーションの説明になります。



```

{
  "version": "1",
  "name": "写真アプリ",
  "description": "これは写真アプリです",
  "launch_path": "/index.html",
  "icons": {
    "16": "/icons/icon16x16.png",
    "48": "/icons/icon48x48.png",
    "128": "/icons/icon128x128.png"
  },
  "type": "privileged",
  "permissions": {}
}

```

書き換えると、下のようにアプリケーション名と説明が変わります。



写真アプリ

PACKAGED PRIVILEGED

説明  
これは写真アプリです

場所  
/Users/msakamaki/project/mozilla/Fxos/photoshare

App ID  
app://b3570e72-6b38-7c49-bfd4-6d08c0b8a773/manifest.webapp

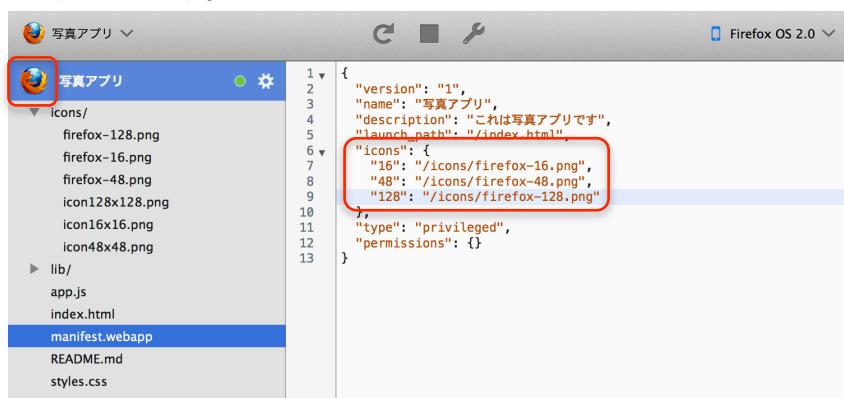
## アイコンを変えてみよう

icon は各解像度に合わせたアイコンが入っています。

これを「icon-XXX.png」から「firefox-XXX.png」へ変更してみましょう。



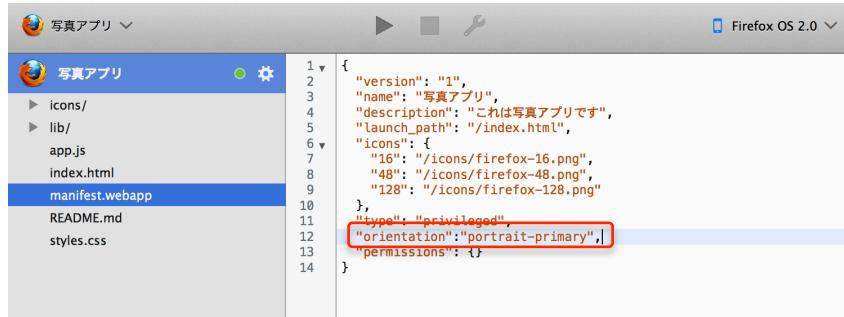
icons のファイル名を書き換えて、アプリマニフェストを保存すると、次のようにアプリケーションのアイコンが変更されます。



## 画面の回転を固定しよう

画面の回転があると、レイアウトの調整に手間取るので、今回は画面の回転を固定します。

画面の回転を固定する為にはアプリマニフェストに「"orientation":"portrait-primary"」を追加します。



The screenshot shows the Firefox OS 2.0 developer mode interface. On the left, there's a sidebar with project files: icons/, lib/, app.js, index.html, manifest.webapp, README.md, and styles.css. The manifest.webapp file is currently selected. On the right, the code editor displays the manifest file with line numbers 1 through 14. The line containing the "orientation" key is highlighted with a red box. The code is as follows:

```
1  {
2   "version": "1",
3   "name": "写真アプリ",
4   "description": "これは写真アプリです",
5   "launch_path": "/index.html",
6   "icons": {
7     "16": "/icons/firefox-16.png",
8     "48": "/icons/firefox-48.png",
9     "128": "/icons/firefox-128.png"
10  },
11  "type": "privileged",
12  "orientation": "portrait-primary",
13  "permissions": []
14 }
```

