Using Direct Update in Cordova applications

Overview

With Direct Update, Cordova applications can be updated "over-the-air" with refreshed web resources, such as changed, fixed or new applicative logic (JavaScript), HTML, CSS or images. Organizations are thus able to ensure that end-users always use the latest version of the application.

In order to update an application, the updated web resources of the application need to be packaged and uploaded to the MobileFirst Server using the MobileFirst CLI or by deploying a generated archive file. Direct Update is then activated automatically Once activated, it will be enforced on every request to a protected resource.

Supported Cordova platforms

Direct Update is supported in the Cordova iOS and Cordova Android platforms.

Direct Update in development, testing, and production

For development and testing purposes, developers typically use Direct Update by simply uploading an archive to the development server. While this process is easy to implement, it is not very secure. For this phase, an internal RSA key pair that is extracted from an embedded MobileFirst self-signed certificate is used.

For the phases of live production or even pre-production testing, however, it is strongly recommended to implement secure Direct Update before you publish your application to the app store. Secure Direct Update requires an RSA key pair that is extracted from a real CA signed server certificate.

Note: Take care that you do not modify the keystore configuration after the application was published, updates that are downloaded can no longer be authenticated without reconfiguring the application with a new public key and republishing the application. Without performing these two steps, Direct Update fails on the client.

> Learn more in Secure Direct Update.

Direct Update data transfer rates

At optimal conditions, a single MobileFirst Server can push data to clients at the rate of 250 MB per second. If higher rates are required, consider a cluster or a CDN service.

> Learn more in Serving Direct Update requests from a CDN (cdn-support)

Notes

- Direct Update updates only the application's web resources. To update native resources a new application version must be submitted to the respective app store.
- When you use the Direct Update feature and the web resources checksum (../cordova-apps/securing-apps/#enabling-the-web-resources-checksum-feature) feature is enabled, a new checksum base is established with each Direct Update.
- If the MobileFirst Server was upgraded by using a fix pack, it continues to serve direct updates properly. However, if a recently built Direct Update archive (.zip file) is uploaded, it can halt updates to older clients. The reason is that the archive contains the version of the cordova-plugin-mfp plug-in. Before it serves that archive to a mobile client, the server compares the client version with the

plug-in version. If both versions are close enough (meaning that the three most significant digits are identical), Direct Update occurs normally. Otherwise, MobileFirst Server silently skips the update. One solution for the version mismatch is to download the cordova-plugin-mfp with the same version as the one in your original Cordova project and regenerate the Direct Update archive.

