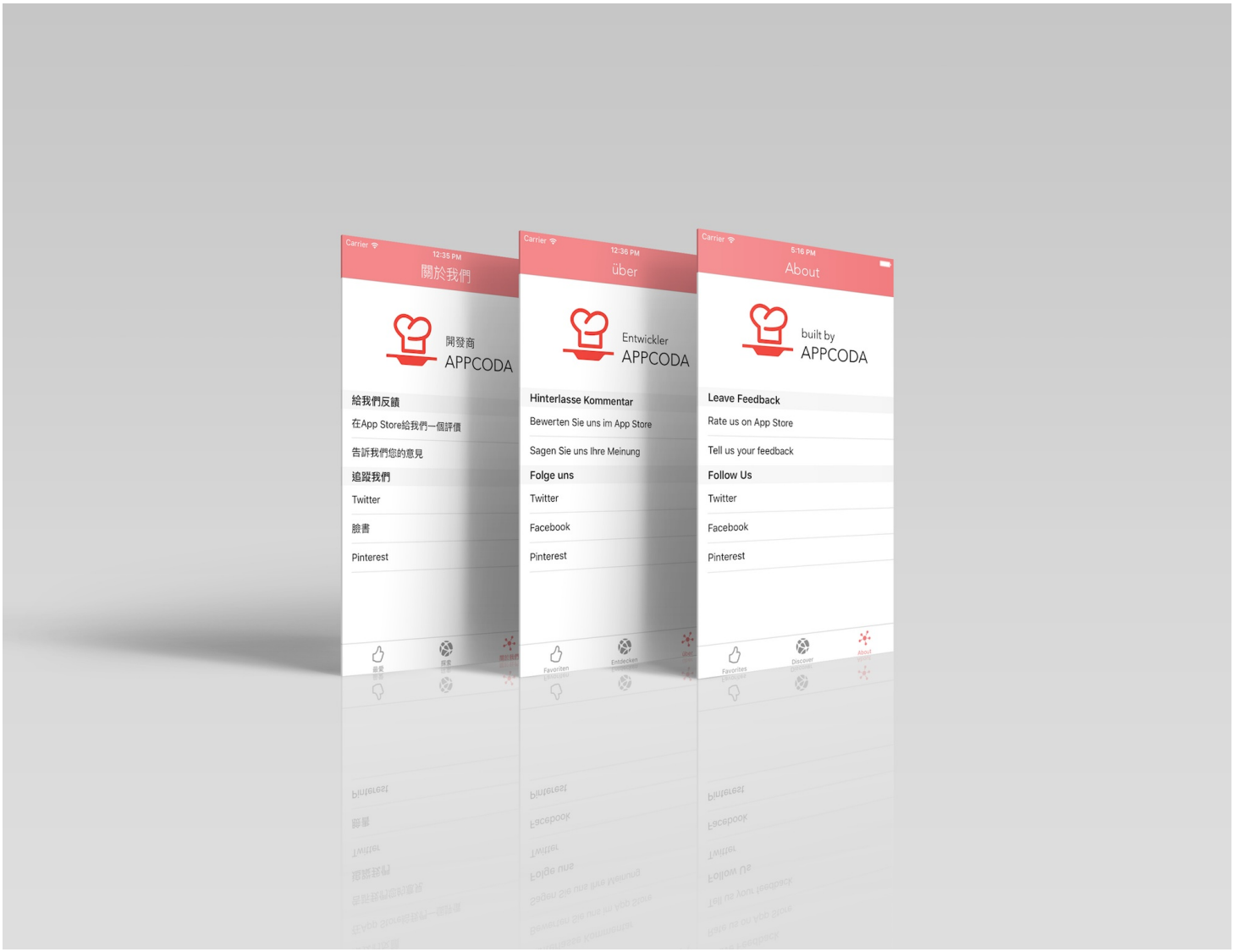
Chapter 25
Localizing Your App to Reach More Users



> Good code is its own best documentation. As you're about to add a comment, ask yourself, "How can I improve the code so that this comment isn't needed?" Improve the code and then document it to make it even clearer.
>
> - Steve McConnell

In this chapter, let's talk about localization. The iOS devices including iPhone and iPad are available globally. The App Store is available in 150 countries around the world. Your users are from different countries and speak different languages. To deliver a great user experience and reach a global audience, you would want to make your app available in multiple languages. The process of adapting an app to support a particular language is usually known as *localization*.

Xcode has the built-in support for localization. It's fairly easy for developers to localize an app through the localization feature and a few API calls. You may have heard of localization and internationalization. You may think that both terms refer to the process of translation; that's partially correct. In iOS development, internationalization is considered a milestone of building a localized app. Before your app can be adapted to different languages, you design and structure the app to be language and region independent. This process is known as *internationalization*. For instance, your app displays a price field. As you may know, some countries use a dot to indicate decimal place (e.g. $1000.50), while many other countries use a comma instead (e.g. $1000,50). The internationalization process involves designing the price field so that it can be adapted to different regions.

Localization is the process of adapting an internationalized app for different languages and regions. This involves translating static and visible text to a specific language and adding country-specific elements such as images, videos and sounds.

In this chapter, we'll localize the FoodPin app into Chinese and German. However, don't expect me to translate all the text in the app - I just want to show you the overall process of localization using Xcode.

Internationalizing Your App

The first step of building a localized app is *internationalization*. In this section, we will modify the app so that it can be easily adapted to various languages.

First, let's talk about the user-facing text of the app. There is tons of user-facing text in the source code. For convenience, we simply specify the strings in the source code. Figure 25-1 displays a few user-facing text in the `RestaurantTableViewController` class. When it comes to localization, these hardcoded strings are not localizable. We have to internationalize them first.

