Turbo Native Modules

> **⚠ CAUTION**
>
> This documentation is still **experimental** and details are subject to changes as we iterate. Feel free to share your feedback on the discussion inside the working group for this page.
>
> Moreover, it contains several **manual steps**. Please note that this won't be representative of the final developer experience once the New Architecture is stable. We're working on tools, templates and libraries to help you get started fast on the New Architecture, without having to go through the whole setup.

If you've worked with React Native, you may be familiar with the concept of Native Modules, which allow JavaScript and platform-native code to communicate over the React Native "bridge", which handles cross-platform serialization via JSON.

Turbo Native Modules are the next iteration on Native Modules that provide a few extra benefits:

- Strongly typed interfaces that are consistent across platforms
- The ability to write your code in C++, either exclusively or integrated with another native platform language, reducing the need to duplicate implementations across platforms
- Lazy loading of modules, allowing for faster app startup
- The use of JSI, a JavaScript interface for native code, allows for more efficient communication between native and JavaScript code than the bridge

This guide will show you how to create a basic Turbo Native Module compatible with the latest version of React Native.

> **⚠ CAUTION**

> Turbo Native Modules only work with the **New Architecture** enabled. To migrate to the **New Architecture**, follow the Migration guide

How to Create a Turbo Native Module

To create a Turbo Native Module, we need to:

1. Define the JavaScript specification.
2. Configure the module so that Codegen can generate the scaffolding.
3. Write the native code to finish implementing the module.

1. Folder Setup

In order to keep the module decoupled from the app, it's a good idea to define the module separately from the app and then add it as a dependency to your app later. This is also what you'll do for writing Turbo Native Modules that can be released as open-source libraries later.

Next to your application, create a folder called `RTNCalculator`. **RTN** stands for "**R**eac**t** **N**ative", and is a recommended prefix for React Native modules.

Within `RTNCalculator`, create three subfolders: `js`, `ios`, and `android`.

The final result should look like this:

```
TurboModulesGuide
├── MyApp
└── RTNCalculator
    ├── android
    ├── ios
    └── js
```

2. JavaScript Specification

The **New Architecture** requires interfaces specified in a typed dialect of JavaScript (either Flow or TypeScript). **Codegen** will use these specifications to generate code in strongly-typed languages, including C++, Objective-C++, and Java.

There are two requirements the file containing this specification must meet:

1. The file **must** be named `Native<MODULE_NAME>`, with a `.js` or `.jsx` extension when using Flow, or a `.ts`, or `.tsx` extension when using TypeScript. Codegen will only look for files matching this pattern.
2. The file must export a `TurboModuleRegistrySpec` object.

TypeScript Flow

NativeCalculator.ts

```
import type {TurboModule} from 'react-native/Libraries/TurboModule/RCTExport';
import {TurboModuleRegistry} from 'react-native';

export interface Spec extends TurboModule {
  add(a: number, b: number): Promise<number>;
}

export default TurboModuleRegistry.get<Spec>(
  'RTNCalculator',
) as Spec | null;
```

At the beginning of the spec files are the imports:

- The `TurboModule` type, which defines the base interface for all Turbo Native Modules
- The `TurboModuleRegistry` JavaScript module, which contains functions for loading Turbo Native Modules

The second section of the file contains the interface specification for the Turbo Native Module. In this case, the interface defines the `add` function, which takes two numbers and returns a promise that resolves to a number. This interface type **must** be named `Spec` for a Turbo Native Module.

Finally, we invoke `TurboModuleRegistry.get`, passing the module's name, which will load the Turbo Native Module if it's available.

> **CAUTION**
>
> We are writing JavaScript files importing types from libraries, without setting up a proper node module and installing its dependencies. Your IDE will not be able to resolve the import statements and you may see errors and warnings. This is expected and will not cause problems when you add the module to your app.

3. Module Configuration

Next, you need to add some configuration for **Codegen** and auto-linking.

Some configuration files are shared between iOS and Android, while the others are platform-specific.