基本的なアプリマニフェストの設定はこれで完了です。

# 実機に接続してみよう

## 実機の設定変更/確認

USB接続をする前に、Firefox OSの設定を確認していきましょう。



環境設定まで戻ると、開発者が選べるようになっているので選びます。

USB経由のデバッグと言う項目があり、ここにADBと開発ツールとあれば問題ありません。



## 実機の接続

では早速 USB ケーブルで端末を接続してみましょう。

設定が無事完了していれば、下記のように Web IDE の USB デバイスに実機が表示されます。



後は Firefox OS シミュレーターと同じ操作で開発していくことができます。

## おまけ

開発中スリープに入らないよう、スクリーンロックをしない設定にしておきましょう。

(スクリーンがロックされると WebIDE との接続が切れてしまいます。)



# カメラアプリにしてみよう

これよりアプリにカメラ機能を追加していきます。

Firefox OS アプリケーションでカメラを使用するには「[Camera API](#)」を使用する方法と「[getUserMedia](#)」を使用する方法の2種類があります。

今回は比較的容易に実装できる `getUserMedia` を用いてカメラアプリを作ります。

※ [CameraAPI](#) を使用してみたい方は、ハンズオン完了後、独自に Camera API を使うように拡張してみください。

※ 開発に実機を使う場合は前章「[実機に接続してアプリを動かしてみよう](#)」を行ってください。

※ カメラのある PC であれば、開発中のカメラを PC の物で代替できます。

## カメラの入力を画面に映そう

### 権限を設定しよう

Firefox OS アプリでカメラを使用するには、アプリケーションに権限を与える必要があります。

与える権限は「video-capture」です。

`permissions` に以下のように設定を追加しておきましょう。

```

1  {
2      "version": "1",
3      "name": "写真アプリ",
4      "description": "これは写真アプリです",
5      "launch_path": "/index.html",
6      "icons": {
7          "16": "/icons/firefox-16.png",
8          "48": "/icons/firefox-48.png",
9          "128": "/icons/firefox-128.png"
10     },
11     "type": "privileged",
12     "orientation": "portrait-primary",
13     "permissions": {
14         "video-capture": {
15             "description": "カメラを使う"
16         }
17     }
18 }
```

## 機能を追加しよう

さっそくカメラ機能を追加して行きましょう。  
ここからは長いコードを書いて行きます。

まずは画面に「写真を取る」ボタンと<video>タグを追加します。  
写真を取るボタンを押す事で、video タグに映像を表示させてみましょう。

```
<body>
  <div>
    <div>
      <button id="shutter">写真を撮る</button>
      <span class="colorpick">
        <input type="text" value="" class="pickval hide" />
        <span class="add-on"><i></i></span>
      </span>
      <button id="undo">戻す</button>
      <button id="redo">進む</button>

      <span></span>
    </div>
    <div>
      <div>
        <video id="camera_video" class="hide"></video>
        <canvas id="canvas" class="show"></canvas>
      </div>
    </div>
  </body>
```

次にアプリケーションの本体であるファイル「app.js」にカメラを写す機能を追加していきます。



アプリケーション本体に、以下のソースコードを書いていきます。

コードの詳細な解説は次のページからです。

```
window.navigator.getUserMedia = ( navigator.getUserMedia || navigator.mozGetUserMedia);

// Flame size : 320 x 240
var constrainedWidth = 240;
var constrainedHeight = 320;

var $$ = function(selector) {return document.querySelector(selector); }
var $$$ = function(tag) { return document.createElement(tag); }

var FxOSApp = {
    /** 初期化処理 **/
    init: function() {
        FxOSApp.video.element = $$('#camera_video');
        FxOSApp.resize();
    },
    resize: function(e) {
        FxOSApp.video.element.width = constrainedWidth;
        FxOSApp.video.element.height = constrainedHeight;
    },
    /** canvas と video タグの切り替え **/
    show: {
        video: function() {
            $$('#camera_video').className = 'show';
            $$('#canvas').className = 'hide';
        },
        canvas: function() {
            $$('#camera_video').className = 'hide';
            $$('#canvas').className = 'show';
        }
    },
    /** ビデオ要素関係の操作 **/
    video: {
        shutter: function() {
            FxOSApp.show.video();
            window.navigator.getUserMedia({
                video: {
                    "mandatory": {
                        "minWidth": constrainedWidth,
                        "minHeight": constrainedHeight,
                        "minFrameRate": "30"
                    },
                    audio: false
                },
                FxOSApp.video.sucess,
                FxOSApp.video.error
            });
        },
        element: null,
        MediaStream: null,
        sucess: function(localMediaStream) {
            FxOSApp.video.MediaStream = localMediaStream;
            FxOSApp.video.element.src = window.URL.createObjectURL(localMediaStream);
            FxOSApp.video.element.play();
        },
        error: function(err) {
            console.log('The following error occured: ', err);
        }
    }
};

var onload_func = function() {
    // 初期化処理
    FxOSApp.init();

    /***** カメラを使う *****/
    $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
};

document.addEventListener('DOMContentLoaded', onload_func, false);
```