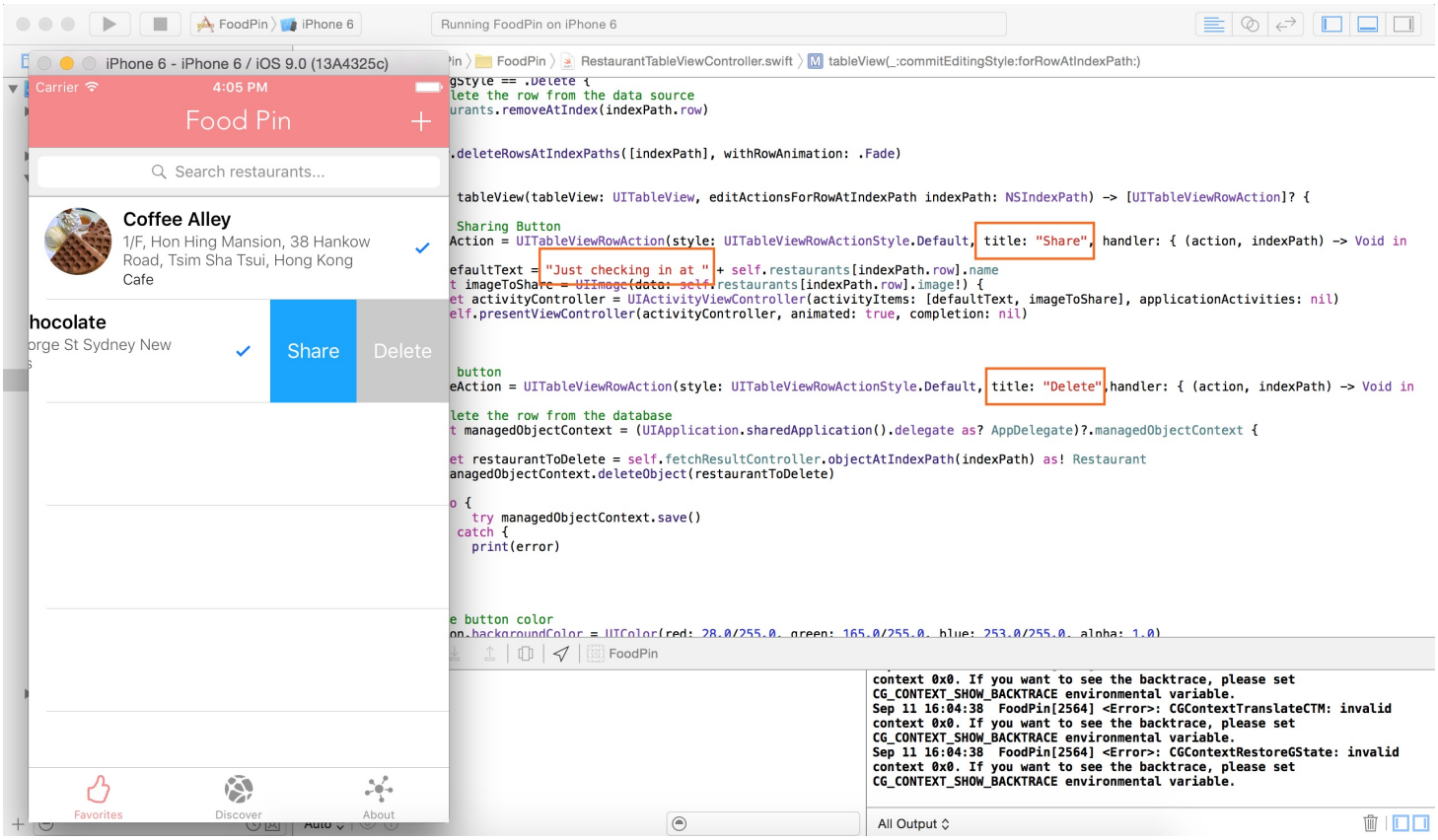


Figure 25-1. Sample user-facing text

At the heart of string internationalization is the `NSLocalizedString` macro. The macro allows you to internationalize the user-facing strings with ease. It has two arguments:

- `key` - the string to be localized
- `comment` - the string that is used to provide additional information for translation.

The macro returns a localized version of string. To internationalize the user-facing strings in your app, all you need to do is wrap the existing strings with `NSLocalizedString`. Let's do some code changes so you can fully understand the macro. Take a look at the below code snippet from the `RestaurantDetailViewController` class:

```
switch indexPath.row {
case 0:
    cell.fieldLabel.text = "Name"
    cell.valueLabel.text = restaurant.name
case 1:
    cell.fieldLabel.text = "Type"
```

```
    cell.valueLabel.text = restaurant.type
case 2:
    cell.fieldLabel.text = "Location"
    cell.valueLabel.text = restaurant.location
case 3:
    cell.fieldLabel.text = "Phone"
    cell.valueLabel.text = restaurant.phoneNumber
case 4:
    cell.fieldLabel.text = "Been here"
    if let isVisited = restaurant.isVisited?.boolValue {
        cell.valueLabel.text = isVisited ? "Yes, I've been here before" : "No"
    }
default:
    cell.fieldLabel.text = ""
    cell.valueLabel.text = ""
}
```

All the field labels are displayed in English and are non-localizable. To make them language independent, you just need to wrap the strings with `NSLocalizedString`:

```
switch indexPath.row {
case 0:
    cell.fieldLabel.text = NSLocalizedString("Name", comment: "Name Field")
    cell.valueLabel.text = restaurant.name
case 1:
    cell.fieldLabel.text = NSLocalizedString("Type", comment: "Type Field")
    cell.valueLabel.text = restaurant.type
case 2:
    cell.fieldLabel.text = NSLocalizedString("Location", comment:
"Location/Address Field")
    cell.valueLabel.text = restaurant.location
case 3:
    cell.fieldLabel.text = NSLocalizedString("Phone", comment: "Phone Field")
    cell.valueLabel.text = restaurant.phoneNumber
case 4:
    cell.fieldLabel.text = NSLocalizedString("Been here", comment: "Have you
been here Field")
    if let isVisited = restaurant.isVisited?.boolValue {
        cell.valueLabel.text = isVisited ? NSLocalizedString("Yes, I've been
here before", comment: "Yes, I've been here before") : NSLocalizedString("No",
comment: "No, I haven't been here")
    }
default:
    cell.fieldLabel.text = ""
    cell.valueLabel.text = ""
}
```

Xcode actually stores the localized strings in `Localizable.strings` files. Each language has its own `Localizable.strings` file. Say, your user's device is using German as the default language, `NSLocalizedString` looks up the German version of the Localizable.strings file and returns the string in German.

```
Quick tip: If you who don't know how an app picks a language, it actually
refers to the language settings of iOS (General > International > Language). An
app refers to that setting and access the corresponding localized resources.
```

That's all we need to do to internationalize the user-facing strings. But wait! How can you translate those labels in the storyboard? That's what we're going to discuss in the next section.

Export for Localization

Starting from version 6, Xcode comes with a new export feature to streamline the translation process. The export feature is intelligent enough to extract all localizable strings from your source code and Interface Builder/Storyboard files. The extracted strings are stored in an XLIFF file. If you haven't heard of the term, XLIFF stands for XML Localization Interchange File, which is a well-known and globally-recognized file format for localization.

To use the export feature, select the FoodPin project in the project navigator. Then head up to the Xcode menu and select Editor > Export For Localization. When prompted, choose a folder to save the XLIFF file. Xcode then examines all your files and generates the XLIFF file.

