

Foodie App Documentation

1. Introduction

Foodie is a revolutionary food delivery app designed to bring a variety of delicious cuisines from multiple restaurants straight to your doorstep. Built with Laravel & Flutter. Foodie offers a seamless and intuitive user experience, making it easier than ever to discover, order, and enjoy your favourite meals from local eateries.

Key Features:

1. **Wide Selection of Restaurants:** Explore a vast array of restaurants, from local favourites to popular chains, all available at your fingertips.
2. **User-Friendly Interface:** Enjoy a sleek and intuitive design that makes browsing menus, placing orders, and tracking deliveries effortless.
3. **Real-Time Order Tracking:** Stay updated with real-time tracking of your order from preparation to delivery, ensuring you know exactly when your food will arrive.
4. **Customizable Orders:** Personalise your meals with special instructions and dietary preferences to suit your taste and dietary needs.
5. **Secure Payment Options:** Choose from multiple secure payment methods, including credit/debit cards, digital wallets, and cash on delivery.
6. **Exclusive Deals and Discounts:** Take advantage of exclusive offers, promotions, and discounts available only through the Foodie app.
7. **Easy Reordering:** Quickly reorder your favourite meals with just a few taps, thanks to a user-friendly reorder feature.
8. **Ratings and Reviews:** Make informed decisions with access to detailed ratings and reviews from other Foodie users.
9. **24/7 Customer Support:** Receive dedicated support anytime you need assistance, ensuring a smooth and satisfying food delivery experience.
10. **Multi-Platform Accessibility:** Enjoy Foodie on both Android and iOS devices, with a consistent and optimised performance across platforms.

Why Choose Foodie?

Foodie stands out as a top-tier food delivery app by combining an extensive restaurant selection with a user-centric design, reliable delivery service, and secure payment options. Whether you're craving a gourmet meal, a quick bite, or something in between, Foodie makes it easy to satisfy your hunger with convenience and style.

2. Setup Setting up Flutter involves several steps. Here's a basic guide to get you started:

2.1. System Requirements:

Ensure your system meets the minimum requirements for Flutter development. Flutter supports development on Windows, macOS, and Linux. Make sure your system has the necessary hardware and software requirements as specified by Flutter documentation.

2.2. Install Flutter SDK:

Download the Flutter SDK from the [official Flutter website](#).

Download the Flutter SDK Version of 3.22.3.

Extract the downloaded file to a location on your system. For example, on macOS or Linux, you can extract it to `/usr/local` and on Windows to `C:\`.

Add the Flutter `bin` directory to your system PATH to run Flutter commands from the command line. This step is important for executing Flutter commands globally in your system.

2.3. Install Development Tools:

For Android: Install Android Studio and configure the Flutter plugin.

Android Studio provides the Android SDK, which Flutter uses to develop for Android. Ensure you have the Android SDK and the necessary tools installed through Android Studio.

For iOS: You need a macOS system with Xcode installed to develop and deploy Flutter apps for iOS.

2.4. Set up Android Emulator or iOS Simulator:

For Android development, set up an Android Virtual Device (AVD) using Android Studio's AVD Manager.

For iOS development, use the iOS Simulator provided by Xcode.

Run `flutter doctor`:

Open a terminal and run `flutter doctor`. This command checks your system for any dependencies needed for Flutter development.

It will provide feedback on any missing or outdated components and instructions on how to resolve them.

2.5. Install Flutter and Dart plugins for your preferred IDE:

If you're using VS Code, install the Flutter and Dart plugins to enhance your development experience.

Android Studio also has built-in support for Flutter, but make sure to install the Flutter plugin if it's not already installed.

2.6. Create your first Flutter project:

Use the `flutter create` command to create a new Flutter project.

Navigate to the project directory and explore the file structure. You'll find the main Dart file (`main.dart`) in the `lib` directory, where you'll write your Flutter code.

2.7. Run your Flutter app:

Connect a device or start an emulator/simulator.

Navigate to your Flutter project directory in the terminal and run `flutter run`.

This will compile your Flutter app and launch it on the connected device or emulator.

2.8. Start Developing:

Once your app is running, you can start developing your Flutter UI and logic. Flutter's hot reload feature allows you to see your changes instantly without restarting the app.