Shared

The shared configuration is a `package.json` file used by yarn when installing your module. Create the `package.json` file in the root of the `RTNCalculator` directory.

package.json

```
{
  "name": "rtn-calculator",
  "version": "0.0.1",
  "description": "Add numbers with Turbo Native Modules",
  "react-native": "js/index",
  "source": "js/index",
  "files": [
    "js",
    "android",
    "ios",
    "rtn-calculator.podspec",
    "!android/build",
    "!ios/build",
    "!**/__tests__",
```

```
    "!**/__fixtures__",
    "!**/__mocks__"
  ],
  "keywords": ["react-native", "ios", "android"],
  "repository": "https://github.com/<your_github_handle>/rtn-calculator",
  "author": "<Your Name> <your_email@your_provider.com>
(https://github.com/<your_github_handle>)",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/<your_github_handle>/rtn-calculator/issues"
  },
  "homepage": "https://github.com/<your_github_handle>/rtn-calculator#readme",
  "devDependencies": {},
  "peerDependencies": {
    "react": "*",
    "react-native": "*"
  },
  "codegenConfig": {
    "name": "RTNCalculatorSpec",
    "type": "modules",
    "jsSrcsDir": "js",
    "android": {
      "javaPackageName": "com.rtncalculator"
    }
  }
}
```

The upper part of the file contains some descriptive information like the name of the component, its version, and its source files. Make sure to update the various placeholders which are wrapped in `<>`: replace all the occurrences of the `<your_github_handle>`, `<Your Name>`, and `<your_email@your_provider.com>` tokens.

Then there are the dependencies for this package. For this guide, you need `react` and `react-native`.

Finally, the **Codegen** configuration is specified by the `codegenConfig` field. It contains an array of libraries, each of which is defined by three other fields:

- `name`: The name of the library. By convention, you should add the `Spec` suffix.
- `type`: The type of module contained by this package. In this case, it is a Turbo Native Module; thus, the value to use is `modules`.

- `jsSrcsDir`: the relative path to access the `js` specification that is parsed by **Codegen**.
- `android.javaPackageName`: the package to use in the Java files generated by **Codegen**.

iOS: Create the `podspec` file

For iOS, you'll need to create a `rtn-calculator.podspec` file, which will define the module as a dependency for your app. It will stay in the root of `RTNCalculator`, alongside the `ios` folder.

The file will look like this:

rtn-calculator.podspec

```
require "json"

package = JSON.parse(File.read(File.join(__dir__, "package.json")))

Pod::Spec.new do |s|
  s.name            = "rtn-calculator"
  s.version         = package["version"]
  s.summary         = package["description"]
  s.description     = package["description"]
  s.homepage        = package["homepage"]
  s.license         = package["license"]
  s.platforms       = { :ios => "11.0" }
  s.author          = package["author"]
  s.source          = { :git => package["repository"], :tag => "#{s.version}" }

  s.source_files    = "ios/**/*.{h,m,mm,swift}"

  install_modules_dependencies(s)
end
```

The `.podspec` file has to be a sibling of the `package.json` file, and its name is the one we set in the `package.json`'s `name` property: `rtn-calculator`.

The first part of the file prepares some variables that we use throughout the file. Then, there is a section that contains some information used to configure the pod, like its name, version, and description.

All the requirements for the New Architecture have been encapsulated in the `install_modules_dependencies`. It takes care of installing the proper dependencies based on which architecture is currently enabled. It also automatically installs the `React-Core` dependency in the old architecture.

Android: `build.gradle` and `ReactPackage` class

To prepare Android to run **Codegen** you have to:

1. Update the `build.gradle` file.
2. A Java/Kotlin class that implements the `ReactPackage` interface

At the end of these steps, the `android` folder should look like this:

```
android
├── build.gradle
└── src
    └── main
        └── java
            └── com
                └── rtncalculator
                    └── CalculatorPackage.java
```

The `build.gradle` file

First, create a `build.gradle` file in the `android` folder, with the following contents:

Java Kotlin

build.gradle

```
buildscript {
  ext.safeExtGet = {prop, fallback ->
    rootProject.ext.has(prop) ? rootProject.ext.get(prop) : fallback
  }
  repositories {
    google()
    gradlePluginPortal()
  }
```

```
    dependencies {
      classpath("com.android.tools.build:gradle:7.3.1")
    }
}

apply plugin: 'com.android.library'
apply plugin: 'com.facebook.react'

android {
    compileSdkVersion safeExtGet('compileSdkVersion', 33)
    namespace "com.rtncalculator"
}

repositories {
    mavenCentral()
    google()
}

dependencies {
    implementation 'com.facebook.react:react-native'
}
```

The `ReactPackage` class

Then, you need a class that extends the `TurboReactPackage` interface. To run the **Codegen** process, you don't have to completely implement the package class: an empty implementation is enough for the app to pick up the module as a proper React Native dependency and to try and generate the scaffolding code.