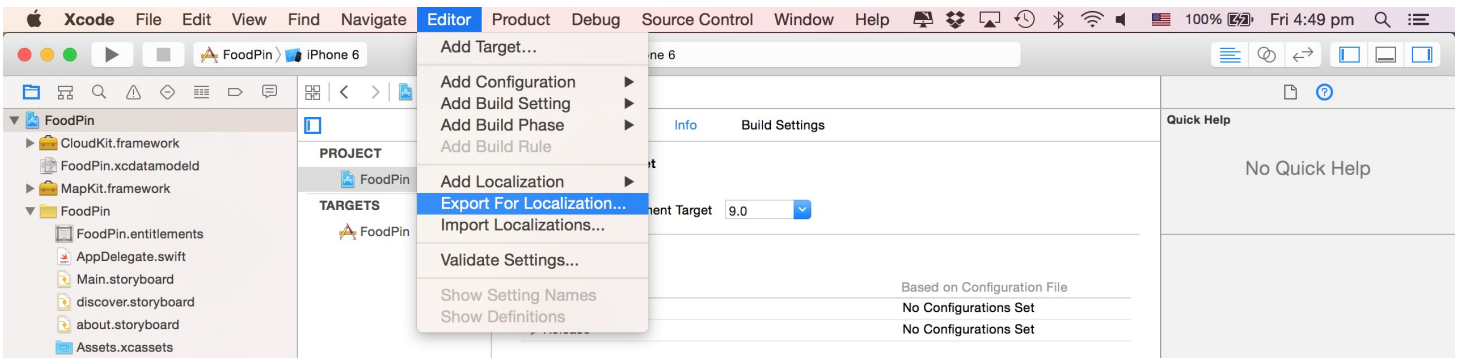


Figure 25-2. Export for Localization

If you accept the default file path, you should find the `en.xliff` file under your project folder. Open it with a text editor, you should find something like this in the file:

```
<trans-unit id="Been here">
        <source>Been here</source>
        <note>Have you been here Field</note>
      </trans-unit>
      <trans-unit id="Location">
        <source>Location</source>
        <note>Location/Address Field</note>
      </trans-unit>
      <trans-unit id="Name">
        <source>Name</source>
        <note>Name Field</note>
      </trans-unit>
      <trans-unit id="No">
        <source>No</source>
        <note>No, I haven't been here</note>
      </trans-unit>
      <trans-unit id="Phone">
        <source>Phone</source>
        <note>Phone Field</note>
      </trans-unit>
      <trans-unit id="Type">
        <source>Type</source>
        <note>Type Field</note>
      </trans-unit>
      <trans-unit id="Yes, I've been here before">
        <source>Yes, I've been here before</source>
        <note>Yes, I've been here before</note>
      </trans-unit>
```

Xcode has extracted the strings that we have just wrapped with the `NSLocalizedString` macro. Other than that, the file should have something like this:

```
<file original="FoodPin/Base.lproj/LaunchScreen.storyboard" source-
language="en" datatype="plaintext">
    <header>
      <tool tool-id="com.apple.dt.xcode" tool-name="Xcode" tool-version="7.0"
build-num="7A192o"/>
    </header>
    <body/>
  </file>
  <file original="FoodPin/Base.lproj/Main.storyboard" source-language="en"
datatype="plaintext">
    <header>
      <tool tool-id="com.apple.dt.xcode" tool-name="Xcode" tool-version="7.0"
```

```
build-num="7A192o"/>
    </header>
    <body>
      <trans-unit id="0Qh-qm-kX9.placeholder">
        <source>Restaurant Phone Number</source>
        <note>Class = "UITextField"; placeholder = "Restaurant Phone Number";
ObjectID = "0Qh-qm-kX9";</note>
      </trans-unit>
      <trans-unit id="3cw-Hj-7xA.text">
        <source>TYPE</source>
        <note>Class = "UILabel"; text = "TYPE"; ObjectID = "3cw-Hj-7xA";</note>
      </trans-unit>
```

The export feature already examined the storyboard and extracted all the localizable strings, including labels and button titles. Typically you pass the file to a professional translator for translation. The translator then uses an XLIFF-enabled tool to add all the missing translations. Or, like me, you can use Google Translate to do the translation and Xcode editor to edit the file. You only need to translate the text enclosed by the tag. Content inside the tag contains comments that we (or Xcode) added, so you do not need to translate them. The translated text is placed inside the tag. Below is a part of the Traditional Chinese translation file:

```
  <file original="FoodPin/Localizable.strings" source-language="en" target-
language="zh-Hant" datatype="plaintext">
    <header>
      <tool tool-id="com.apple.dt.xcode" tool-name="Xcode" tool-version="7.0"
build-num="7A192o"/>
    </header>
    <body>
      <trans-unit id="Been here">
        <source>Been here</source>
        <target>有來過這裡嗎</target>
        <note>Have you been here Field</note>
      </trans-unit>
      <trans-unit id="Location">
        <source>Location</source>
        <target>位置／地址</target>
        <note>Location/Address Field</note>
      </trans-unit>
      <trans-unit id="Name">
        <source>Name</source>
        <target>名字</target>
        <note>Name Field</note>
      </trans-unit>
      <trans-unit id="No">
        <source>No</source>
        <target>沒有</target>
```

```
        <note>No, I haven't been here</note>
      </trans-unit>
      <trans-unit id="Phone">
        <source>Phone</source>
        <target>電話號碼</target>
        <note>Phone Field</note>
      </trans-unit>
      <trans-unit id="Type">
        <source>Type</source>
        <target>類別</target>
        <note>Type Field</note>
      </trans-unit>
      <trans-unit id="Yes, I've been here before">
        <source>Yes, I've been here before</source>
        <target>對！我曾到過這裡。</target>
        <note>Yes, I've been here before</note>
      </trans-unit>
    </body>
  </file>
```

The first line of the XML code specifies the original file and the target language of the translation. The target language specifies the language code of the translation. `zh-Hant` is the language code of Chinese. For German, the language code is `de`.

Import Localizations

Assuming your translator has completed the translations and passed you the translation files (https://www.dropbox.com/s/yw8rczyntkm6k1z/translations.zip?dl=0), you just need a few clicks to import the translations. Head up to the Editor menu and select *Import Localizations*. When prompted, select the translation file (e.g. zh-Hant.xliff). Xcode automatically compares the translation with the existing files and shows you the differences (see figure 25-3).

