SOASTA

TouchTest™ Jenkins CI for iOS Tutorial

Table of Contents

Prerequisites

This tutorial guides the user through the process of using the Jenkins continuous integration tool combined with the CloudTest Jenkins/Hudson Plugin in tandem with an example Xcode project and preconfigured CloudTest test compositions.

This tutorial provides guidance for two audiences:

- Users who would like to add iOS Testing using Xcode to a pre-existing Jenkins setup
- Users who are iOS Developers starting out with TouchTest who would also like to add TouchTest to a continuous integration setup

TIP: If your organization is not already using Jenkins—refer to the documentation on the Use Jenkins page to get started. Additional Jenkins installation references are included at the end of this guide.

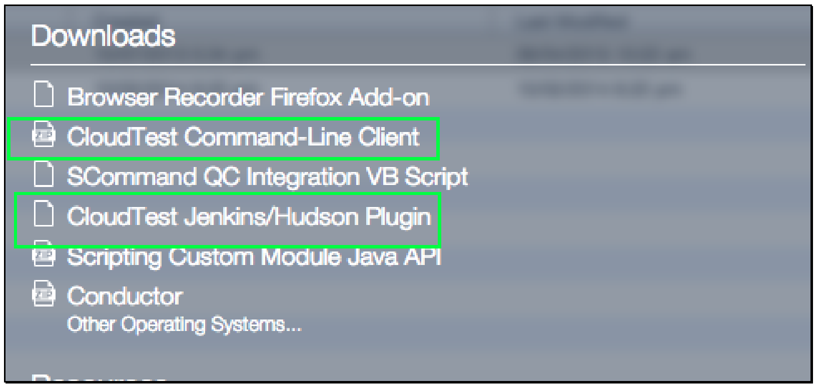
A Jenkins job will be defined that uses git to retrieve an Xcode project, then uses the MakeAppTouchTestable utility to make that Xcode project touchtestable, and which then deploys either an IPA to a device or the built APP to a simulator or device(s) using a deployment step; after which several pre-existing test compositions are called using sCommand to run silently in CloudTest and on the specified devices. Finally, CloudTest results are inspected inline in Jenkins, also via the CloudTest Jenkins/Hudson Plugin.

Note: A version of the Stockfish chess game, an open-source project available from GitHub, has been customized to include the Xcode project that is used as the example project. You can, of course, substitute your own mobile app's source and Xcode project for this example.

CloudTest Utilities and Plugins

Before proceeding, download the following CloudTest plugin and utility software from the CloudTest Resources page, Downloads section.

- **CloudTest Jenkins/Hudson Plugin** (this Jenkins/Hudson plugin will be installed using the Jenkins plugin interface). Jenkins CloudTest Plugin version 2.9 or later and TouchTest 51.07 or later are required for dynamic instrumentation.

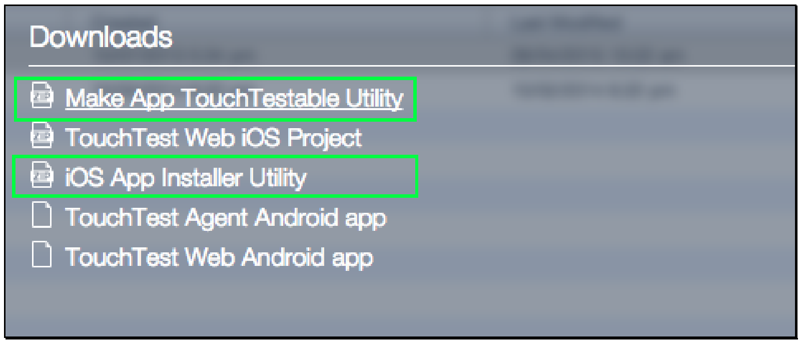

Downloads
Browser Recorder Firefox Add-on
CloudTest Command-Line Client
SCommand QC Integration VB Script
CloudTest Jenkins/Hudson Plugin
Scripting Custom Module Java API
Conductor
Other Operating Systems...

- **CloudTest Command Line Client** (also known as sCommand, this command line interface utility will be called at the appropriate time via a Jenkins job using the CloudTest Jenkins Plugin's MakeAppTouchTestable, Play Composition(s) build step). Users may opt to do some tasks via the Plugin while others are accomplished via the sCommand utility. In either case, you should install the utility.

TouchTest Utilities and Plugins

Next, download the following TouchTest software from the TouchTest Resources page, Downloads section.