2.9. Learn and Explore:

Flutter has an extensive set of widgets and libraries. Explore the official Flutter documentation and community resources to learn more about Flutter development.

3. Changing the package name (also known as the bundle identifier or application ID) in a Flutter project involves a few steps. Here's how you can do it:

3.1. Change the Android package name:

Navigate to the `android` directory within your Flutter project.

Open the `AndroidManifest.xml` file located in the `app/src/main` directory.

Find the `package` attribute in the `<manifest>` tag and change its value to your desired package name.

3.2. Change the iOS bundle identifier:

Navigate to the `ios` directory within your Flutter project.

Open the `Runner.xcodeproj` project file using Xcode.

In Xcode, select the `Runner` project from the project navigator.

Go to the `Runner` target's settings.

Under the `General` tab, find the `Bundle Identifier` field and change it to your desired bundle identifier.

3.3. Update Flutter project configuration:

Open the `pubspec.yaml` file located in the root directory of your Flutter project.

Update the `name` field with your new package name.

Update the `android: package` field under `flutter:` with your new package name.

3.4. Update Android source code references:

Open the `MainActivity.java` file located in `android/app/src/main/java/com/your_old_package_name`.

Replace occurrences of the old package name with the new package name.

3.5. Update iOS source code references:

Open the `AppDelegate.swift` file located in `ios/Runner`.

Replace occurrences of the old bundle identifier with the new bundle identifier.

3.6. Clean and rebuild the project:

After making the necessary changes, clean and rebuild the project.

In Android Studio, you can clean the project by selecting `Build > Clean Project`.

In Xcode, you can clean the project by selecting `Product > Clean Build Folder`.

3.7. Test your changes:

Run your Flutter app on both Android and iOS devices/emulators to ensure that the changes have been applied successfully.

Verify that the app runs without any issues and that the new package name/bundle identifier is reflected correctly.

4. To change the launcher icon (app icon) in a Flutter project, you can follow these steps:

4.1. Prepare Your New Icons:

First, prepare the new launcher icon images in the required sizes. Android and iOS have different size requirements. You typically need icons in various sizes to support different screen densities.

Android:

For Android, you need to replace the existing launcher icon files with your new ones. The launcher icons for Android are stored in the `mipmap` folders inside the `android/app/src/main/res` directory.

Replace the existing icon files (`ic_launcher.png`) in the `mipmap` folders with your new icon files. Make sure to maintain the same file names and sizes.

You may need to replace icons in various drawable folders for different screen densities (e.g., `mipmap-hdpi`, `mipmap-mdpi`, `mipmap-xhdpi`, `mipmap-xxhdpi`, `mipmap-xxxhdpi`).

iOS:

For iOS, you need to replace the existing icon files with your new ones. The icons for iOS are stored in the `Assets.xcassets` directory in the `ios/Runner` directory.

Open your Flutter project in Xcode by navigating to the `ios` directory and opening the `.xcworkspace` file with Xcode.

In Xcode, navigate to `Runner > Assets.xcassets`.

Replace the existing `AppIcon` with your new icon files. You'll typically find different sizes labelled for various devices (e.g., iPhone, iPad).

4.2. Flutter Launcher Icon Package (Optional):

Alternatively, you can use the `flutter_launcher_icons` package to automate the process of updating launcher icons. This package allows

you to define a single source image and generate the required icons for both Android and iOS.

Install the `flutter_launcher_icons` package by adding it to your `pubspec.yaml` file:

```
dev_dependencies:  
  flutter_launcher_icons: "^0.9.2"
```

Run the following command in your terminal to generate launcher icons based on your configuration:

```
flutter pub get  
flutter pub run flutter_launcher_icons:main
```

Follow the prompts to configure the package according to your project's requirements.

4.3. Test Your Changes:

After replacing the icon files or running the `flutter_launcher_icons` package, rebuild your Flutter project and run it on both Android and iOS devices/emulators to ensure that the new launcher icon is displayed correctly.

5. To change the app name in a Flutter project for Android, iOS, and web platforms, you'll need to adjust settings in each platform's configuration. Here's how you can do it:

5.1. For Android:

- 5.1.1. Open the `android/app/src/main/AndroidManifest.xml` file.
- 5.1.2. Locate the `<application>` tag.
- 5.1.3. Change the value of the `android:label` attribute to your desired app name.