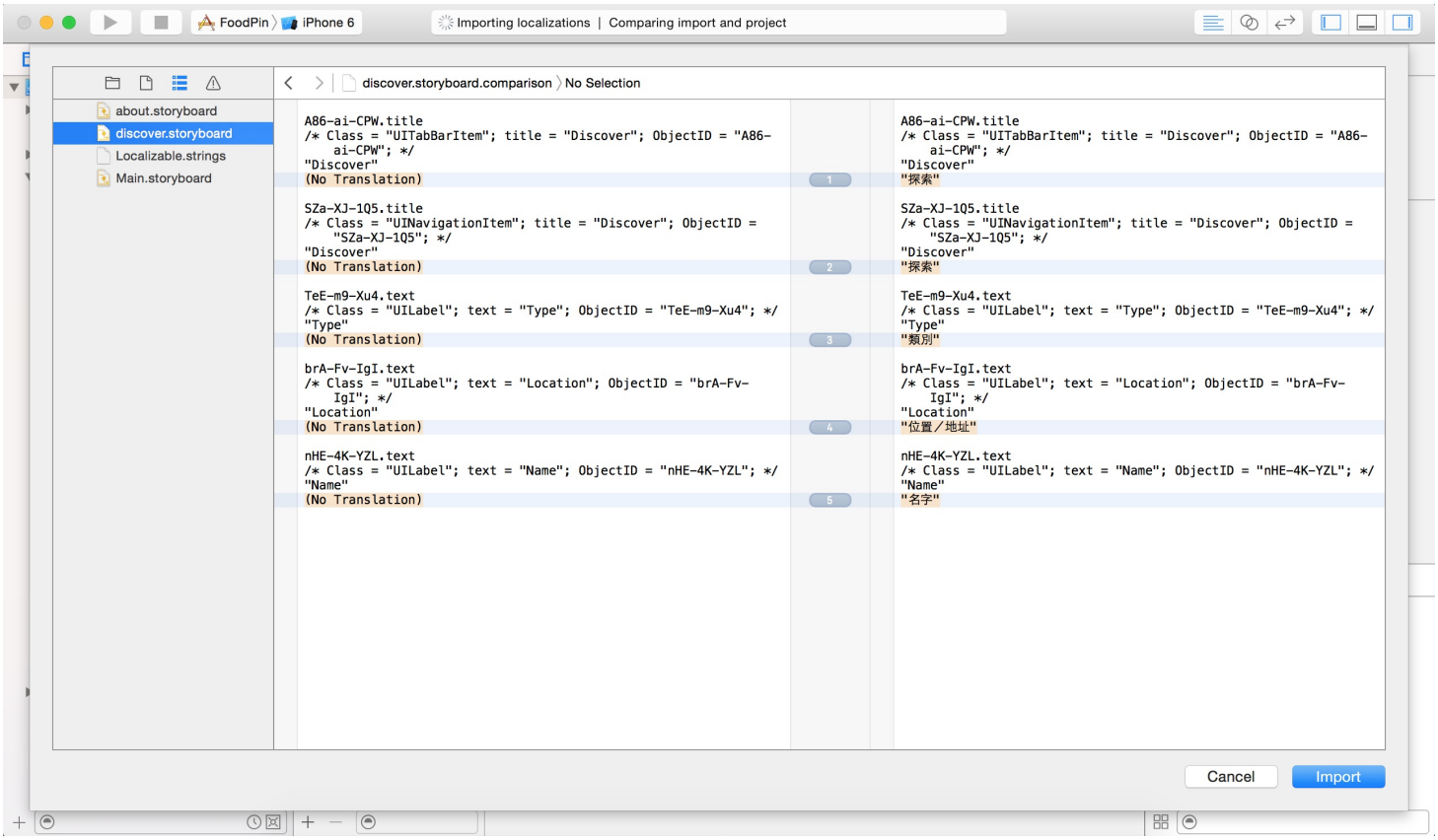


Figure 25-3. Import localizations

If you confirm the translation is correct, click `Import` button to import it right away. If you have other translations (e.g. de.xliff), repeat the same procedures to add them. When finished, Xcode displays the available localizations in the project screen. You should notice that Xcode adds the localized resource files in the project navigator.

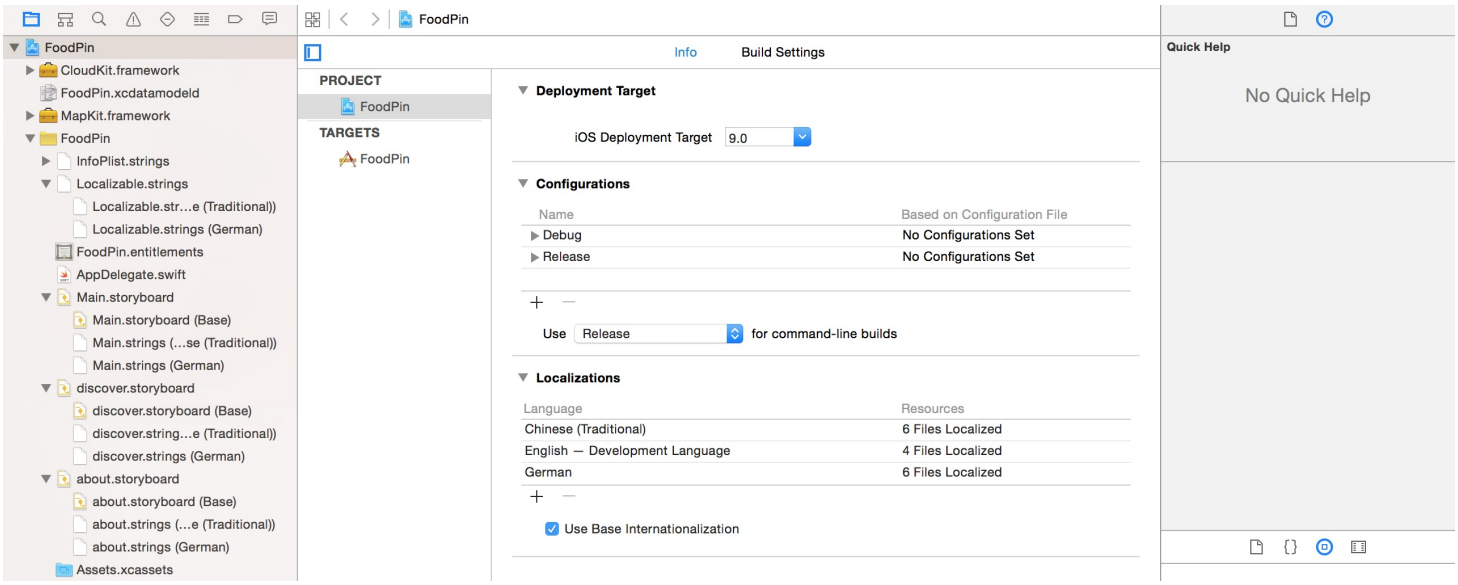


Figure 25-4. German and Chinese localizations added to the Xcode project

Before we move on to testing the localized app, let me give you some more information about base internationalization and the localized resource files. Let's first take a look at `Localizable.strings`. These files are automatically generated after the import process. The localized strings in source code (i.e. those wrapped with `NSLocalizedString`) are stored in a language-dependent `Localizable.strings` file. As we have imported the `zh-Hant` and `de` translations, you should find two versions of `Localizable.strings` files in the project navigator.