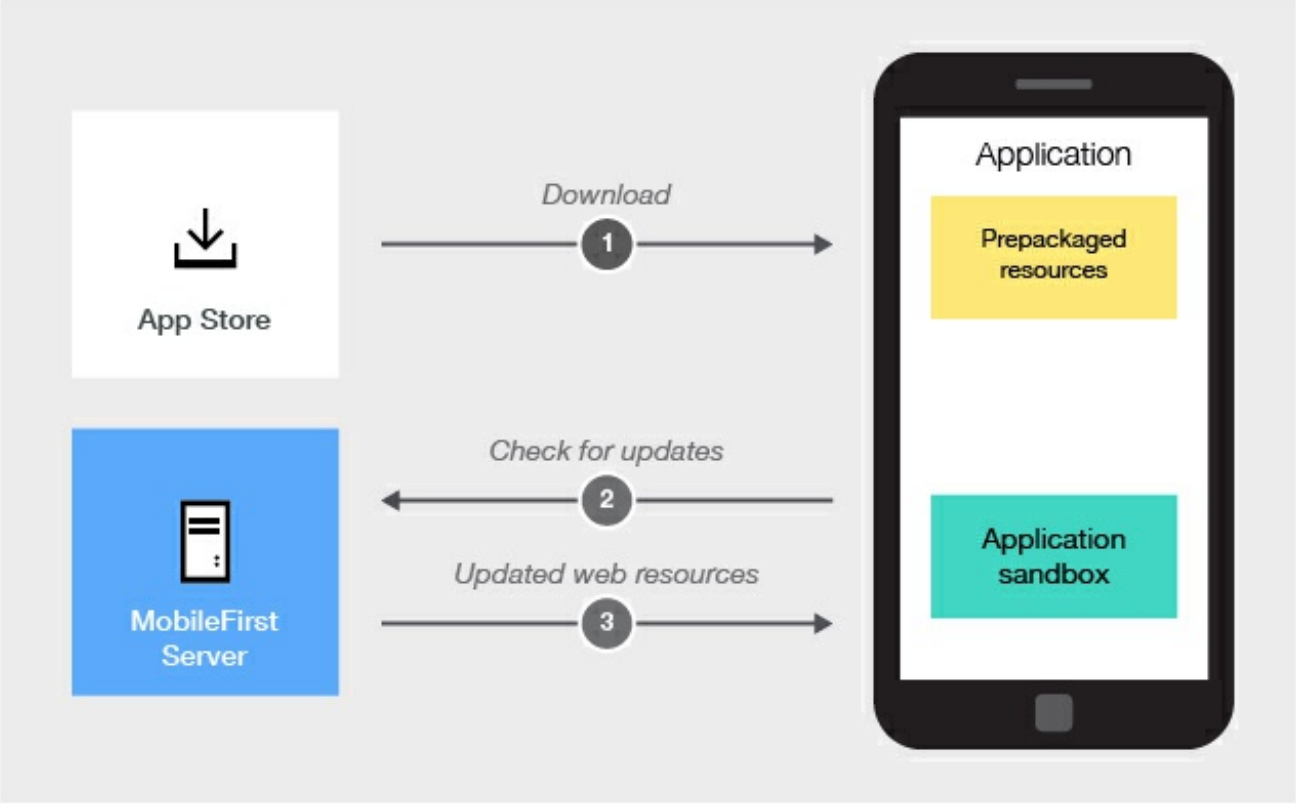
Jump to:

How Direct Update works

The application web resources are initially packaged with the application to ensure first offline availability. Afterwards, the application checks for updates on every request to the MobileFirst Server.

> **Note:** after a Direct Update was performed, it is checked for again after 60 minutes.

After a Direct Update, the application no longer uses the pre-packaged web resources. Instead, it will use the downloaded web resources from the application's sandbox. If the application's cache on the device will be cleared, the original packaged web resources will be used again.



Versioning

A Direct Update applies only to a specific version. In other words, updates generated for an application versioned 2.0 cannot be applied to a different version of the same application.

Creating and deploying updated web resources

Once work on new web resources, such as bug fixes or minor changes and the like, is done, the updated web resources need to be packaged and uploaded to the MobileFirst Server.

1. Open a **Command-line** window and navigate to the root of the Cordova project.
2. Run the command: `mfpdev app webupdate`.

The `mfpdev app webupdate` command packages the updated web resources to a .zip file and uploads it to the default MobileFirst Server running in the developer workstation. The packaged web resources can be found at the **[cordova-project-root-folder]/mobilefirst/** folder.

Alternatives:

- Build the .zip file and upload it to a different MobileFirst Server: `mfpdev app webupdate [server-name] [runtime-name]`. For example:

  ```
  mfpdev app webupdate myQAServer MyBankApps
  ```

- Upload a previously generated .zip file: `mfpdev app webupdate [server-name] [runtime-name] --file [path-to-packaged-web-resources]`. For example:

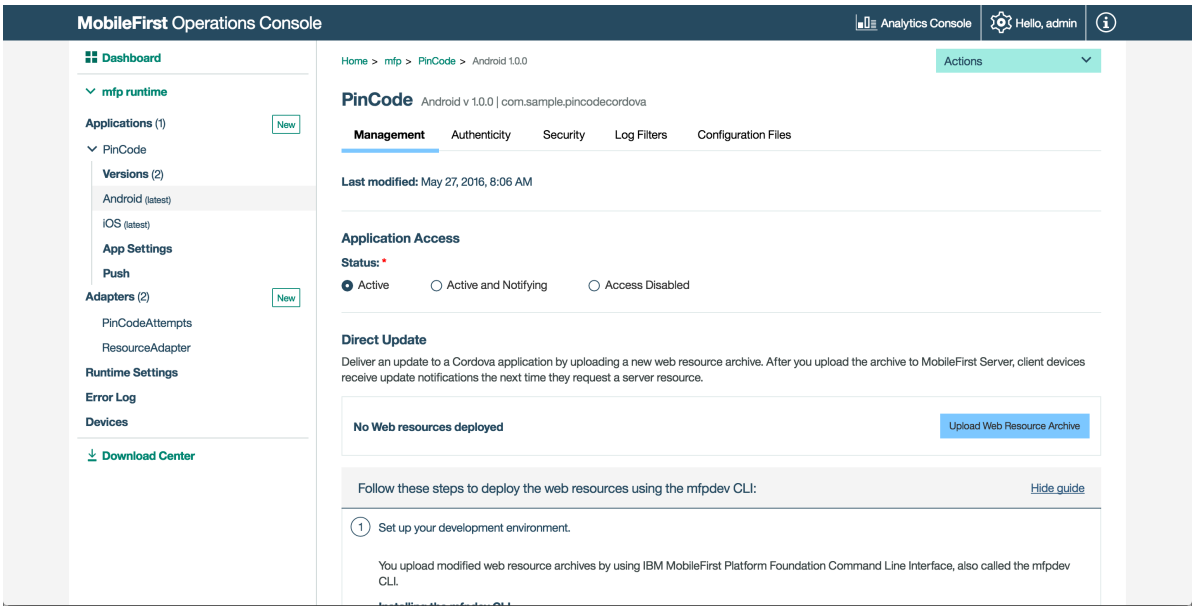
  ```
  mfpdev app webupdate myQAServer MyBankApps --file mobilefirst/ios/com.mfp.myBankApp-1.0.1.zip
  ```

- Manually upload packaged web resources to the MobileFirst Server:

  1. Build the .zip file without uploading it:

     ```
     mfpdev app webupdate --build
     ```

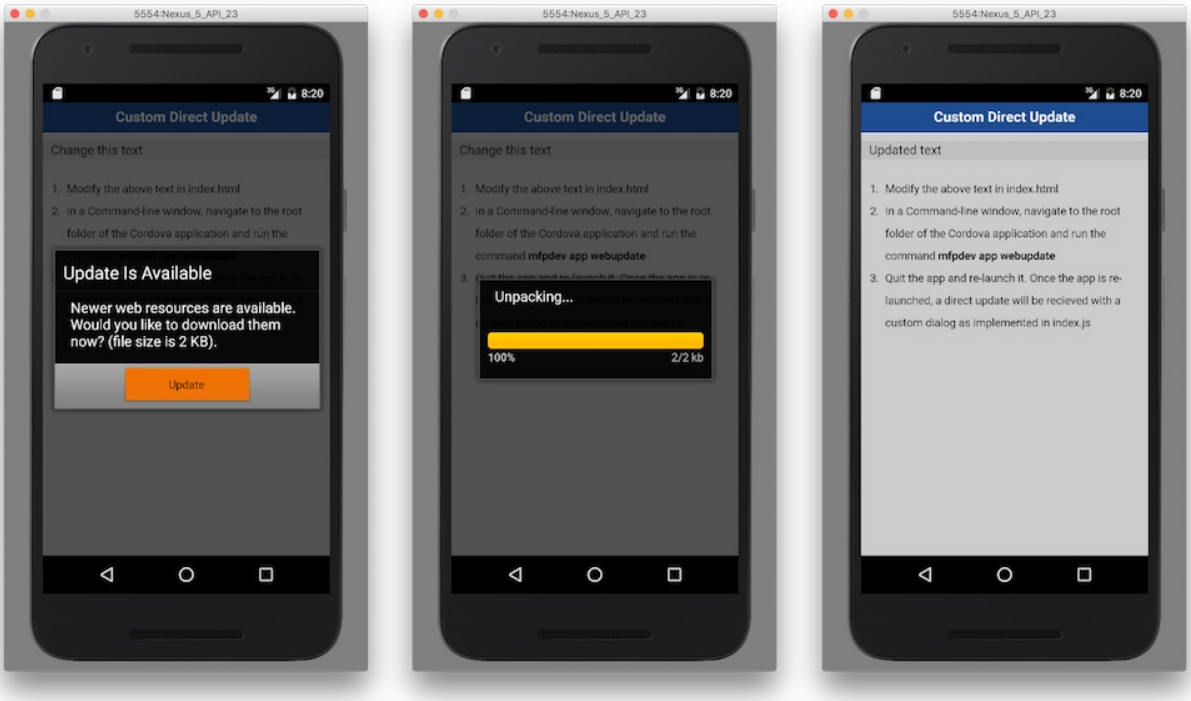
  2. Load the MobileFirst Operations Console and click on the application entry.
  3. Click on **Upload Web Resources File** to upload the packaged web resources.