5.2. For iOS:

- 5.2.1. Open the `ios/Runner/Info.plist` file.
- 5.2.2. Locate the `<key>CFBundleDisplayName</key>` entry.
- 5.2.3. Change the value associated with `<string>` to your desired app name.

5.3. For Web:

- 5.3.1. Open the `web/index.html` file.
- 5.3.2. Locate the `<title>` tag.
- 5.3.3. Change the text within the `<title>` tag to your desired app name.

5.4. Additionally:

In your Flutter project directory, open the `pubspec.yaml` file and ensure that the `name` field is set to the desired app name. This is the name that appears in the app store listings, on the device's home screen, etc.

After making these changes, rebuild your app for each platform to apply the new app name.

5.5. Example:

Let's say you want to change the app name to "My New App":

5.5.1. Android:

```
<application
  android:label="My New App"
  ...>
```

5.5.2. **iOS:**

```
<key>CFBundleDisplayName</key>
<string>My New App</string>
```

5.5.3. **Web:**

```
<title>My New App</title>
```

5.5.4. **Pubspec.yaml:**

```
name: my_new_app
```

6. To change the Google Maps API key in a Flutter app for both Android and iOS, you need to update the respective configuration files in each platform. Here are the steps:

6.1. For Android:

- 6.1.1. Navigate to your Flutter project's android/app/src/main/AndroidManifest.xml file.
- 6.1.2. Inside the <application> element, locate the <meta-data> tag with the name com.google.android.geo.API_KEY.
- 6.1.3. Replace the android:value attribute with your new Google Maps API key.

```
<meta-data
```

```
android:name="com.google.android.geo.API_KEY"  
android:value="YOUR_API_KEY_HERE"/>
```

6.2. For iOS:

- 6.2.1. Open your Flutter project in a text editor or IDE.
- 6.2.2. Navigate to the ios/Runner directory within your Flutter project.
- 6.2.3. Locate the AppDelegate.swift file.
- 6.2.4. In the didFinishLaunchingWithOptions method, set the Google Maps API key.

```
import UIKit  
import Flutter  
import GoogleMaps // Import GoogleMaps framework  
  
@UIApplicationMain  
@objc class AppDelegate: FlutterAppDelegate {  
    override func application(  
        _ application: UIApplication,  
        didFinishLaunchingWithOptions launchOptions:  
[UIApplication.LaunchOptionsKey: Any]?  
    ) -> Bool {  
        // Add your Google Maps API key here  
        GMSServices.provideAPIKey("YOUR_API_KEY_HERE")  
  
        // Rest of your code...  
  
        return super.application(application,  
didFinishLaunchingWithOptions: launchOptions)  
    }  
}
```

Replace "YOUR_API_KEY_HERE" with your actual Google Maps API key.

After updating the API key in the AppDelegate.swift file, rebuild your Flutter app for iOS to apply the changes.

By following these steps, your Flutter app will use the new Google Maps API key specifically for iOS. Make sure you've also updated the API key in the Android configuration as described earlier if you want to apply changes to both platforms.

7. Two Ways to set up a firebase with an application(Using Firebase CLI (5.1) OR Manually(5.2)).

7.1. Setting up Firebase for a Flutter project using the Firebase CLI involves several steps. Below is a detailed guide on how to do it:

7.1.1. Install Firebase CLI:

First, you need to install the Firebase CLI if you haven't already. You can install it via npm (Node Package Manager) by running the following command in your terminal or command prompt:

```
npm install -g firebase-tools
```

7.1.2. Login to Firebase:

After installing the Firebase CLI, log in to your Firebase account by running the following command:

```
firebase login
```

This will open a browser window prompting you to log in to your Google account associated with Firebase.

7.1.3. Create a Firebase Project:

If you haven't already created a Firebase project, you can create one using the Firebase CLI by running:

```
firebase projects:create
```

Follow the prompts to create a new project.

7.1.4. Initialize Firebase in Your Flutter Project:

Navigate to your Flutter project directory in the terminal.