If you look into any of the `Localizable.strings` file (say, the Chinese version), you will find the translation of the localizable strings. At the beginning of each entry, it is the comment you've put in the source code. On the left side of the equal sign, it is the key you specify when using `NSLocalizedString` macro.

```
/* Have you been here Field */
"Been here" = "有來過這裡嗎";

/* Location/Address Field */
"Location" = "位置／地址";

/* Name Field */
"Name" = "名字";

/* No, I haven't been here */
```

```
"No" = "沒有";

/* Phone Field */
"Phone" = "電話號碼";

/* Type Field */
"Type" = "類別";

/* Yes, I've been here before */
"Yes, I've been here before" = "對！我曾到過這裡。";
```

Quick note: While you can edit the translations directly, I would recommend you to use the export feature in Xcode.

The second thing I want to talk about is *base internationalization*. As you can see from figure 22-4, the Main.storyboard is now split into three files. Two of them store the translations. So what is `Main.storyboard (Base)`?

The concept of base internationalization was first introduced in Xcode 4.5. Before Xcode 4.5, there was no concept of base internationalization. Xcode replicates the whole set of storyboards for each localization. For example, let's say if your app is localized into 5 languages. Xcode generates 5 sets of storyboards for localization purposes. There is a major drawback of this design. When you need to add a new UI control in the storyboard, you'll need to add the same element in each localized storyboard. That's a tedious process, which is why Apple introduced base internationalization in Xcode 4.5.

With base internationalization, an Xcode project has just one set of storyboards that is localized to the default language. This storyboard is known as the base internationalization. Whenever the storyboard is localized into another language, Xcode only generates a `.strings` file containing all the text of the base storyboard. This efficiently separates the UI design from the translations.

Testing the Localized App

One way to test the localization is to change the language preference of the simulator and then run the localized app on it. Starting from Xcode 6, Apple provides a powerful preview feature that lets you preview your app in different languages and regions, both at runtime and in Interface Builder. I will go through them with you one by one.

Xcode 6 supports preview at runtime. You can enable this feature by editing the scheme sheet and set your preferred language in the dialog box.

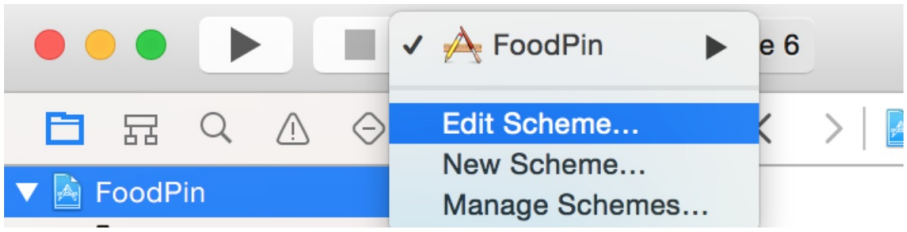


Figure 25-5. Edit scheme option

In the dialog box, select *Options* and change the application language to your preferred language. For example, Chinese (Traditional). Click the Close button to save the setting.