Create an `android/src/main/java/com/rtncalculator` folder and, inside that folder, create a `CalculatorPackage.java` file.

Java Kotlin

CalculatorPackage.java

```
package com.rtncalculator;

import androidx.annotation.Nullable;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
```

```
import com.facebook.react.module.model.ReactModuleInfoProvider;
import com.facebook.react.TurboReactPackage;

import java.util.Collections;
import java.util.List;

public class CalculatorPackage extends TurboReactPackage {

  @Nullable
  @Override
  public NativeModule getModule(String name, ReactApplicationContext reactContext) {
          return null;
  }

  @Override
  public ReactModuleInfoProvider getReactModuleInfoProvider() {
      return null;
  }
}
```

React Native uses the `ReactPackage` interface to understand what native classes the app has to use for the `ViewManager` and `Native Modules` exported by the library.

4. Native Code

For the final step in getting your Turbo Native Module ready to go, you'll need to write some native code to connect the JavaScript side to the native platforms. This process requires two main steps:

- Run **Codegen** to see what it generates.
- Write your native code, implementing the generated interfaces.

When developing a React Native app that uses a Turbo Native Module, it is the responsibility of the app to actually generate the code using **Codegen**. However, when developing a TurboModule as a library, we need to reference the generated code, and it is therefore, useful to see what the app will generate.

As the first step for both iOS and Android, this guide shows how to execute manually the scripts used by **Codegen** to generate the required code. Further information on **Codegen**

can be found here.

> **CAUTION**
>
> The code generated by **Codegen** in this step should not be committed to the versioning system. React Native apps are able to generate the code when the app is built. This allows an app to ensure that all libraries have code generated for the correct version of React Native.

iOS

Generate the code - iOS

To run Codegen for the iOS platform, we need to open a terminal and run the following command:

Running Codegen for iOS

```
cd MyApp
yarn add ../RTNCalculator
cd ..
node MyApp/node_modules/react-native/scripts/generate-codegen-artifacts.js \
  --path MyApp/ \
  --outputPath RTNCalculator/generated/
```

This script first adds the `RTNCalculator` module to the app with `yarn add`. Then, it invokes Codegen via the `generate-codegen-artifacts.js` script.

The `--path` option specifies the path to the app, while the `--outputPath` option tells Codegen where to output the generated code.

The output of this process is the following folder structure:

```
generated
└── build
    └── generated
        └── ios
```

```
├── FBReactNativeSpec
│   ├── FBReactNativeSpec-generated.mm
│   └── FBReactNativeSpec.h
├── RCTThirdPartyFabricComponentsProvider.h
├── RCTThirdPartyFabricComponentsProvider.mm
├── RTNCalculatorSpec
│   ├── RTNCalculatorSpec-generated.mm
│   └── RTNCalculatorSpec.h
└── react
    └── renderer
        └── components
            └── rncore
                ├── ComponentDescriptors.h
                ├── EventEmitters.cpp
                ├── EventEmitters.h
                ├── Props.cpp
                ├── Props.h
                ├── RCTComponentViewHelpers.h
                ├── ShadowNodes.cpp
                └── ShadowNodes.h
```

The relevant path for the Turbo Native Module interface is `generated/build/generated/ios/RTNCalculatorSpec`.

See the Codegen section for further details on the generated files.

> **NOTE**
>
> When generating the scaffolding code using **Codegen**, iOS does not clean the `build` folder automatically. If you changed the Spec name, for example, and then run **Codegen** again, the old files would be retained. If that happens, remember to remove the `build` folder before running the **Codegen** again.
>
> ```
> cd MyApp/ios
> rm -rf build
> ```

Write the Native iOS Code

Now add the Native code for your Turbo Native Module. Create two files in the `RTNCalculator/ios` folder:

1. The `RTNCalculator.h`, a header file for the module.
2. The `RTNCalculator.mm`, the implementation of the module.

RTNCalculator.h

RTNCalculator.h

```
#import <RTNCalculatorSpec/RTNCalculatorSpec.h>

NS_ASSUME_NONNULL_BEGIN

@interface RTNCalculator : NSObject <NativeCalculatorSpec>

@end

NS_ASSUME_NONNULL_END
```

This file defines the interface for the `RTNCalculator` module. Here, we can add any native method we may want to invoke on the view. For this guide, we don't need anything, therefore the interface is empty.