- **MakeAppTouchTestable Utility** (this utility will be called at the appropriate time via a Jenkins job using an Execute Shell build step)


Downloads
Make App TouchTestable Utility
TouchTest Web iOS Project
iOS App Installer Utility
TouchTest Agent Android app
TouchTest Web Android app

Note: The CloudTest user specified to run the MakeAppTouchTestable utility must be a user with Mobile Device Administrator rights.

- For iOS only, the CloudTest **iOS App Installer Utility** is required (this utility contains two executable files; the `ios_app_installer`, which is used to install IPA files to iOS physical devices; and the `ios_sim_launcher`, which is used to install compiled APP bundle files). This archive contains two executable files:
    - For deployment to Simulators, use the `ios_sim_launcher` executable found in the iOS App Installer Utility at the appropriate time(s) via a Jenkins job using an Execute Shell build step.
    - For deployment to iPhone and iPad devices, use the iOS App Installer Utility to deploy .ipa archives to the physical device(s). This executable can be called at the appropriate time(s) via a Jenkins job using an Execute Shell build step.

  The appropriate executable will be called at the appropriate time(s) via a Jenkins job using an Execute Shell build step.

Prerequisites for Deploying to a Mac using Xcode

One of the key steps during an iOS automated build is deploying the app to your test device, *without requiring any human interaction*. Typical solutions (e.g. over-the-air distribution) require that the user accept a prompt. SOASTA's iOS App Installer Utility includes two tools that *silently* deploy either an IPA file or an App file.

You will need the following:

1. A dedicated machine running Mac OS X, with Xcode 4.2 or later. If you are using Jenkins or Hudson, this can be either the master node or a slave.
2. One or more **tethered** devices. If you have more devices than USB inputs, you can use a USB hub. Note also that sufficient power to prevent the device from running down unexpectedly should be available via that USB input. Simulators can also be used.

   A note on tethering: SOASTA TouchTest™ does **not** require tethering for recording or playback. However, you do need to tether the device for silent deployment of your app.
3. The physical iOS device(s) should have the iOS "Auto-Lock" setting set to "Never" and passcodes should not be in use on any device.

Test Composition Prerequisites

The test compositions you will use must already exist on the CloudTest instance that you specify and you must use the correct SOASTA Repository path to invoke them. The test composition specifies the device(s) that it will run on.

This tutorial will call two such test compositions. All steps are performed from the same Jenkins job that will use either Static Instrumentation or Dynamic Instrumentation.

For Static Instrumentation the Jenkins workflow is

- Retrieve the project source code using Git (or another SCM tool)
- Make the app touchtestable
- Build the app (or app and ipa) for the specified devices/simulators using an Execute Shell step
- Deploy the app/ipa on the specified devices/simulators
- Silently play the specified compositions

For Dynamic Instrumentation the Jenkins workflow is

- Retrieve the project source code using Git (or another SCM tool)
- Build the app (or app and ipa) for the specified devices/simulators using an Execute Shell step
- Make the APP file touchtestable or generate the IPA from the APP and then make the IPA touchtestable
- Deploy the app/ipa on the specified devices/simulators
- Silently play the specified compositions

If you're a new TouchTest user, refer to the following documentation before proceeding with this tutorial.

- Basic TouchTest recording is covered in the TouchTest iOS Tutorial. Jenkins users should note the advantages of static vs. dynamic instrumentation with their own development environment in mind.

TIP: Determining whether to use dynamic instrumentation of a project file, or dynamic instrumentation of an APP or IPA file, before creating the Jenkins job will save a lot of trial and.

- The Simulator(s) or Device(s) you intend to use must be registered with your CloudTest instance. Refer to the "Registering Your Device to Use TouchTest™" section of the TouchTest iOS Tutorial for steps.
- Advanced TouchTest recording, including the use of validations and other accessors in the open source Stockfish mobile app, are covered in the TouchTest Advanced Tutorial.

TIP: The test clips shown in the result dashboards at the end of this were created using the GitHub version of Stockfish used in this guide simply by following the steps presented in the following two TouchTest Advanced Tutorial sections:

  - Create a Simple TouchTest Clip – King Gambit Declined
  - Advanced Clip Editing – Fool's Mate

About Shell Steps and the Signing/Provisioning Prerequisite

In general, the best practice for all shell steps presented in this tutorial is to first execute all of them from the command line.

With respect to signing and provisioning, you must perform the following one-time procedure from Terminal to sign each profile/application combination that is in its first use. Provisioning will not be silent until this is done.

TIP: If signing is not done, then a "User interaction not allowed" error will occur in the build. This is because the Operating System requires a user to sign an application using a specific Provisioning profile for the first time. In that case, the Operating System needs a user to authorize the signing process.