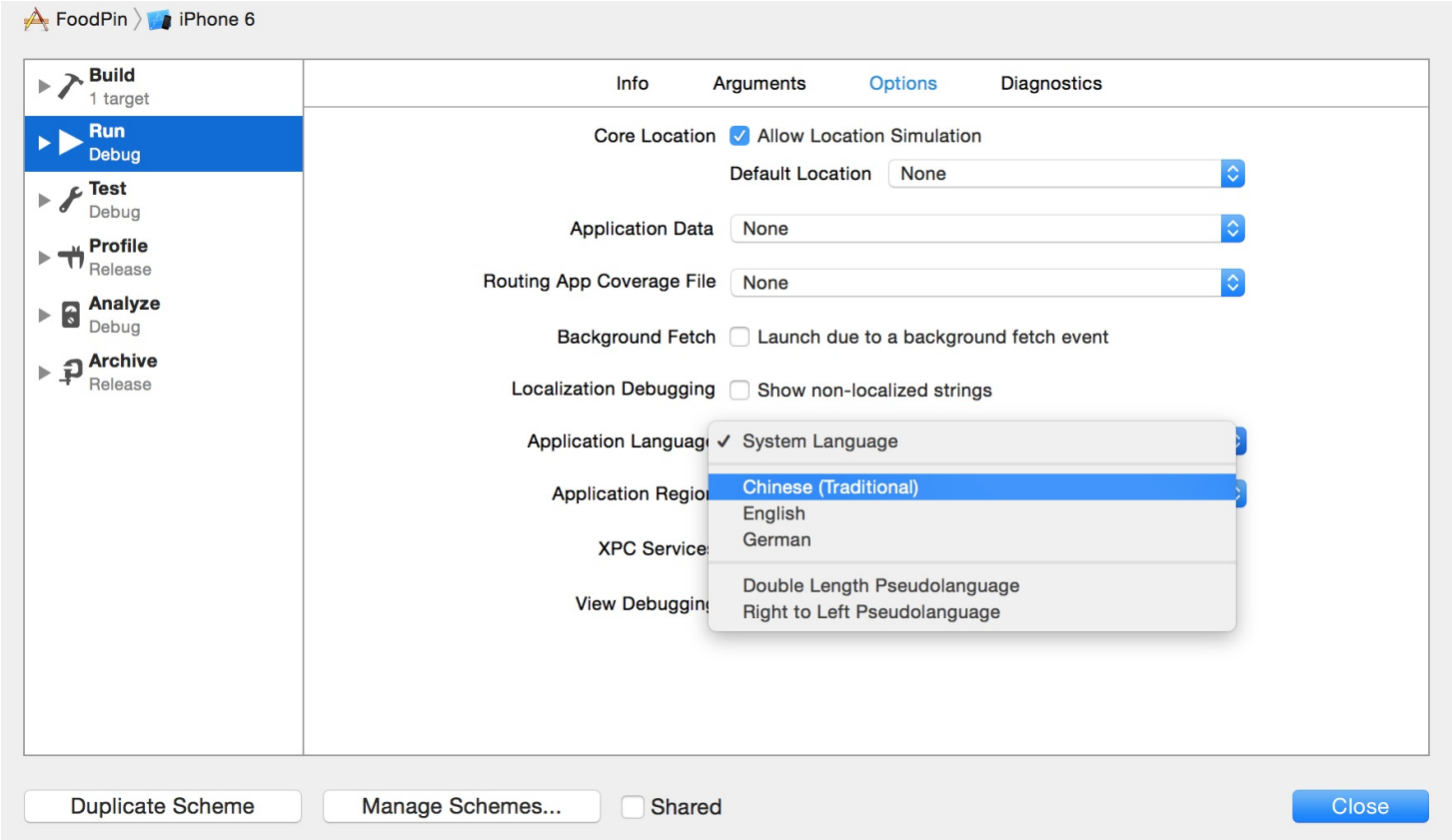


Figure 25-6. Changing the application language to your preferred language

Now click the Run button to launch the app; the language of the simulator should set to your preferred language. If you've set it to Chinese, your app should look like the screenshot shown in figure 25-7.

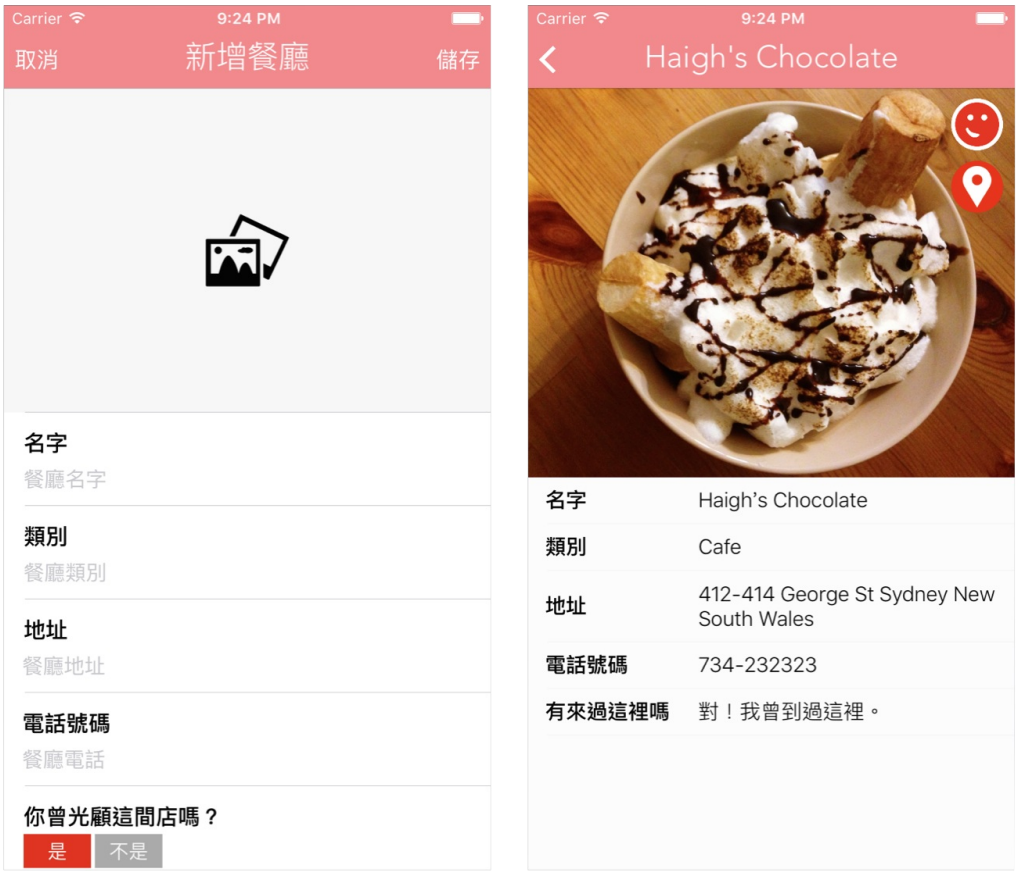


Figure 25-7. FoodPin App in Traditional Chinese

Alternatively, Xcode allows you to preview the localized app UI in Interface Builder. In the preview assistant, you can preview the localized UI by selecting the language from the language pop-up at the lower right corner of the preview window. If you forgot how to open the preview assistant, please go back to Chapter 5 and check out the *Previewing Storyboards* section.