RTNCalculator.mm

RTNCalculator.mm

```
#import "RTNCalculatorSpec.h"
#import "RTNCalculator.h"

@implementation RTNCalculator

RCT_EXPORT_MODULE()

- (void)add:(double)a b:(double)b resolve:(RCTPromiseResolveBlock)resolve reject:
(RCTPromiseRejectBlock)reject {
    NSNumber *result = [[NSNumber alloc] initWithInteger:a+b];
    resolve(result);
}

- (std::shared_ptr<facebook::react::TurboModule>)getTurboModule:
    (const facebook::react::ObjCTurboModule::InitParams &)params
{
```

```
    return std::make_shared<facebook::react::NativeCalculatorSpecJSI>(params);
}

@end
```

The most important call is to the `RCT_EXPORT_MODULE`, which is required to export the module so that React Native can load the Turbo Native Module.

Then the `add` method, whose signature must match the one specified by the Codegen in the `RTNCalculatorSpec.h`.

Finally, the `getTurboModule` method gets an instance of the Turbo Native Module so that the JavaScript side can invoke its methods. The function is defined in (and requested by) the `RTNCalculatorSpec.h` file that was generated earlier by Codegen.

> **INFO**
>
> There are other macros that can be used to export modules and methods. You view the code that specifies them here.

Android

Android follows similar steps to iOS. We have to generate the code for Android, and then we have to write some native code to make it work.

Generate the Code - Android

To generate the code for Android, we need to manually invoke Codegen. This is done similarly to what we did for iOS: first, we need to add the package to the app, and then we need to invoke a script.

Running Codegen for Android

```
cd MyApp
yarn add ../RTNCalculator
cd android
./gradlew generateCodegenArtifactsFromSchema
```

This script first adds the package to the app, in the same way iOS does. Then, after moving to the `android` folder, it invokes a Gradle task to create the generated code.

> **NOTE**
>
> To run **Codegen**, you need to enable the **New Architecture** in the Android app. This can be done by opening the `gradle.properties` files and by switching the `newArchEnabled` property from `false` to `true`.

The generated code is stored in the `MyApp/node_modules/rtn-calculator/android/build/generated/source/codegen` folder and it has this structure:

```
codegen
├── java
│   └── com
│       └── rtncalculator
│           └── NativeCalculatorSpec.java
├── jni
│   ├── Android.mk
│   ├── CMakeLists.txt
│   ├── RTNCalculator-generated.cpp
│   ├── RTNCalculator.h
│   └── react
│       └── renderer
│           └── components
│               └── RTNCalculator
│                   ├── ComponentDescriptors.h
│                   ├── EventEmitters.cpp
│                   ├── EventEmitters.h
│                   ├── Props.cpp
│                   ├── Props.h
│                   ├── ShadowNodes.cpp
│                   └── ShadowNodes.h
└── schema.json
```

Write the Native Android Code

The native code for the Android side of a Turbo Native Module requires:

1. to create a `CalculatorModule.java` that implements the module.

2. to update the `CalculatorPackage.java` created in the previous step.

The final structure within the Android library should look like this:

```
android
├── build.gradle
└── src
    └── main
        └── java
            └── com
                └── rtncalculator
                    ├── CalculatorModule.java
                    └── CalculatorPackage.java
```

Creating the `CalculatorModule.java`

Java Kotlin

CalculatorModule.java

```java
package com.rtncalculator;

import androidx.annotation.NonNull;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.Promise;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;
import java.util.Map;
import java.util.HashMap;
import com.rtncalculator.NativeCalculatorSpec;

public class CalculatorModule extends NativeCalculatorSpec {

    public static String NAME = "RTNCalculator";

    CalculatorModule(ReactApplicationContext context) {
        super(context);
    }

    @Override
```

```
    @NonNull
    public String getName() {
        return NAME;
    }

    @Override
    public void add(double a, double b, Promise promise) {
        promise.resolve(a + b);
    }
}
```

This class implements the module itself, which extends the `NativeCalculatorSpec` that was generated from the `NativeCalculator` JavaScript specification file.

