Crosswalk Shared Mode

This document aims to provide features description, build configuration, tools integration and the details of new embedding APIs for shared mode.

1. Summary

Shared mode allows multiple Crosswalk applications to share one Crosswalk runtime. Each Crosswalk application is bundled with a reflection layer instead of the full library files, whereas there would be one and only one APK of Crosswalk runtime to be installed on the device for the Crosswalk applications to use.

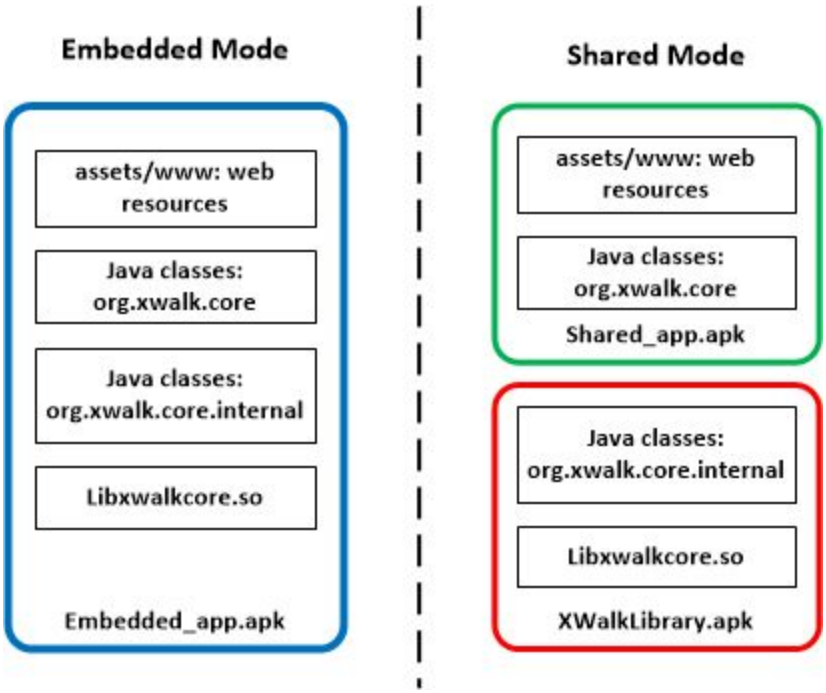

Figure: Embedded Mode vs. Shared Mode

The APK of Crosswalk runtime is published on Google Play Store and Crosswalk Project's official website. The end-users are able to get the APK via Google Play Store or the developer's self-hosting service. The Crosswalk runtime will be upgraded periodically to introduce the latest feature of Crosswalk and Chromium. The developer has the responsibility to follow up the update cycle and keep their applications compatible with the latest version of Crosswalk runtime.

Pros:

- Produces a significant smaller APK size for Crosswalk applications. For example of packaging a simple HelloWorld web application, the APK file size is 20MB for ARM and 23MB for x86. If the same contents is packaged in shared mode, the APK file size will shrink to 68KB.
- The Crosswalk application built in shared mode can run on both of the IA and ARM devices, even the application built in embedded mode for ARM can run on the IA devices, as long as there is a Crosswalk runtime installed on the device.
- Great change for the release of Crosswalk application. New Crosswalk feature and improvement can be delivered separately from applications. It also provides the possibility to integrate Crosswalk runtime into customized ROM.

Cons:

- For devices that the Crosswalk runtime is not installed, the end-user have to download the APK of Crosswalk runtime in the first run. The normal startup process of the application will be interrupted.

- The new Crosswalk runtime may be incompatible with the old Crosswalk applications. The developer need to do the beta testing beforehand before each release. Embedded mode is recommended for the developers who want to stick to a specific version of Crosswalk.

2. Feature Description

2.1. Version Check Mechanism

This is to guarantee the compatibility between the Crosswalk application and Crosswalk runtime.

