Automating the Test Process

In addition to running tests interactively during development, take advantage of automating test runs using a server.

Server-Based Testing with Continuous Integration

The Xcode testing capabilities, used interactively, ensure that your code stays on track with respect to its specified requirements and ensure that bugs which develop are easy to find and fix. A suite of fast-running functional tests proofs your work as you go and ensures solid app foundations that you can build upon efficiently and confidently.

That said, successful development projects tend to grow beyond the bounds of a single developer to implement and maintain. Like source management, automated testing on a server provides benefits by allowing your development effort to scale to the needs of a team smoothly and efficiently.

Here are some benefits of using server-based testing:

- Using a server for offline build and test frees your development system to do implementation and debugging, particularly in the case when the full suite of tests takes a long time to run.
- All members of the development team run the same tests on the server by using the same scheme, thus enhancing test consistency. A server also makes build products available to the entire team, just as with build and test reports.
- You can adjust scheduling to both the project needs and your team needs. For instance, test runs can start when any member of the team commits new work to the source management system, or periodically, at set times. Test runs can also be started manually whenever required.
- The server runs the tests time after time, in exactly the same way. The reportage from the server benefits you and your team by giving you over time a picture of build issues, build warnings, and test resolutions.
- Your projects can be tested on many more destinations, automatically—and with more economy—than on a manually run testing system. For example, you can have an arbitrary number of iOS devices connected to the server, and with a single configuration, the system can build and test the libraries, apps, and tests on all of them, and in multiple versions of Simulator as well.

Command Line Testing

Using Xcode command-line tools, you can script and automate both the building and testing of your project. Use this capability to take advantage of existing build automation systems.

Running tests with xcodebuild

The `xcodebuild` command-line tool drives tests just like the Xcode IDE. Run `xcodebuild` with the action `test` and specify different destinations with the `-destination` argument. For example, to test MyApp on the local macOS “My Mac 64 Bit,” specify that destination and architecture with this command:

```
> xcodebuild test -project MyAppProject.xcodeproj -scheme MyApp -destination
'platform=OS X,arch=x86_64'
```

If you have development-enabled devices plugged in, you can call them out by name or id. For example, if you have an iPod touch named “Development iPod touch” connected that you want to test your code on, you use the following command:

```
> xcodebuild test -project MyAppProject.xcodeproj -scheme MyApp -destination
'platform=iOS,name=Development iPod touch'
```

Tests can run in Simulator, too. Use the simulator to target different form factors, operating systems, and OS versions easily. Simulator destinations can be specified by name or id. For example:

```
> xcodebuild test -project MyAppProject.xcodeproj -scheme MyApp -destination
'platform=Simulator,name=iPhone,OS=8.1'
```

The `-destination` arguments can be chained together, letting you issue just one command to perform integrations across the destinations for the specified shared scheme. For example, the following command chains the previous three examples together into one command:

```
> xcodebuild test -project MyAppProject.xcodeproj -scheme MyApp

-destination 'platform=OS X,arch=x86_64'

-destination 'platform=iOS,name=Development iPod touch'

-destination 'platform=Simulator,name=iPhone,OS=9.0'
```

If any tests fail, `xcodebuild` returns a nonzero exit code.

For more information, see How do I run unit tests in OS X and iOS from the command line? in *Building from the Command Line with Xcode FAQ*. You can also get complete usage information for the `xcodebuild` command with the following command:

```
> man xcodebuild
```

Using ssh with xcodebuild

Invoking `xcodebuild` from a remote login with `ssh` (or from a launch demon) fails unless the correct session environment is created on the host.

An “Aqua session” environment is created when you interactively log into your macOS system as a user. Aqua sessions initialize the infrastructure of the macOS interactive environment; they are required in order to be able to run macOS apps. To be more specific, code using UI frameworks (AppKit or UIKit) needs to run in an Aqua session. Because of this requirement, testing on macOS (and also testing on the Simulator, itself an macOS app) requires an Aqua session.

By default, when you use `ssh` to login to an macOS system that has no active user session running, a command-line session is created. To ensure that an Aqua session is created for an `ssh` login, you must have a user logged in on the remote macOS host system. The existence of a user running on the remote system forces Aqua session for the `ssh` login.

Once there is a user running on the host system, running `xcodebuild` from an `ssh` login works for all types of tests. For example, the following Terminal app commands run the tests defined for “MyApp” on the development system host from `ssh`:

```
> ssh localhost

> cd ~/Development/MyAppProject_Folder

> xcodebuild test -project MyApp.xcodeproj -scheme MyApp -destination
'platform=Simulator,name=iPhone 6'
```

For those who need a deeper of understanding of `ssh`, launch demons, and launch agents, and how they interact with the system, see the technical notes *Daemons and Agents* and *Daemons and Services Programming Guide*.

Using Xcode Server and Continuous Integration

Xcode supports a fully integrated, server-based continuous integration workflow through Xcode Server. Xcode Server, available in macOS Server, automates the integration process of building, analyzing, testing, and archiving your app. Using Xcode Server and the continuous integration workflow is designed to be seamless and transparent to your interactive development work.

To learn all about setting up and using Xcode Server, see *Xcode Server and Continuous Integration Guide*.