Updating the `CalculatorPackage.java`

Java Kotlin

CalculatorPackage.java

```
package com.rtncalculator;

import androidx.annotation.Nullable;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
+ import com.facebook.react.module.model.ReactModuleInfo;
import com.facebook.react.module.model.ReactModuleInfoProvider;
import com.facebook.react.TurboReactPackage;

import java.util.Collections;
import java.util.List;
+ import java.util.HashMap;
+ import java.util.Map;

public class CalculatorPackage extends TurboReactPackage {

  @Nullable
  @Override
  public NativeModule getModule(String name, ReactApplicationContext reactContext) {
+     if (name.equals(CalculatorModule.NAME)) {
+         return new CalculatorModule(reactContext);
+     } else {
          return null;
```

```
+       }
    }


    @Override
    public ReactModuleInfoProvider getReactModuleInfoProvider() {
-       return null;
+       return () -> {
+           final Map<String, ReactModuleInfo> moduleInfos = new HashMap<>();
+           moduleInfos.put(
+                   CalculatorModule.NAME,
+                   new ReactModuleInfo(
+                           CalculatorModule.NAME,
+                           CalculatorModule.NAME,
+                           false, // canOverrideExistingModule
+                           false, // needsEagerInit
+                           true, // hasConstants
+                           false, // isCxxModule
+                           true // isTurboModule
+           ));
+           return moduleInfos;
+       };
    }
}
```

This is the last piece of Native Code for Android. It defines the `TurboReactPackage` object that will be used by the app to load the module.

Final structure

The final structure should look like this:

```
TurboModulesGuide
├── MyApp
└── RTNCalculator
    ├── android
    │   ├── build.gradle
    │   └── src
    │       └── main
    │           └── java
    │               └── com
    │                   └── rtncalculator
    │                       ├── CalculatorPackage.java
```

```
│                    └── CalculatorModule.java
├── generated
├── ios
│   ├── RTNCalculator.h
│   └── RTNCalculator.mm
├── js
│   └── NativeCalculator.js
├── package.json
└── rtn-calculator.podspec
```

5. Adding the Turbo Native Module to your App

Now you can install and use the Turbo Native Module in your app.

Shared

First of all, we need to add the NPM package which contains the Component to the app. This can be done with the following command:

```
cd MyApp
yarn add ../RTNCalculator
```

This command will add the `RTNCalculator` module to the `node_modules` of your app.

iOS

Then, you need to install the new dependencies in your iOS project. To do so, run these commands:

```
cd ios
RCT_NEW_ARCH_ENABLED=1 bundle exec pod install
```

This command will look for all the dependencies of the project and it will install the iOS ones. The `RCT_NEW_ARCH_ENABLED=1` instruct **CocoaPods** that it has to run some additional operations to run **Codegen**.

> **NOTE**
>
> You may have to run `bundle install` once before you can use `RCT_NEW_ARCH_ENABLED=1 bundle exec pod install`. You won't need to run `bundle install` anymore, unless you need to change the Ruby dependencies.

Android

Android configuration requires to enable the **New Architecture**:

1. Open the `android/gradle.properties` file
2. Scroll down to the end of the file and switch the `newArchEnabled` property from `false` to `true`.

JavaScript

Now you can use your Turbo Native Module calculator in your app!

Here's an example App.js file using the `add` method:

TypeScript Flow

App.tsx

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 */
import React from 'react';
import {useState} from 'react';
import {
  SafeAreaView,
  StatusBar,
  Text,
  Button,
} from 'react-native';
import RTNCalculator from 'rtn-calculator/js/NativeCalculator';
```

```
const App: () => JSX.Element = () => {
  const [result, setResult] = useState<number | null>(null);
  return (
    <SafeAreaView>
      <StatusBar barStyle={'dark-content'} />
      <Text style={{marginLeft: 20, marginTop: 20}}>
        3+7={result ?? '??'}
      </Text>
      <Button
        title="Compute"
        onPress={async () => {
          const value = await RTNCalculator?.add(3, 7);
          setResult(value ?? null);
        }}
      />
    </SafeAreaView>
  );
};
export default App;
```

Try this out to see your Turbo Native Module in action!

Is this page useful? 

Edit this page

*Last updated on **Aug 24, 2023***