Along with the version name like 13.38.208.0 already in used, each release of Crosswalk will also have an independent version which is an incremental version code to show the revision of embedding API. This version is recorded in API_VERSION file as below:

```
API: 4
MIN_API: 2
```

The API version is for both of the Crosswalk application and Crosswalk runtime. The MIN_API version is only for the Crosswalk runtime, it indicates the minimal version of Crosswalk applications that the Crosswalk runtime would support.

The Crosswalk application will check the version of Crosswalk runtime installed on the device before loading. If the version of application is between the current version and minimal backward-compatible version of Crosswalk runtime, it means the version matched. Otherwise, a dialog will pop up to to prompt the user to get the latest version of Crosswalk runtime.

2.2. Architecture Independent

The Crosswalk application in embedded mode will check whether the architecture of the embedding Crosswalk runtime matches the device. If the architecture matched, the application will use the embedding Crosswalk runtime directly. Otherwise, it will switch to shared mode.

The Crosswalk application in shared mode will check the architecture of the Crosswalk runtime installed on the device at startup before loading. If the architecture doesn’t match, a dialog will pop up to to prompt the user to get the suitable Crosswalk runtime.

2.3. Security Check

Before the application loading the Crosswalk runtime, it is necessary to make sure the APK of Crosswalk runtime is not tampered and is exactly the one we published. The officially released APK of Crosswalk runtime is signed with the signature from codesign.intel.com

The verification process are:

A. Get signatures of the package of Crosswalk runtime via package manager
B. Generate SHA1 fingerprint with each signature
C. Check against the fingerprint from codesign.intel.com

2.4. Built-in Updater

There is a built-in updater to help the user to get the APK of Crosswalk runtime. By default, the updater will jump to the page of Crosswalk runtime on Google Play Store, subsequent process will be up to the user. If the developer specified the download URL, the updater will launch the download manager to fetch the APK.

The developer can also distribute and maintain the specific version of Crosswalk runtime in their own way. The built-in updater won't be activated as long as the version of the installed Crosswalk runtime matches the Crosswalk application.

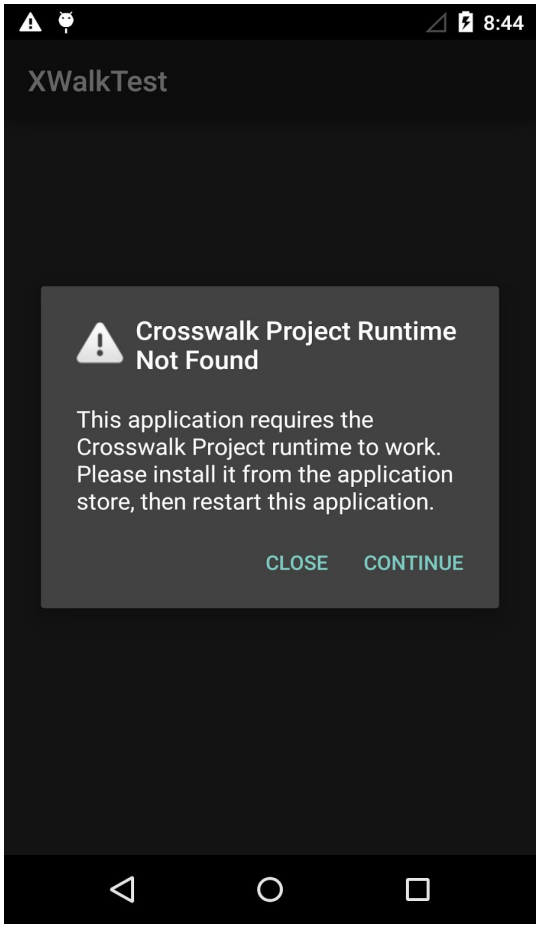

Figure: Built-in Updater

3. Tools Integration

3.1. Enable Shared mode

- **From Packaging Tool**

To package the application in shared mode, use the option:

```
./make_apk.py --mode=shared ...
```

- **From Cordova 3.x**

To create a cordova project in shared mode, use the option:

```
./bin/create --xwalk-shared-library ...
```

- **From Cordova 4.x**

Coming soon.