まずは初期設定を行いましょう。

※「//」より右側はコメントなので書かなくても問題ありません。

```
// 画像を取り込む機能 getUserMedia の設定をします。
// Firefox ではベンダープレフィックスのついた「mozGetUserMedia」を使います。
// 標準的な名前である getUserMedia としても使えるようにしましょう。
window.navigator.getUserMedia = ( navigator.getUserMedia || navigator.moz GetUserMedia );

// <canvas>と<video>の共通サイズを指定します。
// Flame size : 320 x 240
var constrainedWidth = 240;
var constrainedHeight = 320;

// querySelector と createElement はよく使うので、楽に書けるよう関数 $$ と $$$ として作っておきます。
var $$ = function(selector) { return document.querySelector(selector); }
var $$$ = function(tag) { return document.createElement(tag); };

// ここにアプリケーションの処理をこの関数に書いて行きます。
var FxOSApp = {};

// HTML が解釈された後に実行される処理(DOMContentLoaded event)をこの関数に書きます。
var onload_func = function() {};
document.addEventListener('DOMContentLoaded', onload_func, false);
```

次に関数 **FxOSApp** に処理を追加して行きます

```
var FxOSApp = {
    // ここに初回だけ実行される処理を書いておきます。処理内容は次のようになっています。
    init: function() {
        FxOSApp.video.element = $$('#camera_video');           // 1. <video>エレメントを取得する
        FxOSApp.resize();                                       // 2. <video>のサイズを設定する
    },
    // ここは<video>のサイズを変更する処理を書いておきます
    resize: function(e) {
        FxOSApp.video.element.width = constrainedWidth; // <video>の幅を設定します
        FxOSApp.video.element.height = constrainedHeight; // <video>の高さを設定します
    },
    show: {
        // <video>を表示して<canvas>を隠す処理です。
        video: function() {
            $$('#camera_video').className = 'show'; // <video>を表示させます
            $$('#canvas').className = 'hide'; // <canvas>を非表示にします
        },
        video: {} // ここに<video>関係の処理を纏めています
    },
    shutter: function() {
        FxOSApp.show.video(); // <video>を表示して<canvas>を隠します。
        window.navigator.getUserMedia({ // getUserMedia を使い、カメラから映像を取得します
            video: {
                mandatory: {
                    "minWidth": constrainedWidth,
                    "minHeight": constrainedHeight,
                    "minFrameRate": 30
                },
            },
            audio: false
        }, FxOSApp.video.sucess, // ここでは getUserMedia が成功した処理を書きます
        FxOSApp.video.error // ここには getUserMedia が失敗したときの処理を書きます
    },
    element: null,
    MediaStream: null,
    sucess: function(localMediaStream) { // getUserMedia が成功した処理です
        FxOSApp.video.MediaStream = localMediaStream; // 成功するとカメラからの映像が取得できます
        FxOSApp.video.element.src = window.URL.createObjectURL(localMediaStream); // 取得した映像を<video>タグに紐づけます。
        FxOSApp.video.element.play(); // video.play()で映像を画面に表示します
    },
    error: function(err) { // getUserMedia が失敗した処理です
        console.log("The following error occurred: ", err);
    }
};
```

最後に、HTML が読み込まれた後の処理を書いて行きます。

```
var onload_func = function() {
    // カメラを動かす為の初期設定を走らせます。
    FxOSApp.init();

    // 「写真を撮る」ボタンと、処理を紐づけます。
    $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
};

document.addEventListener('DOMContentLoaded', onload_func, false);
```

