**Creating an Apple developer account**

Step 1: Visit developer.apple.com

Step 2: Click Member Center.

Step 3: Sign in with your Apple ID.

If you already have an Apple ID, you’ll just need to agree to Apple’s Developer Agreement to turn it into a free developer account. Type in your Apple ID and click Sign In.

If you don’t already have an Apple ID, you’ll need to create one first. Click Create Apple ID, and fill out the required information and click Continue. You’ll then need to sign in with the Apple ID that you just created, and go back to the Member Center.

Step 4: On the Apple Developer Agreement page, click the first check box to accept the agreement and click the Submit button.

Your Apple ID now works as a free developer account. As stated at the outset, this account can be used for side loading apps via Xcode. You will not be able to submit apps to the App Store, or download iOS, OS X, watch OS, or tvOS developer betas. A $99/year paid developer account is still required to enjoy those privileges.

**Signing in with Xcode**

Step 1: Download Xcode from the Mac App Store.

Step 2: Launch Xcode.

Step 3: Click Xcode → Preferences → Accounts and click the ‘+’ sign and choose Add Apple ID.

Step 4: Login with the Apple ID that you just enabled with free developer privileges.

**Figure 2-1 Steps to create a development team**



Adding Members to a Team

After the team agent has joined a developer program, they add other people to the team and set their privileges. If you are the team agent and the sole developer on your team, no additional configuration is needed, because the team agent always has access to all account features. However, you should continue to read this section to understand the kinds of tasks you may need to perform throughout the rest of the development process.

To add a new person to the team, the team agent sends an invitation to the person; part of sending an invitation includes setting that person’s privileges on the team. When the person accepts the invitation, they are automatically added to the team.

|  |  |
| --- | --- |
| Team agent | A *team agent* is legally responsible for the team and acts as the primary contact with Apple. The team agent can change the access level of any other member of the team. |
| Team admin | A *team admin* can set the privilege levels of other participants, although a team admin cannot demote the team agent. Team admins manage all assets used to sign your apps, either during development or when your team is ready to distribute an app. Team admins are the only people on a team that can sign apps for distribution on non-development devices. Team admins also approve signing certificate requests made by team members. |
| Team member | A *team member* gains access to prerelease content delivered by Apple on that program’s portal. A team member can also sign apps during development, and but only after they make a request for a development signing certificate and has that request approved by a team admin. |

**Organizing the Development Team**

Now that you understand the roles people can assume in Member Center and iTunes Connect, you should consider how you want to organize the team. Organizing the team requires more than setting privileges described above. The size of the team affects how you organize it and its assets.

The smallest team is a single person—you. You act as the team agent and have full privileges to perform any task. The disadvantage of a one-person team is that you have to do all the work. You need to set up the assets needed to sign and publish an app, configure all the information in iTunes Connect, develop an app, and market it.

**Figure 2-2 An individual is the team agent**



A more common configuration is a small developer team. On a small team, the team agent is also a programmer, but defers some of the administration overhead to another person on the team. The team agent handles all of the financial and sales operations for the team, while the team admin ensures that the developers on the team have what they need to get the job done.

**Figure 2-3 A small development team**



If your team is large, you can divide the tasks further. Some people on the team might not be programmers, including the team admin. The sales and financial roles might be filled by people with a business background. You might even have in-house testers who are not also programmers. The task of creating and shipping apps could be delegated to one or more dedicated team admins. You might even partition the work further and divide the team between iOS and OS X development.

**Figure 2-4 A large development team**



As the team grows, the need for coordination between the different people on the team increases. In particular, the cost of development errors increases when you have more programmers on the team. For example, if you are the sole member of the team, you can store all your work on a single computer and do all of the work there. Xcode even allows you to create a local source code repository to store your code. But what happens when the team grows in size?

When the team grows to a moderate size, you need more infrastructure. You want a separate computer to act as a remote source code repository; members of the team pull down the code from the remote repository to their computers, make changes, and send those changes back to the server. But when multiple developers are syncing code, the chance of an error being checked into the code increases. To minimize problems, you usually want multiple branches of development, including stable branches that hold the code you use to build your shipping app and experimental or developmental branches used for new development. These strategies require a deeper understanding of the underlying source code repository and require you to define specific policies that describe how code gets integrated between the different branches you maintain.

If your team grows very large, checking erroneous code into developmental branches could impair your team’s productivity. By the time you discover an error, it might already have been synced to other people’s computers. After you correct the error, propagating the fixes to everyone on the team still takes time. The frequency of such incidents and the time lost for each incident increase as the size of the team increases; these losses, when aggregated, cripple your team’s effectiveness. To avoid this lost productivity, you should add additional infrastructure for code management. For example, you might adopt a process of continuous integration by creating a dedicated build machine (known as a build bot) that automatically checks out each change and rebuilds your app. By building new changes as they arrive, you discover errors earlier; you might even configure the server to back out such changes before others on the team pull down the offending code. If your project includes unit tests or automated user tests, the build computer can also run those tests to verify that your source code still passes the tests. This process, known as smoke testing, increases the team’s confidence that code stored in the development branches builds and works properly.

As the team grows larger, you want to automate tasks that are repeatable, predictable, and costly to perform by hand. By automating common tasks, you reduce the burden on the team and allow it to focus on designing and implementing code.