- **From Eclipse ADT**

Download the Crosswalk package with the file name crosswalk-[version].zip from the link below. Unzip the package and there is a project directory called xwalk_shared_library in it. Reference this library project from Eclipse.

https://download.01.org/crosswalk/releases/crosswalk/android/

- **From Android Studio**

Download the AAR with the file name crosswalk-shared-[version].aar from the link below. Import this AAR from your project.

https://download.01.org/crosswalk/releases/crosswalk/android/

3.2. Configure the download URL(NOT necessary)

- **From Packaging Tool**

Use the option below for make_apk.py:

```
./make_apk.py --xwalk-apk-url=http://host/XWalkRuntimeLib.apk ...
```

- **From The Crosswalk Manifest**

Define an element with the key “xwalk_apk_url”:

```
{
    "name": "XWalkApp",
    "start_url": "index.html",
    "xwalk_apk_url": "http://10.0.2.2/XWalkRuntimeLib.apk",
    ...
}
```

- **From The Android Manifest**

Define a meta-data element with the name "xwalk_apk_url" inside the application tag:

```
<application>
    <meta-data android:name="xwalk_apk_url"
               android:value="http://host/XWalkRuntimeLib.apk" />
    ...
</application>
```

4. Embedding API

This section describes what extra effort is needed to take the benefit of shared mode. The full instructions about embedding API is shown on https://crosswalk-project.org/documentation/embedding_crosswalk.html

4.1. Permissions

To download the Crosswalk runtime, you need to grant following permissions in the Android manifest:

```
 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
 <uses-permission android:name="android.permission.INTERNET" />
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

4.2. XWalkActivity

`XWalkActivity` helps to execute all procedures to make the Crosswalk runtime workable and displays dialogs to interact with the user if needed. The activities that hold the `XWalkView` objects might want to extend `XWalkActivity` to obtain this capability. For those activities, there's no need to use `XWalkInitializer` and `XWalkUpdater`.

`XWalkActivity` will try to run in embedded mode firstly. If the Crosswalk runtime hasn't been embedded in the app, or the embedded Crosswalk runtime doesn't match the CPU architecture of the device, it will switch to shared mode, i.e., link to the separately installed Crosswalk runtime package which is shared by multiple apps. If the Crosswalk runtime hasn't been installed on the device, or the installed Crosswalk runtime isn't compatible with the app, it will pop up dialogs to prompt the user to download suitable one. Once the user has agreed to download, it will navigate to the page of Crosswalk runtime on the default application store,

subsequent process will be up to the user. If the developer specified the download URL of the Crosswalk runtime, it will launch the download manager to fetch the package.

In old versions, the developer can use the embedding API in `onCreate()` directly or any where at any time as they wish. But in latest version, the Crosswalk runtime isn't loaded yet at the moment the activity is created, so the embedding API won't be usable immediately. To make your code compatible with new implementation somtimes, all routines using the embedding API should be inside `onXWalkReady()` or after `onXWalkReady()` is invoked. Please refer to following example for more details.

```
public class MyActivity extends XWalkActivity {
     XWalkView mXWalkView;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);

         // Until onXWalkReady() is invoked, you should do nothing with the
         // embedding API except the following:
         // 1. Instanciate the XWalkView object
         // 2. Call XWalkPreferences.setValue()
         // 3. Call XWalkView.setUIClient()
         // 4. Call XWalkView.setResourceClient()

         XWalkPreferences.setValue(XWalkPreferences.ANIMATABLE_XWALK_VIEW,
true);

         setContentView(R.layout.activity_xwalkview);
         mXWalkView = (XWalkView) findViewById(R.id.xwalkview);
         mXWalkView.setUIClient(new MyXWalkUIClient(mXWalkView));
         mXWalkView.setResourceClient(new
MyXWalkResourceClient(mXWalkView));
     }