ここまで書いて問題がなければ、次のように「写真を撮る」ボタンを押す事で、画面にカメラの内容が表示されるようになります。

PC のカメラの性能により、縦横比がずれてしまう可能性がありますが、そのときは「constrainedWidth」と「constrainedHeight」で調整してみましょう。



## 写真を取ってみよう

これから、カメラで写している内容を<canvas>に移して、写真にお絵描きが出来るアプリにしてみましょう。コードは以下の濃い色の部分を追加して行きます。

解説は次のページからです。

```
window.navigator.getUserMedia = ( navigator.getUserMedia || navigator.mozGetUserMedia);

var constrainedWidth = 240;
var constrainedHeight = 320;

var $$ = function(selector) {return document.querySelector(selector); }
var $$$ = function(tag) { return document.createElement(tag); }

var FxOSApp = [
    /** 初期化処理 */
    init: function() {
        FxOSApp.canvas = $$('#canvas');
        FxOSApp.ctx = FxOSApp.canvas.getContext('2d');
        FxOSApp.video.element = $$('#camera_video');
        FxOSApp.resize();
    },
    /** キャンバスとカメラ入力の操作 */
    drawImg: function(img) {
        FxOSApp.ctx.drawImage(img, 0, 0, FxOSApp.canvas.width, FxOSApp.canvas.height);
    },
    resize: function(e) {
        FxOSApp.canvas.width = constrainedWidth;
        FxOSApp.canvas.height = constrainedHeight;
        FxOSApp.video.element.width = constrainedWidth;
        FxOSApp.video.element.height = constrainedHeight;
    },
    /** canvas と video タグの切り替え */
    show: {
        video: function() {
            $$('#camera_video').className = 'show';
            $$('#canvas').className = 'hide';
        },
        canvas: function() {
            $$('#camera_video').className = 'hide';
            $$('#canvas').className = 'show';
        }
    },
    /** ビデオ要素関係の操作 */
    video: [
        shutter: function() {
            FxOSApp.show.video();
            window.navigator.getUserMedia({
                video: {
                    "mandatory": {
                        "minWidth": constrainedWidth,
                        "minHeight": constrainedHeight,
                        "minFrameRate": "30"
                    },
                    audio: false
                },
                FxOSApp.video.sucsess,
                FxOSApp.video.error
            });
        },
        element: null,
        MediaStream: null,
        sucsess: function(localMediaStream) {
            FxOSApp.video.MediaStream = localMediaStream;
            FxOSApp.video.element.src = window.URL.createObjectURL(localMediaStream);
            FxOSApp.video.element.play();
            FxOSApp.video.element.addEventListener("click", function shot() {
                FxOSApp.drawImg(FxOSApp.video.element);
                FxOSApp.video.element.pause();
                FxOSApp.video.MediaStream.stop();
                FxOSApp.show.canvas();
                FxOSApp.video.element.removeEventListener("click", shot);
            });
        },
        error: function(err) {
            console.log("The following error occured: ", err);
        }
    ]
];

var onload_func = function() {
    FxOSApp.init();
    $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
};

document.addEventListener('DOMContentLoaded', onload_func, false);
```

まずは初期処理を追加しましょう、<canvas>を JavaScript で操作する準備をします。

```
var FxOSApp = {
  /** 初期化処理 */
  init: function() {
    FxOSApp.canvas = $$('#canvas');
    FxOSApp.ctx = FxOSApp.canvas.getContext('2d');
    FxOSApp.video.element = $$('#camera_video');
    FxOSApp.resize();
  }
};
```

<video>から<canvas>へ書き出す関数を用意します。(後で使います)

resize 処理に<canvas>のサイズ設定を追加します、ここで<video>と<canvas>のサイズを合わせます。

```
/** キャンバスとカメラ入力の操作 */
drawImg: function(img) {
  FxOSApp.ctx.drawImage(img, 0, 0, FxOSApp.canvas.width, FxOSApp.canvas.height);
},
resize: function(e) {
  FxOSApp.canvas.width = constrainedWidth;
  FxOSApp.canvas.height = constrainedHeight;
  FxOSApp.video.element.width = constrainedWidth;
  FxOSApp.video.element.height = constrainedHeight;
},
```

