

Headless JS

Headless JS is a way to run tasks in JavaScript while your app is in the background. It can be used, for example, to sync fresh data, handle push notifications, or play music.

The JS API

A task is an async function that you register on `AppRegistry`, similar to registering React applications:

```
import {AppRegistry} from 'react-native';
AppRegistry.registerHeadlessTask('SomeTaskName', () =>
  require('SomeTaskName'),
);
```

Then, in `SomeTaskName.js`:

```
module.exports = async taskData => {
  // do stuff
};
```

You can do anything in your task such as network requests, timers and so on, as long as it doesn't touch UI. Once your task completes (i.e. the promise is resolved), React Native will go into "paused" mode (unless there are other tasks running, or there is a foreground app).

The Platform API

Yes, this does still require some native code, but it's pretty thin. You need to extend `HeadlessJsTaskService` and override `getTaskConfig`, e.g.:

Java Kotlin

```

package com.your_application_name;

import android.content.Intent;
import android.os.Bundle;
import com.facebook.react.HeadlessJsTaskService;
import com.facebook.react.bridge.Arguments;
import com.facebook.react.jstasks.HeadlessJsTaskConfig;
import javax.annotation.Nullable;

public class MyTaskService extends HeadlessJsTaskService {

    @Override
    protected @Nullable HeadlessJsTaskConfig getTaskConfig(Intent intent) {
        Bundle extras = intent.getExtras();
        if (extras != null) {
            return new HeadlessJsTaskConfig(
                "SomeTaskName",
                Arguments.fromBundle(extras),
                5000, // timeout in milliseconds for the task
                false // optional: defines whether or not the task is allowed in foreground.
                Default is false
            );
        }
        return null;
    }
}

```

Then add the service to your `AndroidManifest.xml` file:

```
<service android:name="com.example.MyTaskService" />
```

Now, whenever you start your service, e.g. as a periodic task or in response to some system event / broadcast, JS will spin up, run your task, then spin down.

Example:

Java **Kotlin**

```

Intent service = new Intent(getApplicationContext(), MyTaskService.class);
Bundle bundle = new Bundle();

```

```
bundle.putString("foo", "bar");
service.putExtras(bundle);

getApplicationContext().startService(service);
```

Retries

By default, the headless JS task will not perform any retries. In order to do so, you need to create a `HeadlessJsRetryPolicy` and throw a specific `Error`.

`LinearCountingRetryPolicy` is an implementation of `HeadlessJsRetryPolicy` that allows you to specify a maximum number of retries with a fixed delay between each attempt. If that does not suit your needs then you can implement your own `HeadlessJsRetryPolicy`. These policies can be passed as an extra argument to the `HeadlessJsTaskConfig` constructor, e.g.

Java Kotlin

```
HeadlessJsRetryPolicy retryPolicy = new LinearCountingRetryPolicy(
    3, // Max number of retry attempts
    1000 // Delay between each retry attempt
);

return new HeadlessJsTaskConfig(
    'SomeTaskName',
    Arguments.fromBundle(extras),
    5000,
    false,
    retryPolicy
);
```

A retry attempt will only be made when a specific `Error` is thrown. Inside a headless JS task, you can import the error and throw it when a retry attempt is required.

Example:

```
import {HeadlessJsTaskError} from 'HeadlessJsTask';

module.exports = async taskData => {
  const condition = ...;
  if (!condition) {
    throw new HeadlessJsTaskError();
  }
};
```

If you wish all errors to cause a retry attempt, you will need to catch them and throw the above error.

Caveats

- The function passed to `setTimeout` does not always behave as expected. Instead the function is called only when the application is launched again. If you only need to wait, use the retry functionality.
- By default, your app will crash if you try to run a task while the app is in the foreground. This is to prevent developers from shooting themselves in the foot by doing a lot of work in a task and slowing the UI. You can pass a fourth `boolean` argument to control this behaviour.
- If you start your service from a `BroadcastReceiver`, make sure to call `HeadlessJsTaskService.acquireWakeLockNow()` before returning from `onReceive()`.

Example Usage

Service can be started from Java API. First you need to decide when the service should be started and implement your solution accordingly. Here is an example that reacts to network connection change.

Following lines shows part of Android manifest file for registering broadcast receiver.

```
<receiver android:name=".NetworkChangeReceiver" >
  <intent-filter>
    <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
  </intent-filter>
</receiver>
```

```

    </intent-filter>
  </receiver>

```

Broadcast receiver then handles intent that was broadcasted in onReceive function. This is a great place to check whether your app is on foreground or not. If app is not on foreground we can prepare our intent to be started, with no information or additional information bundled using putExtra (keep in mind bundle can handle only parcelable values). In the end service is started and wakelock is acquired.

Java **Kotlin**

```

import android.app.ActivityManager;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.Network;
import android.net.NetworkCapabilities;
import android.net.NetworkInfo;
import android.os.Build;

import com.facebook.react.HeadlessJsTaskService;

public class NetworkChangeReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(final Context context, final Intent intent) {
        /**
         * This part will be called every time network connection is changed
         * e.g. Connected -> Not Connected
         */
        if (!isAppOnForeground(context)) {
            /**
             * We will start our service and send extra info about
             * network connections
             */
            boolean hasInternet = isNetworkAvailable(context);
            Intent serviceIntent = new Intent(context, MyTaskService.class);
            serviceIntent.putExtra("hasInternet", hasInternet);
            context.startService(serviceIntent);
            HeadlessJsTaskService.acquireWakeLockNow(context);
        }
    }
}

```

```

private boolean isAppOnForeground(Context context) {
    /**
     * We need to check if app is in foreground otherwise the app will crash.
     * http://stackoverflow.com/questions/8489993/check-android-application-is-in-
foreground-or-not
     */
    ActivityManager activityManager = (ActivityManager)
context.getSystemService(Context.ACTIVITY_SERVICE);
    List<ActivityManager.RunningAppProcessInfo> appProcesses =
        activityManager.getRunningAppProcesses();
    if (appProcesses == null) {
        return false;
    }
    final String packageName = context.getPackageName();
    for (ActivityManager.RunningAppProcessInfo appProcess : appProcesses) {
        if (appProcess.importance ==
            ActivityManager.RunningAppProcessInfo.IMPORTANCE_FOREGROUND &&
            appProcess.processName.equals(packageName)) {
            return true;
        }
    }
    return false;
}

public static boolean isNetworkAvailable(Context context) {
    ConnectivityManager cm = (ConnectivityManager)
        context.getSystemService(Context.CONNECTIVITY_SERVICE);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        Network networkCapabilities = cm.getActiveNetwork();

        if(networkCapabilities == null) {
            return false;
        }

        NetworkCapabilities actNw = cm.getNetworkCapabilities(networkCapabilities);

        if(actNw == null) {
            return false;
        }


        if(actNw.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) ||
actNw.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) ||
actNw.hasTransport(NetworkCapabilities.TRANSPORT_ETHERNET)) {
            return true;
        }
    }
}

```

```
        return false;
    }

    // deprecated in API level 29
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    return (netInfo != null && netInfo.isConnected());
}
}
```

Is this page useful?  

 Edit this page

Last updated on **Jun 21, 2023**