> Run the command `mfpdev help app webupdate` to learn more.

User Experience

By default, after a Direct Update is received a dialog is displayed and the user is asked whether to begin the update process. After the user approves a progress bar dialog is displayed and the web resources are downloaded. The application is automatically reloaded after the update is complete.



Customizing the Direct Update UI

The default Direct Update UI that is presented to the end-user can be customized.
Add the following inside the `wlCommonInit()` function in **index.js**:

```
wl_DirectUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContex
t) {
    // Implement custom Direct Update logic
};
```

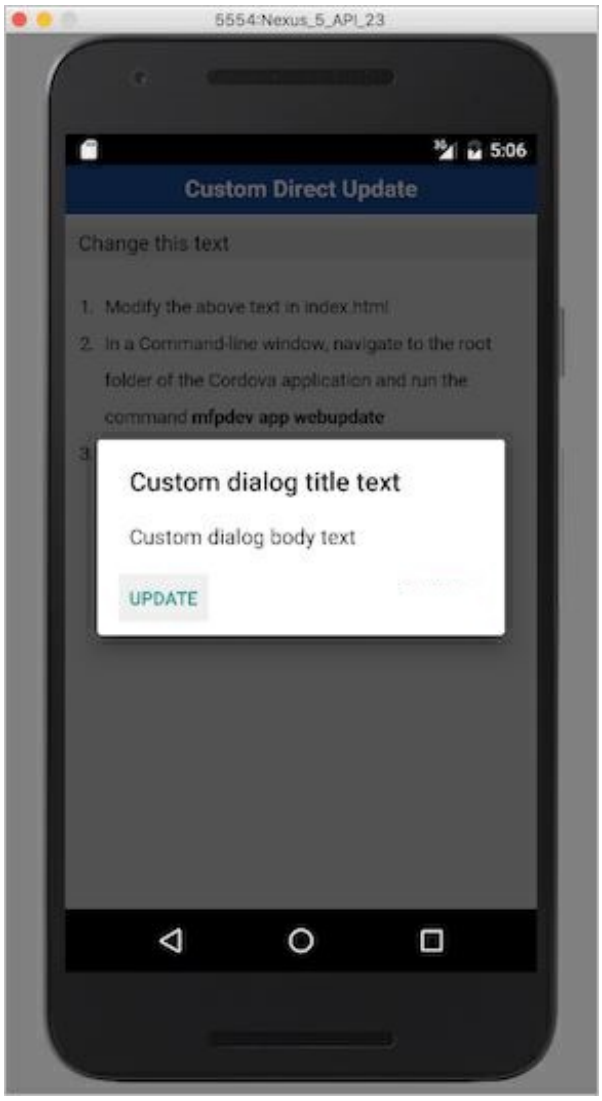
- `directUpdateData` - A JSON object containing the `downloadSize` property that represents the file size (in bytes) of the update package to be downloaded from MobileFirst Server.
- `directUpdateContext` - A JavaScript object exposing the `.start()` and `.stop()` functions, which start and stop the Direct Update flow.

If the web resources are newer on the MobileFirst Server than in the application, Direct Update challenge data is added to the server response. Whenever the MobileFirst client-side framework detects this direct update challenge, it invokes the `wl_directUpdateChallengeHandler.handleDirectUpdate` function.

The function provides a default Direct Update design: a default message dialog that is displayed when a Direct Update is available and a default progress screen that is displayed when the direct update process is initiated. You can implement custom Direct Update user interface behavior or customize the Direct Update dialog box by overriding this function and implementing your own logic.