表示処理にも、<video>と<canvas>の表示切り替え処理を書いて行きましょう。

```
/** canvas と video タグの切り替え */
show: [
  video: function() {
    $$('#camera_video').className = 'show';
    $$('#canvas').className = 'hide';
  },
  canvas: function() {
    $$('#camera_video').className = 'hide';
    $$('#canvas').className = 'show';
  }
],
```

少し長いですが、最後に<video>の内容を<canvas>へ書き込む処理を書いて行きます。

現在<video>タグで表示されている内容を<canvas>へ書き出し、getUserMedia や<video>の処理をクリアしています。

```
success: function(localMediaStream) {
  FxOSApp.video.MediaStream = localMediaStream;
  FxOSApp.video.element.src = window.URL.createObjectURL(localMediaStream);
  FxOSApp.video.element.play();
  // この処理は<video>タグがクリックされると、実行される処理です。
  FxOSApp.video.element.addEventListener('click', function shot() {
    // <video>から<canvas>へ書き出す処理です
    FxOSApp.drawImg(FxOSApp.video.element);
    // video の秒が処理を止めます。
    FxOSApp.video.element.pause();
    // getUserMedia の処理を止めます。
    FxOSApp.video.MediaStream.stop();
    // 画面に<canvas>を表示して、<video>タグを隠します。
    FxOSApp.show.canvas();
    // <video>タグがクリックされたときの処理を解放します。
    FxOSApp.video.element.removeEventListener('click', shot);
  });
},
```

これでお絵描きアプリにカメラ機能が追加され、カメラで移している映像をクリックすると、<video>タグの内容が<canvas>に書き出されるようになりました。

# サーバーに接続してみよう

ここまでで写真機能付きのお絵描きアプリは完成です。

ここから機能を追加して、写真のアップロード機能をつけてみましょう。

## 作者と写真のタイトルを入力できるようにしよう

アップロードボタンと、写真が誰の写真か判るように、タイトルと作者を入力する欄を画面に設けましょう。

```
<body>
  <div>

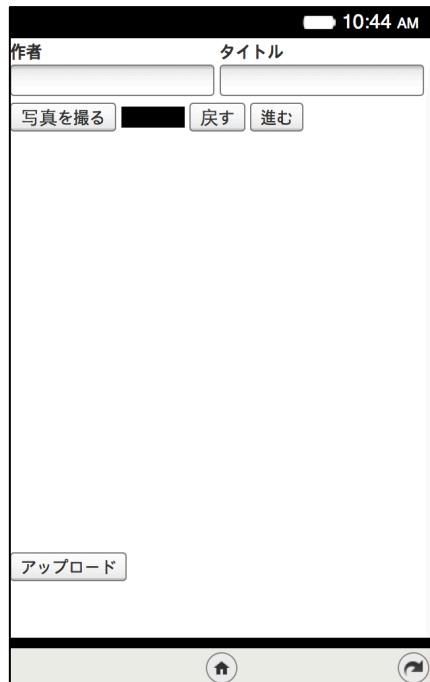
    <label>作者
    <div><input type="text" id="username"></div></label>

    <label>タイトル
    <div><input type="text" id="title"></div></label>

  </div>
  <div>
    <div>
      <button id="shutter">写真を撮る</button>
      <span class="colorpick">
        <input type="text" value="" class="pickval hide" />
        <span class="add-on"><i></i></span>
      </span>
      <button id="undo">戻す</button>
      <button id="redo">進む</button>

      <span></span>
    </div>
  </div>
  <div>
    <video id="camera_video" class="hide"></video>
    <canvas id="canvas" class="show"></canvas>
  </div>
  <div>
    <button id="imageSave">アップロード</button>
  </div>
</body>
```