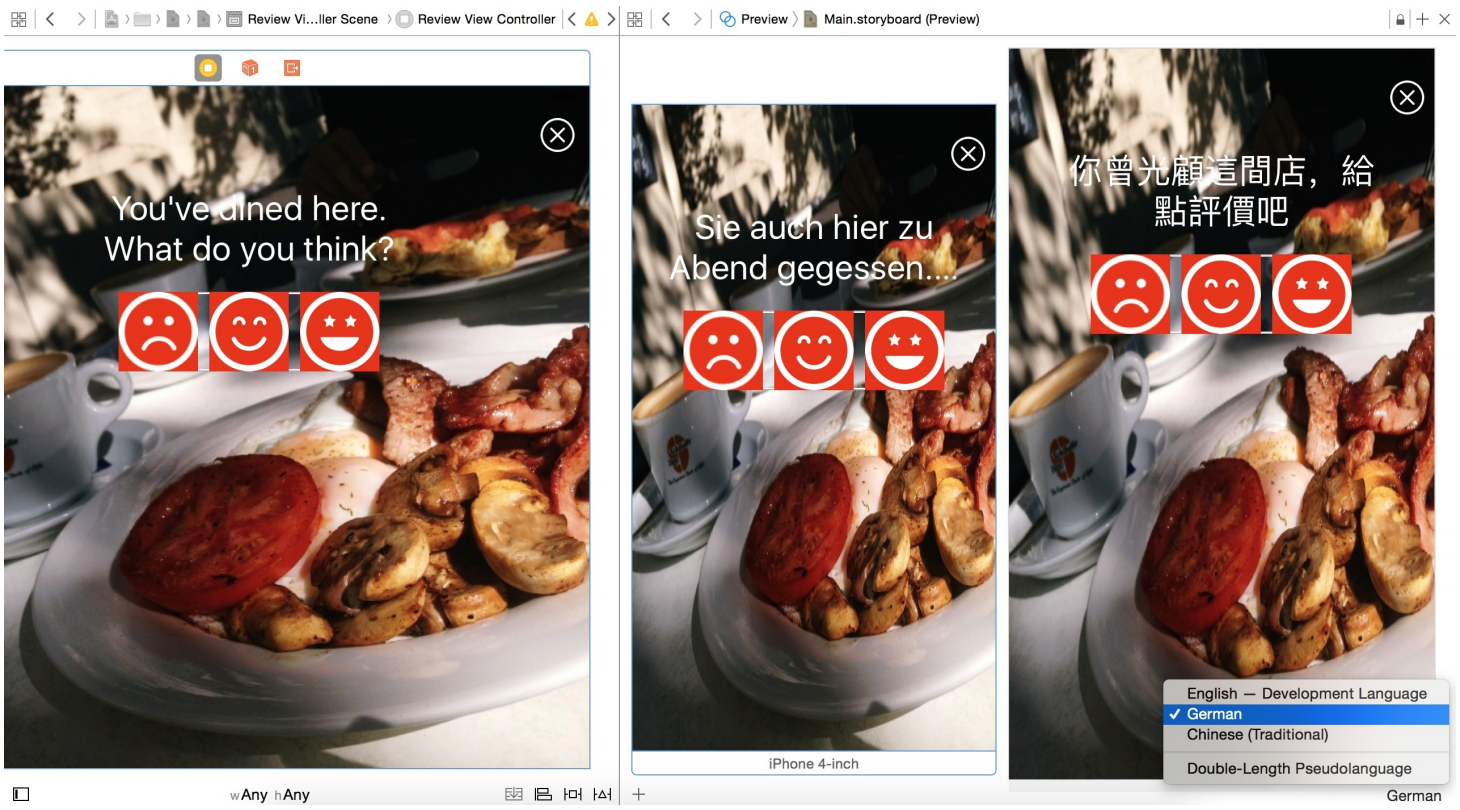


Figure 25-8. Previewing your localized app UI using Preview Assistant

Localizing Resources

Other than text, you may want to localize resources such as images, videos and sounds. For example, you can display a locale-specific image in the About tab. Figure 25-9 shows the German and Chinese version of the image.

Figure 25-9. German (left) and Chinese (right) version of the image in About tab

From the very beginning, we used the asset catalog to manage the project's images. Unfortunately, asset catalogs do not support localization. To localize an image in your app, one way is to import the file directly into the project navigator. First, download this image pack from https://www.dropbox.com/s/vizw3tm2i76a6n4/aboutfoodpin.zip?dl=0. After unzipping the file, you will find three versions of the about logo.

Now, right click on the FoodPin folder in the project navigator and select `New Group`. Name the group *images*. This step is optional but creating a new group allows you to better organize your project resources. Next, drag `aboutfoodpin.png` from Finder to the project navigator. Make sure that *Copy item if needed* is enabled when prompted. The image is then copied to the project folder. Now, select the file and bring up the File inspector. In the localization section, you should find the `Localize…` button. Click the button, choose `Base` and click the `Localize` button to confirm.

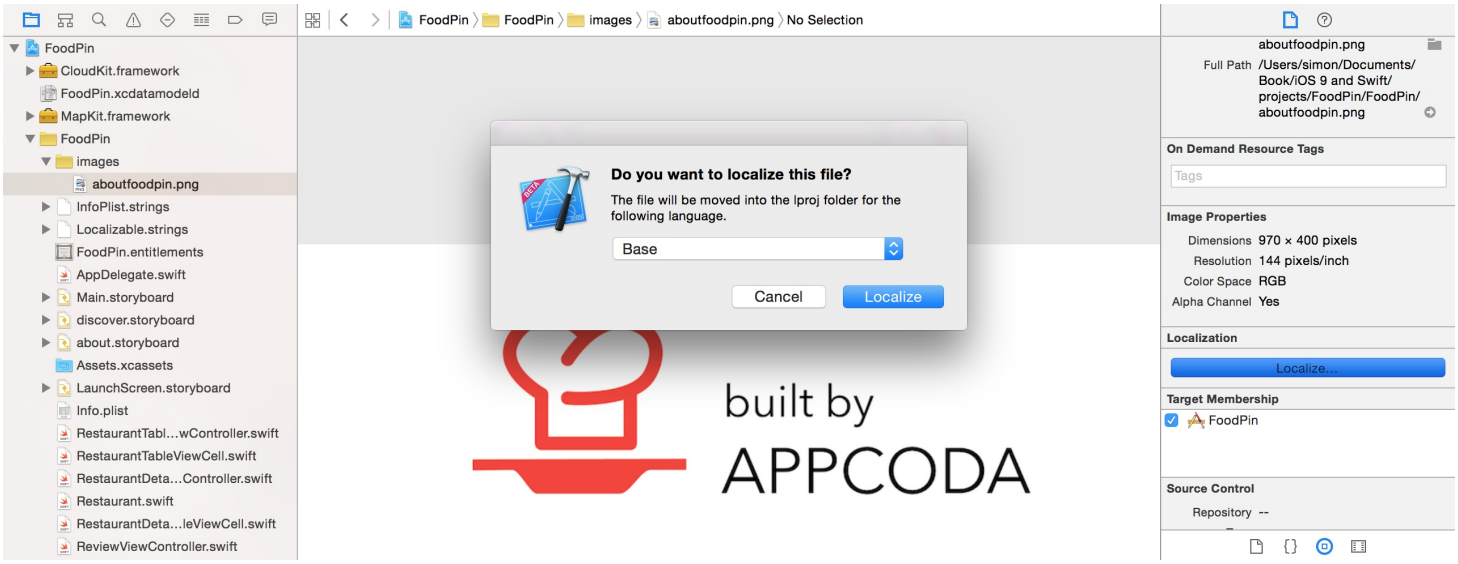


Figure 25-10. Localize the about logo

After that, you'll see the Chinese (Traditional) and German options in the localization section. Select both languages and Xcode code will generate three versions of aboutfoodpin.png in project navigator.