In the example code below, a `handleDirectUpdate` function implements a custom message in the Direct Update dialog. Add this code into the **www/js/index.js** file of the Cordova project.
Additional examples for a customized Direct Update UI:

- A dialog that is created by using a third-party JavaScript framework (such as Dojo or jQuery Mobile, Ionic, ...)
- Fully native UI by executing a Cordova plug-in
- An alternate HTML that is presented to the user with options
- And so on...




```
wl_directUpdateChallengeHandler.handleDirectUpdate
= function(directUpdateData, directUpdateContext) {

   navigator.notification.confirm(  // Creates a dialog.
      'Custom dialog body text',
      // Handle dialog buttons.
        directUpdateContext.start();
      },
      'Custom dialog title text',
      ['Update']
   );
};
```

You can start the Direct Update process by running the `directUpdateContext.start()` method whenever the user clicks the dialog button. The default progress screen, which resembles the one in previous versions of MobileFirst Server is shown.

This method supports the following types of invocation:

- When no parameters are specified, the MobileFirst Server uses the default progress screen.
- When a listener function such as `directUpdateContext.start(directUpdateCustomListener)` is supplied, the Direct Update process runs in the background while the process sends lifecycle events to the listener. The custom listener must implement the following methods:

```
var directUpdateCustomListener  = {
   onStart : function ( totalSize ){ },
   onProgress : function ( status , totalSize , completedSize ){ },
   onFinish : function ( status ){ }
};
```

The listener methods are started during the direct update process according to following rules: * `onStart` is called with the `totalSize` parameter that holds the size of the update file. * `onProgress` is called multiple times with status `DOWNLOAD_IN_PROGRESS`, `totalSize`, and `completedSize` (the volume that is downloaded so far). * `onProgress` is called with status `UNZIP_IN_PROGRESS`. * `onFinish` is called with one of the following final status codes:

Status code	Description
`SUCCESS`	Direct update finished with no errors.
`CANCELED`	Direct update was canceled (for example, because the `stop()` method was called).
`FAILURE_NETWORK_PROBLEM`	There was a problem with a network connection during the update.
`FAILURE_DOWNLOADING`	The file was not downloaded completely.
`FAILURE_NOT_ENOUGH_SPACE`	There is not enough space on the device to download and unpack the update file.
`FAILURE_UNZIPPING`	There was a problem unpacking the update file.
`FAILURE_ALREADY_IN_PROGRESS`	The start method was called while direct update was already running.
`FAILURE_INTEGRITY`	Authenticity of update file cannot be verified.
`FAILURE_UNKNOWN`	Unexpected internal error.

If you implement a custom direct update listener, you must ensure that the app is reloaded when the direct update process is complete and the `onFinish()` method has been called. You must also call `wl_directUpdateChalengeHandler.submitFailure()` if the direct update process fails to complete successfully.

The following example shows an implementation of a custom direct update listener:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
    //show custom progress dialog
  },
  onProgress: function(status,totalSize,completedSize){
    //update custom progress dialog
  },
  onFinish: function(status){

    if (status == 'SUCCESS'){
      //show success message
      WL.Client.reloadApp();
    }
    else {
      //show custom error message

      //submitFailure must be called is case of error
      wl_directUpdateChallengeHandler.submitFailure();
    }
  }
};

wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext
){

  WL.SimpleDialog.show('Update Avalible', 'Press update button to download version 2.0', [{
    text : 'update',
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Scenario: Running UI-less direct updates

IBM MobileFirst Foundation supports UI-less direct update when the application is in the foreground.

To run UI-less direct updates, implement `directUpdateCustomListener`. Provide empty function implementations to the `onStart` and `onProgress` methods. Empty implementations cause the direct update process to run in the background.

To complete the direct update process, the application must be reloaded. The following options are available: * The `onFinish` method can be empty as well. In this case, direct update will apply after the application has restarted. * You can implement a custom dialog that informs or requires the user to restart the application. (See the following example.) * The `onFinish` method can enforce a reload of the application by calling `WL.Client.reloadApp()`.

