Chapter 1 Cross-platform device capabilities

Applications in the real-world need to make sure the user has always control on what's happening, especially with unexpected situations, and they need to respond accordingly. For example, if the network connection drops off, not only the app must intercept the loss of connection, but it also needs to communicate to the user that a problem happened and provide one or more actions, and this should be done with a nice user interface, avoiding unnecessary modal dialogs that might be okay for desktop apps, but not on mobile apps where the user should never feel to be blocked. This chapter will describe how to solve common problems that mobile apps might encounter, in a cross-platform approach. You will not find how to work with sensors, which is something specific to some app categories, rather you will find ways of handling scenarios that almost every app will need to face.

Platform integration in .NET MAUI

If you have experience with Xamarin.Forms, you know that accessing device capabilities in a cross-platform approach is possible via the Xamarin.Essentials library, an open source project created and maintained by Microsoft that you can install from NuGet. This library exposes a large set of APIs that you can invoke from the shared project and that avoids the need of writing platform-specific code for capabilities that are common to all the supported systems. In .NET MAUI, there is no external library. A similar set of APIs is already built-in the core library, so you will just need to add the appropriate using directives (as you will see in this chapter). This also avoids the need of an additional NuGet package in your solution.

Note: You might often hear about .NET MAUI Essentials as a way to refer to this set of APIs, but actually this does not really exist. It's a common way for existing Xamarin.Forms developers to quickly refer to that part of MAUI that allows for accessing device capabilities.

Chapter prerequisites

In order to follow the examples provided in this chapter, create a new .NET MAUI solution using the default template, as shown in Figure 1.

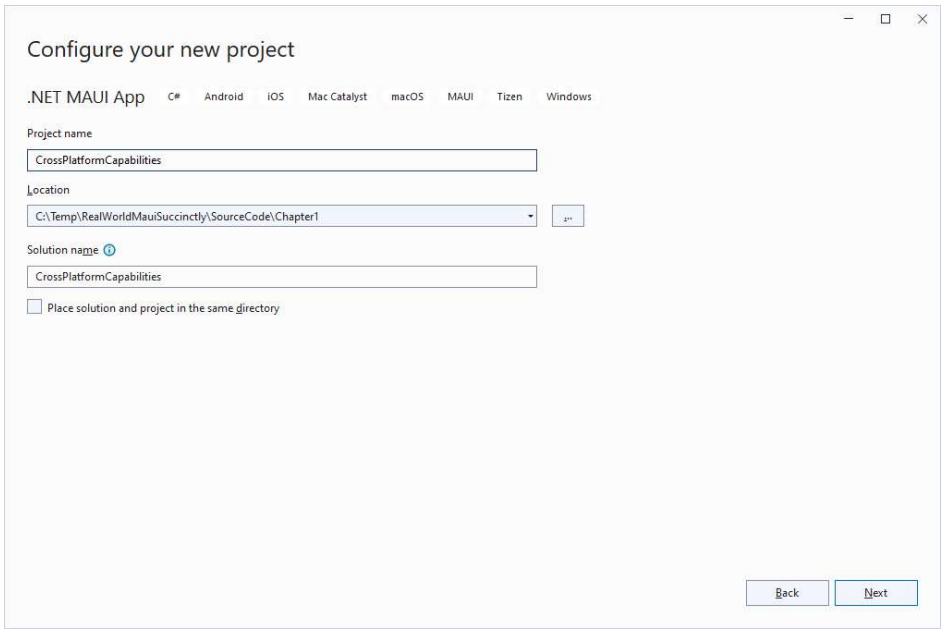

Figure 1: Creating a new .NET MAUI solution

Select .NET 7 as the target technology. When the solution is ready, make sure you install the .NET MAUI Community Toolkit. This library is required to use Snackbar views later. To do so, right-click the solution name in Solution Explorer, then select **Manage NuGet Packages for Solution** and then open the Browse tab. Search for the library (see Figure 2) and then click the **Install** button.

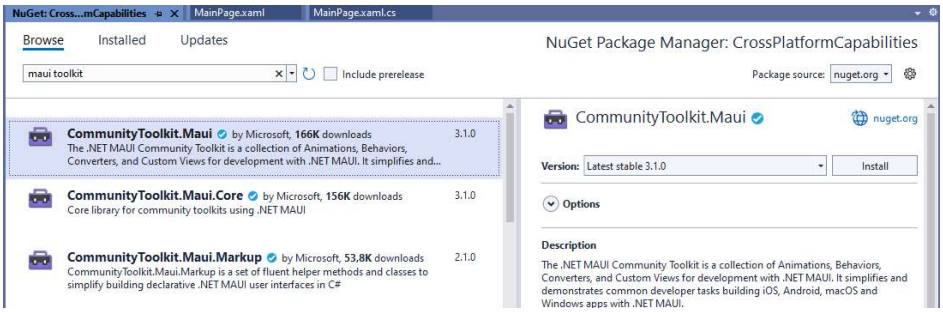

Figure 2: Installing the .NET MAUI Community Toolkit

The Community Toolkit must be initialized before you can use it. In the MauiProgram.cs file, change the **`UseMauiApp`** invocation as follows:

```
builder.UseMauiApp<App>().ConfigureFonts(fonts =>
{
    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
}).UseMauiCommunityToolkit();
```