次のように作者とタイトル、アップロードボタンが表示されるようになります。



## サーバーに情報を送ってみよう

これからサーバーへ写真を送る処理を追加していきます。

### サーバへの送信処理を書く

XMLHttpRequest を使ってサーバにデータを POST する処理を書いてみましょう。

上の処理はサーバにデータを送信する処理を書いています。

FxOSApp オブジェクトの中に追加しましょう。

下の処理は「アップロード」ボタンと上で書いた処理を紐づけています。

```

FxOSApp.video.element = $$('#camera_video');
FxOSApp.resize();
}
/** データの保存 */
service: {
  save: function() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function(data) {
      if( this.readyState === 4 && this.status === 200 ) {
        console.log('sucess!', data);
      }
    }
    xhr.open( 'POST', 'http://fxos-ps.azurewebsites.net/api/photos' );
    xhr.setRequestHeader( 'Content-Type', 'application/json; charset=utf-8' );
    xhr.send(JSON.stringify({
      img: FxOSApp.canvas.toDataURL(),
      title: $$('#title').value,
      usr: $$('#username').value
    }));
  },
  /** キャンバスとカメラ入力の操作 */
  drawImg: function(img) {
    FxOSApp.ctx.drawImage(img, 0, 0, FxOSApp.canvas.width, FxOSApp.canvas.height);
  }
}
~~~~~ 中略 ~~~~~
var onload_func = function() {
  FxOSApp.init();
  // アップロードボタンを押したときの処理
  $$('#imageSave').addEventListener('click', FxOSApp.service.save, false);
  $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
}
document.addEventListener('DOMContentLoaded', onload_func, false);

```

完成すると、以下のサイトに写真が共有されます。

ハンズオンが完了した事を、お絵描きした写真のアップロードでアピールしましょう。

<http://fxos-ps.azurewebsites.net/>

#### フォトギャラリー

投稿者: test タイトル: photo



投稿者: sakamaki タイトル: このはたん



投稿者: sakamaki タイトル: sample

# その他の API 紹介

ここまででアプリケーション作成は一旦終わりです。

これからは色々なデバイス API を使い、機能を追加して行きましょう。

今回スポットを当てる API 以外にも様々な物があるので、この機会に是非触ってみて下さい。

<https://developer.mozilla.org/ja/docs/WebAPI>

## Flame の仕様

### 写真を取ったときに振動させてみよう

#### バイブルーション API

バイブルーションハードウェアを操作するための API です。

写真を取った時に振動させるには「`window.navigator.vibrate()`」を、次場所に書きます。

```
success: function(localMediaStream) {
  FxOSApp.video.MediaStream = localMediaStream;
  FxOSApp.video.element.src = window.URL.createObjectURL(localMediaStream);
  FxOSApp.video.element.play();
  // この処理は<video>タグがクリックされると、実行される処理です。
  FxOSApp.video.element.addEventListener("click", function shot() {
    window.navigator.vibrate([200, 100, 200, 100, 200]);
    FxOSApp.drawImage(FxOSApp.video.element);
    FxOSApp.video.element.pause();
    FxOSApp.video.MediaStream.stop();
    FxOSApp.show.canvas();
    FxOSApp.video.element.removeEventListener("click", shot);
  });
},
```

配列で渡した場合、振動する時間(ミリ秒)、振動しない時間を交互に書いて行きます。

例では0.2秒振動した後、0.1秒とまるを3回繰り返します。

### 画面に触らず写真を取ってみよう

近接センサ API は、デバイスの近接センサによって、デバイスの近くに物体が存在するかどうかを示します。

今回追加するコードは「写真を撮る」ボタンと同じ動きを、画面を触れずに実行してみるサンプルです。

```
var onload_func = function() {
  FxOSApp.init();
  window.addEventListener('userproximity', FxOSApp.userproximity);
  $$('#imageSave').addEventListener('click', FxOSApp.service.save, false);
  $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
};
document.addEventListener('DOMContentLoaded', onload_func, false);
```

```
var FxOSApp = {
  ~~~~~
  error: function(err) {
    console.log("The following error occurred: ", err);
  },
  userproximity: function(event) {
    FxOSApp.video.shutter();
  }
};
```