Initialize Firebase in your project by running:

```
firebase init
```

This command will prompt you to select the Firebase features you want to set up. Choose the Firebase services you intend to use, such as Authentication, Firestore, etc.

7.1.5. Configure Firebase:

During the initialization process (`firebase init`), you'll be asked to select the Firebase project you created or configured earlier. Choose the appropriate project from the list.

7.1.6. Install Required Dependencies:

After selecting the Firebase features, Firebase CLI will generate the necessary configuration files for your project.

Next, you need to install the required Firebase packages in your Flutter project. You can do this by adding the dependencies to your `pubspec.yaml` file:

```
dependencies:
  firebase_core: latest_version
  firebase_auth: latest_version # (if you need
  Firebase Authentication)
  cloud_firestore: latest_version # (if you need
  Cloud Firestore)
  # Add other Firebase plugins as needed
```

Replace `latest_version` with the actual version numbers of the Firebase plugins you want to use. You can find the latest versions on pub.dev.

7.1.7. Initialize Firebase in Your Flutter App:

```
import 'package:firebase_core/firebase_core.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

7.1.8. Using Firebase Services:

Now you can use Firebase services in your Flutter app. Import the necessary Firebase packages and follow the respective documentation for each service you want to use (e.g., Firebase Authentication, Cloud Firestore).

7.1.9. Testing:

Run your Flutter app on a device or emulator to verify that Firebase services are correctly integrated and working.

7.2. Configuring and setting up FlutterFire in your Flutter project involves several steps. Here's a detailed guide on how to do it:

7.2.1. Create a Firebase Project:

Go to the Firebase Console: <https://console.firebase.google.com/>

Click on "Add Project" and follow the instructions to create a new project.

Once your project is created, you'll be redirected to the project dashboard.

7.2.2. Add an App to Your Firebase Project:

In the Firebase console, select your project.

Click on the "Add app" button (usually represented by an Android or iOS icon).

Follow the setup instructions to register your app with Firebase.

Download the `google-services.json` file for Android or `GoogleService-Info.plist` file for iOS. These files contain your Firebase project configuration details.

7.2.3. To update the DefaultFirebaseOptions in your Flutter project based on the `google-services.json` file, follow these steps:

- 7.2.3.1. Open your `lib/firebase_options.dart` file.
- 7.2.3.2. Extract the necessary keys from your `google-services.json` file, especially fields like `apiKey`, `projectId`, `appId`, etc.
- 7.2.3.3. Modify the `DefaultFirebaseOptions` class with the appropriate configuration for each platform.

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/foundation.dart';

class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return const FirebaseOptions(
        apiKey: 'your-web-apiKey',
        appId: 'your-web-appId',
        messagingSenderId: 'your-web-senderId',
        projectId: 'your-web-projectId',
        authDomain: 'your-web-authDomain',
        storageBucket: 'your-web-storageBucket',
        measurementId: 'your-web-measurementId',
      );
    } else if (defaultTargetPlatform == TargetPlatform.iOS
    || defaultTargetPlatform == TargetPlatform.macOS) {
      return const FirebaseOptions(
        apiKey: 'your-ios-apiKey',
        appId: 'your-ios-appId',
```



```

        messagingSenderId: 'your-ios-senderId',
        projectId: 'your-ios-projectId',
        storageBucket: 'your-ios-storageBucket',
        iosClientId: 'your-ios-clientId',
        iosBundleId: 'your-ios-bundleId',
    );
} else {
    return const FirebaseOptions(
        apiKey: 'your-android-apiKey',
        appId: 'your-android-appId',
        messagingSenderId: 'your-android-senderId',
        projectId: 'your-android-projectId',
        storageBucket: 'your-android-storageBucket',
    );
}
}
}

```

7.2.4. Flutter Project Setup:

Open your Flutter project in your preferred editor.

Navigate to the `android/app` directory for Android or `ios/Runner` directory for iOS, and paste the `google-services.json` (for Android) or `GoogleService-Info.plist` (for iOS) file that you downloaded earlier.

7.2.5. Add Firebase SDK to Your Flutter Project:

Open your Flutter project's `pubspec.yaml` file.