     @Override
     public void onXWalkReady() {
         // Do anyting with the embedding API
         mXWalkView.load("http://crosswalk-project.org/", null);
     }
 }
```

4.3. XWalkInitializer

`XWalkInitializer` is an alternative to `XWalkActivity` with the difference that it provides a way to initialize the Crosswalk runtime in background silently. Another advantage is that the developer can use their own activity class directly rather than having it extend `XWalkActivity`. However, `XWalkActivity` is still recommended because it makes the code simpler.

If the initialization failed, which means the Crosswalk runtime doesn't exist or doesn't match the app, you could use `XWalkUpdater` to prompt the user to download suitable Crosswalk runtime.

For example:

```
public class MyActivity extends Activity implements
        XWalkInitializer.XWalkInitListener {
    XWalkView mXWalkView;
    XWalkInitializer mXWalkInitializer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Must call initAsync() before anything that involes the
embedding
        // API, including invoking setContentView() with the layout which
        // holds the XWalkView object.

        mXWalkInitializer = new XWalkInitializer(this, this);
        mXWalkInitializer.initAsync();

        // Until onXWalkInitCompleted() is invoked, you should do nothing
        // with the embedding API except the following:
        // 1. Instanciate the XWalkView object
        // 2. Call XWalkPreferences.setValue()
        // 3. Call XWalkView.setUIClient()
        // 4. Call XWalkView.setResourceClient()

        XWalkPreferences.setValue(XWalkPreferences.ANIMATABLE_XWALK_VIEW,
true);

        setContentView(R.layout.activity_xwalkview);
        mXWalkView = (XWalkView) findViewById(R.id.xwalkview);
        mXWalkView.setUIClient(new MyXWalkUIClient(mXWalkView));
```

```
        mXWalkView.setResourceClient(new
MyXWalkResourceClient(mXWalkView));
    }

    @Override
    public void onXWalkInitCompleted() {
        // Do anyting with the embedding API
        mXWalkView.load("http://crosswalk-project.org/", null);
    }

    @Override
    public void onXWalkInitStarted() {
    }

    @Override
    public void onXWalkInitCancelled() {
        // Perform error handling here
    }

    @Override
    public void onXWalkInitFailed() {
        // Perform error handling here, or launch the XWalkUpdater
    }
}
```

4.4. XWalkUpdater

`XWalkUpdater` is a follow-up solution for `XWalkInitializer` in case the initialization has failed. The users of `XWalkActivity` don't need to use this class.

`XWalkUpdater` helps to download the Crosswalk runtime and displays dialogs to interact with the user. By default, it will navigate to the page of Crosswalk runtime on the default application store, subsequent process will be up to the user. If the developer specified the download URL of the Crosswalk runtime, it will launch the download manager to fetch the APK.

After the proper Crosswalk runtime is downloaded and installed, the user will return to current activity from the application store or the installer. The developer should check this point and invoke `XWalkInitializer.initAsync()` again to repeat the initialization process. Please note that from now on, the application will be running in shared mode.

For example:

```
public class MyActivity extends Activity implements
```

```
        XWalkInitializer.XWalkInitListener,
        XWalkUpdater.XWalkUpdateListener {
    XWalkUpdater mXWalkUpdater;

    ......

    @Override
    protected void onResume() {
        super.onResume();

        // Try to initialize again when the user completed updating and
        // returned to current activity. The initAsync() will do nothing
if
        // the initialization has already been completed successfully.
        mXWalkInitializer.initAsync();
    }

    @Override
    public void onXWalkInitFailed() {
        if (mXWalkUpdater == null) {
            mXWalkUpdater = new mXWalkUpdater(this, this);
        }

        // The updater won't be launched if previous update dialog is
        // showing.
        mXWalkUpdater.updateXWalkRuntime();
    }

    @Override
    public void onXWalkUpdateCancelled() {
        // Perform error handling here
    }
 }
```