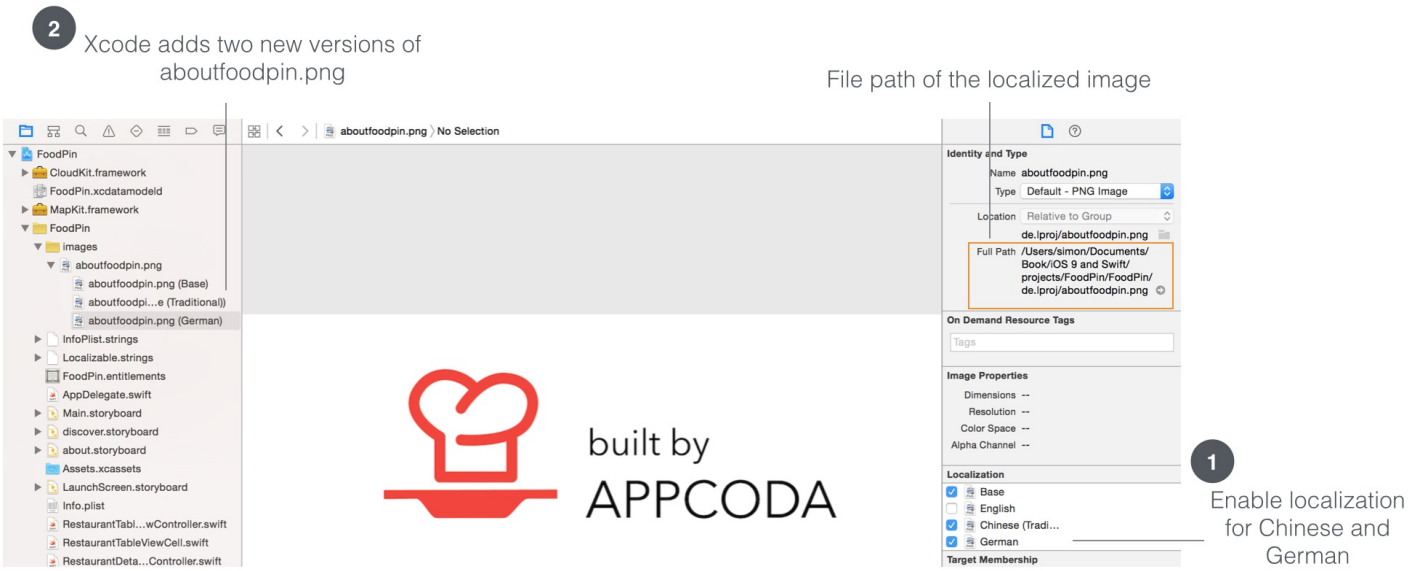


Figure 25-11. Enable the German and Chinese (Traditional) localization

Now that you've created the localized versions of the image, they are stored in separate folders. You can locate the folder by selecting the localized version of image to reveal the full path in the File Inspector. The German version of the image is stored under the `de.lproj` folder, while the Traditional Chinese version is stored under the `zh-Hant.lproj` folder.

By default, the image generated in the localized folders is the same as the original. To localize the image, what you need to do is to replace the images under `de.lproj` and `zh-Hant.lproj` with their own localized versions.

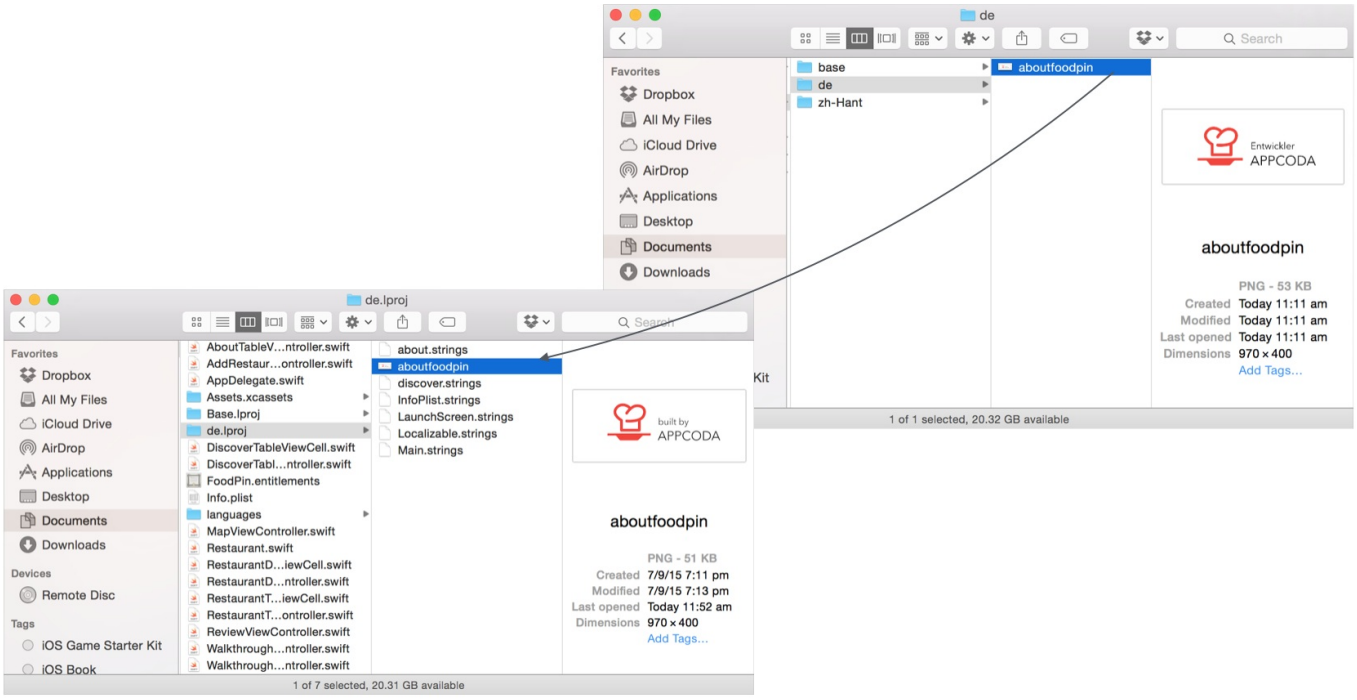


Figure 25-12. Copy your own localized images to replace the original image

Lastly, remember to change the image of the About view controller from about to `aboutfoodpin.png` in `about.storyboard`, because we no longer use the about image from the asset catalog.

Cool! It's time to test the localized UI. If you got everything right, here are what you'll see for both German and Traditional Chinese versions.

Figure 25-13. Food Pin app in German (left) and Traditional Chinese (right)

Summary

I have walked you through the localization process in Xcode 7. With the export feature and preview function, localizing an iPhone app has never been as easy as it is today. In this chapter, you've learned how to localize text and images in storyboards using the built-in localization support of Xcode. You should also know how to localize strings using the `NSLocalizedString` macro. Remember that your apps are marketed to a global audience - users prefer to have an app UI in their own language. By localizing your app, you'll be able to provide a better user experience and attract more downloads.

For reference, you can download the complete Xcode project from https://www.dropbox.com/s/6htjtx0tdti2i4k/FoodPinLocalization.zip?dl=0.