Add the following dependencies:

```
dependencies:  
  flutter:  
    sdk: flutter  
  firebase_core: latest_version  
  firebase_auth: latest_version # (if you need  
Firebase Authentication)  
  cloud_firestore: latest_version # (if you need  
Cloud Firestore)  
  # Add other Firebase plugins as needed
```

Replace `latest_version` with the latest version numbers of the Firebase plugins you want to use. You can find the latest versions on pub.dev.

7.2.6. Using Firebase Services:

You can now use Firebase services in your Flutter app. Import the necessary Firebase packages and follow the respective documentation for each service you want to use (e.g., Firebase Authentication, Cloud Firestore).

7.2.7. Testing:

Run your Flutter app on a device or emulator to verify that Firebase services are correctly integrated and working.

8. To generate SHA-1 and SHA-256 keys for your Flutter app and add them to your Firebase project, you can follow these steps:

8.1. Using Keytool:

- 8.1.1. Navigate to your JDK's `bin` directory in the command line.
- 8.1.2. Execute the following command to generate the SHA-1 key:

```
keytool -list -v -keystore path-to-your-keystore-file  
-alias your-alias-name
```

- 8.1.3. Replace `path-to-your-keystore-file` with the path to your keystore file (usually `debug.keystore` for debug builds and a custom keystore for release builds) and `your-alias-name` with the alias name used to generate the keystore.
- 8.1.4. Similarly, execute the following command to generate the SHA-256 key:

```
keytool -list -v -keystore path-to-your-keystore-file  
-alias your-alias-name -storetype PKCS12 -keyalg RSA
```

8.2. Using Gradle (Android Studio):

- 8.2.1. Open your Flutter project in Android Studio.
- 8.2.2. In the Android view, navigate to `app -> Gradle Scripts -> build.gradle (Module: app)`.
- 8.2.3. Add the following lines to the `android` block

```
android {  
    ...  
    signingConfigs {  
        debug {  
            storeFile file('path-to-debug.keystore')  
            storePassword 'password'  
            keyAlias 'key-alias'  
            keyPassword 'password'  
        }  
    }  
}
```

- 8.2.4. Replace `path-to-debug.keystore`, `password`, and `key-alias` with your keystore file path, password, and key alias respectively.
- 8.2.5. Sync your project to apply changes.
- 8.2.6. Run the following Gradle task in the terminal to get the SHA-1 and SHA-256 keys

```
./gradlew signingReport
```

8.3. Adding Keys to Firebase Console:

8.3.1. Go to the [Firebase Console](#).

8.3.2. Select your project.

8.3.3. For Android:

8.3.3.1. Click on the Android icon to add an Android app.

8.3.3.2. Follow the setup instructions, including adding the `google-services.json` file to your Flutter project's `android/app` directory.

8.3.3.3. During setup, you'll be asked for your SHA-1 key. Paste the SHA-1 key you generated earlier.

8.3.3.4. After setup is complete, you'll have the option to download the `google-services.json` file again. Replace the existing file in your project if needed.