Here is an example implementation of `directUpdateCustomListener`:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
  },
  onProgress: function(status,totalSize,completeSize){
  },
  onFinish: function(status){
    WL.SimpleDialog.show('New Update Available', 'Press reload button to update to new version', [ {
      text : WL.ClientMessages.reload,
      handler : WL.Client.reloadApp
    }]);
  }
};
```

Implement the `wl_directUpdateChallengeHandler.handleDirectUpdate` function. Pass the `directUpdateCustomListener` implementation that you have created as a parameter to the function. Make sure `directUpdateContext.start(directUpdateCustomListener)` is called. Here is an example `wl_directUpdateChallengeHandler.handleDirectUpdate` implementation:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext
){

  directUpdateContext.start(directUpdateCustomListener);
};
```

Note: When the application is sent to the background, the direct-update process is suspended.

Scenario: Handling a direct update failure

This scenario shows how to handle a direct update failure that might be caused, for example, by loss of connectivity. In this scenario, the user is prevented from using the app even in offline mode. A dialog is displayed offering the user the option to try again.

Create a global variable to store the direct update context so that you can use it subsequently when the direct update process fails. For example:

```
var savedDirectUpdateContext;
```

Implement a direct update challenge handler. Save the direct update context here. For example:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext
){

  savedDirectUpdateContext = directUpdateContext; // save direct update context

  var downloadSizeInMB = (directUpdateData.downloadSize / 1048576).toFixed(1).replace(".", WL.App.ge
tDecimalSeparator());
  var directUpdateMsg = WL.Utils.formatString(WL.ClientMessages.directUpdateNotificationMessage, dow
nloadSizeInMB);

  WL.SimpleDialog.show(WL.ClientMessages.directUpdateNotificationTitle, directUpdateMsg, [{
    text : WL.ClientMessages.update,
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Create a function that starts the direct update process by using the direct update context. For example:

```
restartDirectUpdate = function () {
  savedDirectUpdateContext.start(directUpdateCustomListener); // use saved direct update context to resta
rt direct update
};
```

Implement `directUpdateCustomListener`. Add status checking in the `onFinish` method. If the status starts with "FAILURE", open a modal only dialog with the option "Try Again". For example:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
    alert('onStart: totalSize = ' + totalSize + 'Byte');
  },
  onProgress: function(status,totalSize,completeSize){
    alert('onProgress: status = ' + status + ' completeSize = ' + completeSize + 'Byte');
  },
  onFinish: function(status){
    alert('onFinish: status = ' + status);
    var pos = status.indexOf("FAILURE");
    if (pos > -1) {
      WL.SimpleDialog.show('Update Failed', 'Press try again button', [ {
        text : "Try Again",
        handler : restartDirectUpdate // restart direct update
      }]);
    }
  }
};
```

When the user clicks the **Try Again** button, the application restarts the direct update process.

Delta and Full Direct Update

Delta Direct Updates enables an application to download only the files that were changed since the last update instead of the entire web resources of the application. This reduces download time, conserves bandwidth, and improves overall user experience.

❗ **Important:** A **delta update** is possible only if the client application's web resources are one version behind the application that is currently deployed on the server. Client applications that are more than one version behind the currently deployed application (meaning the application was deployed to the server at least twice since the client application was updated), receive a **full update** (meaning that the entire web resources are downloaded and updated).

Secure Direct Update

Disabled by default, Secure Direct Update prevents a 3rd-party attacker from altering the web resources that are transmitted from the MobileFirst Server (or from a Content Delivery Network (CDN)) to the client application.

To enable Direct Update authenticity:
Using a preferred tool, extract the public key from the MobileFirst Server keystore and convert it to base64.
The produced value should then be used as instructed below:

1. Open a **Command-line** window and navigate to the root of the Cordova project.
2. Run the command: `mfpdev app config` and select the "Direct Update Authenticity public key" option.
3. Provide the public key and confirm.

Any future Direct Update deliveries to client applications will be protected by Direct Update authenticity.

> To configure the application server with the updated keystore file, see Implementing secure Direct Update (secure-direct-update)

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomDirectUpdate/tree/release80) the Cordova project.

Sample usage

Follow the sample's README.md file for instructions.