## 明るさでペンの色を変更してみよう

### 環境光センサ(デバイスライト)API

環境光センサはデバイス周辺の照度を検出するセンサです。

今回追加するコードでは周囲の明るさに応じてペンの色を白(明るい)から黒(暗い)などをリアルタイムに変化させる動きをさせてみます。

```
var onload_func = function() {
    FxOSApp.init();
    window.addEventListener('devicelight', FxOSApp.devicelight);
    $$('#imageSave').addEventListener('click', FxOSApp.service.save, false);
    $$('#shutter').addEventListener('click', FxOSApp.video.shutter, false);
};

document.addEventListener('DOMContentLoaded', onload_func, false);

var FxOSApp = {
    ~~~~~
    error: function(err) {
        console.log("The following error occurred: ", err);
    }
};

deviceLight: function(event) {
    // 光は0~500を想定
    var rgbRight = ((event.value >= 500) ? 500 : event.value) / 2;
    var palet = $$('.pickval').value || "#000000";
    var color = {
        r: parseInt(palet.substr(1, 2), 16),
        g: parseInt(palet.substr(3, 2), 16),
        b: parseInt(palet.substr(5, 2), 16),
        value: () => {
            return "#"
                + ("0" + color.r.toString(16)).slice(-2)
                + ("0" + color.g.toString(16)).slice(-2)
                + ("0" + color.b.toString(16)).slice(-2)
        }
    };
    color.r = parseInt(rgbRight);
    color.g = parseInt(rgbRight);
    color.b = parseInt(rgbRight);
    $$('.colorpick').colorpicker('setValue', color.value())
}
};
```

## アップロードした事を通知しよう

### Web Notifications API

Web Notifications API は、システムレベルでページ外部に表示される通知を Web ページから送ることを可能にします。

今回は、サーバへのアップロード時に通知を行う処理を入れてみましょう。

Notifications API を使うには権限が必要です。

manifest.webapp に「desktop-notification」を設定しましょう。

```
type : "privileged",
"orientation": "portrait-primary",
"permissions": {
  "video-capture": {}
  "desktop-notification": {
    "description": "デスクトップ通知を行います。"
  }
}
```

次にサーバへ情報を送り出した時の通知処理を書いてみます。

```
/** データの保存 */
service: {
  save: function() {
    // ユーザー名とタイトルは必須！
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function(data) {
      if( this.readyState == 4 && this.status == 200 ) {
        console.log(' sucess!', data);
      }
    }
    xhr.open( 'POST', 'http://fxos-ps.azurewebsites.net/api/photos' );
    xhr.setRequestHeader( 'Content-Type', 'application/json; charset=utf-8' );
    xhr.send(JSON.stringify({
      img: FxOSApp.canvas.toDataURL(),
      title: $$('#title').value,
      usr: $$('#username').value
    }));
    var notification = new Notification(' サーバへ保存しました。', {
      body: 'インターネットの世界へ送り出しました',
      icon: window.location.origin + '/icons/firefox-16.png'
    });
  },
}
```

以上で通知処理は完了です。

写真をサーバーに送るとき、通知が発生するようになります。

## カメラ API に置き換えてみよう

### Camera API

Camera API によってデバイスのカメラで写真を撮影して、その写真をアプリケーションで使う事が出来ます。

これは、type="file" および画像を受け入れることを宣言する accept 属性を持つ input 要素によって実現します。以下のような HTML です。

```
</div>
</div>
<input type="file" id="camera" accept="image/*" class="hide">
<video id="camera_video" class="hide"></video>
<canvas id="canvas" class="show"></canvas>
</div>
</div>
<button id="imageSave">アップロード</button>
```

カメラ API を使う input 要素を用意したら、次に処理を書いて行きましょう。  
以下のコードがカメラから画像を取得して、キャンバスに設定する処理です。

```
var FxOSApp = {
~~~~~中略~~~~~
/** カメラの操作 */
camera: {
  shutter: function() {
    var camera = $$('#camera');
    camera.onchange = function(event) {
      // 撮影された写真または選択された画像への参照を取得
      var files = event.target.files;
      if (files && files.length > 0) {
        var file = files[0];
        var img = $$('img');
        img.onload = function() {
          FxOSApp.drawImage(img);
        };
        var dataURL = URL.createObjectURL(file);
        img.src = dataURL;
        URL.revokeObjectURL(dataURL);
      }
    }
  }
},
};

camera.click();
```

最後に、「写真を撮る」ボタンのアクションを「FxOSApp.video.shutter」から「FxOSApp.camera.shutter」に変更しましょう。

```
var onload_func = function() {
  FxOSApp.init();
  $$('#imageSave').addEventListener('click', FxOSApp.service.save, false);
  $$('#shutter').addEventListener('click', FxOSApp.camera.shutter, false);
};

document.addEventListener('DOMContentLoaded', onload_func, false);
```

これで、getUserMedia の代わりに、Firefox OS のカメラ API を使って写真を撮るようになりました。