8.3.4. For iOS:

8.3.4.1. Click on the iOS icon to add an iOS app.

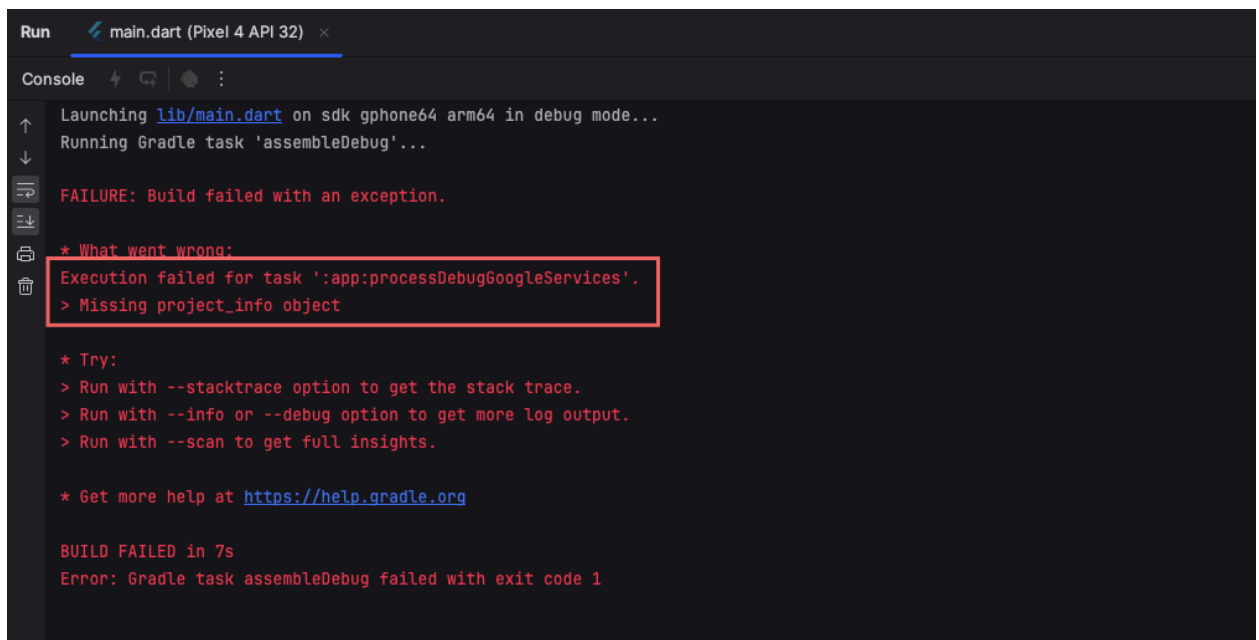
8.3.4.2. Follow the setup instructions, including downloading the `GoogleService-Info.plist` file.

8.3.4.3. There's no need to add SHA-1 or SHA-256 keys for iOS.

8.3.4.4. Add the downloaded `GoogleService-Info.plist` file to your Flutter project's `ios/Runner` directory.

9. FAQ:

9.1. What should I do if I get the error message "Missing project_info object" in my Flutter project?



```
Run main.dart (Pixel 4 API 32) x
Console
Lauching lib/main.dart on sdk gphone64 arm64 in debug mode...
Running Gradle task 'assembleDebug'...
FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app:processDebugGoogleServices'.
> Missing project_info object

* Try:
> Run with --stacktrace option to get the stack trace.
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 7s
Error: Gradle task assembleDebug failed with exit code 1
```

The error message you're encountering, "Missing project_info object," typically arises in Flutter projects when there's a misconfiguration or missing information related to Firebase services. Here's how you can troubleshoot and potentially resolve this issue

Solution :

- 9.1.1. Check `google-services.json`: If you're using Firebase services in your Flutter app, ensure that you have the `google-services.json` file placed in the correct location within your Android project. This file contains important configuration information for Firebase services. It should be located in the `android/app` directory of your Flutter project.
- 9.1.2. Verify Firebase Setup: Double-check that you've completed all the necessary steps to set up Firebase for your Flutter project. This includes adding your app to the Firebase project using the Firebase Console, downloading and adding the `google-services.json` file to your Android project, and configuring Firebase services in your Flutter app's code (if applicable).
- 9.1.3. Update Google Services Plugin: Ensure that you're using the latest version of the Google Services Gradle plugin in your Android project. You can check the latest version on the Google Services Gradle Plugin page and update the version number in your `android/build.gradle` file accordingly.
- 9.1.4. Check Flutter and Firebase Versions Compatibility: Make sure that you're using compatible versions of Flutter and Firebase. Sometimes, using outdated versions of either can lead to compatibility issues. Refer to the Firebase Flutter documentation and ensure compatibility with your Flutter version.
- 9.1.5. Sync Gradle Files: After making any changes to your Android project configuration, such as updating the `google-services.json` file or modifying the Gradle build files, make sure to sync your Gradle files in Android Studio. This can be done by clicking on the "Sync Project with Gradle Files" button.
- 9.1.6. Rebuild the Project: After ensuring all configurations are correct, try rebuilding your Flutter project. You can do this by running `flutter clean` followed by `flutter pub get` and then rebuilding the project (`flutter run` or `flutter build`).

- 9.1.7. Check for Error Logs: Look for more detailed error messages or stack traces in the console output or build logs. These can provide additional clues about the underlying issue causing the "Missing project_info object" error.

Thank You

© 2022-2024 eMart. All Rights Reserved.