Use the following steps:

1. In Terminal, run the xcrun command from your Jenkins job manually.

Note: If you are using the Jenkins $WORKSPACE variable, you'll need to change it to a Mac OS X path to work here.

For example: "/Users/username/.jenkins/jobs/Stockfish Functional Tests/workspace"

When you do that, OS will popup a dialog box asking permission for the signing process.

2. Choose "Always allow" and this error will not show up again.

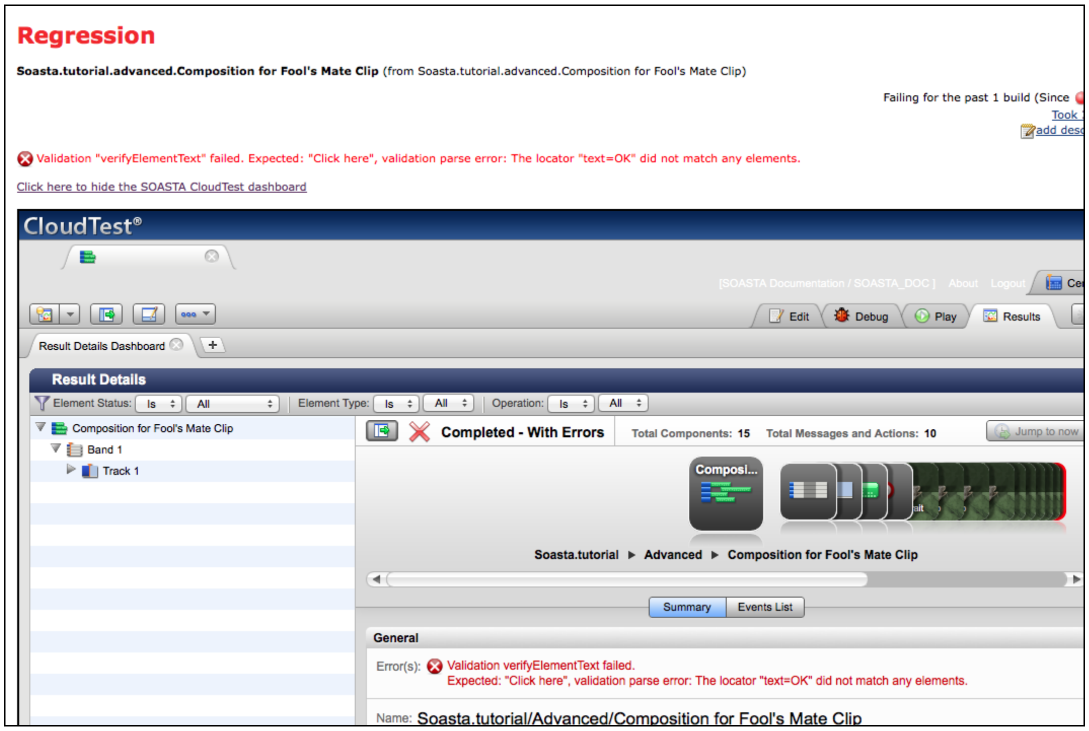
CloudTest Continuous Integration Support

SOASTA CloudTest includes first-class support for including test output in build reports for Jenkins and Hudson via the CloudTest Command-Line Client (a.k.a. the SCommand utility). Additionally, the CloudTest Jenkins/Hudson Plugin provides visual integration of CloudTest dashboards within Jenkins itself.

CloudTest outputs JUnitXML-compatible test output that is the basis for test result display in Jenkins. Since Jenkins provides out-of-the-box JUnit support these result details from a given test composition run in Jenkins will appear on the corresponding Test Results page in Jenkins.

By placing all its XML into a directory that is also provided to Jenkins, we can easily display these JUnit-friendly test results inside Jenkins. Using this configuration, a Jenkins Test Result detail page will display details from CloudTest.

While it is not necessary to install the CloudTest Jenkins/Hudson Plugin to utilize CloudTest's JUnitXML-formatted output in Jenkins in this manner, plugin installation adds the capability to also jump to specific test composition errors in the CloudTest Result Details dashboard from within Jenkins.


Regression

Soasta.tutorial.advanced.Composition for Fool's Mate Clip (from Soasta.tutorial.advanced.Composition for Fool's Mate Clip)

Failing for the past 1 build (Since

Validation "verifyElementText" failed. Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

Click here to hide the SOASTA CloudTest dashboard

Error(s): Validation verifyElementText failed.
Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

Name: Soasta.tutorial/Advanced/Composition for Fool's Mate Clip

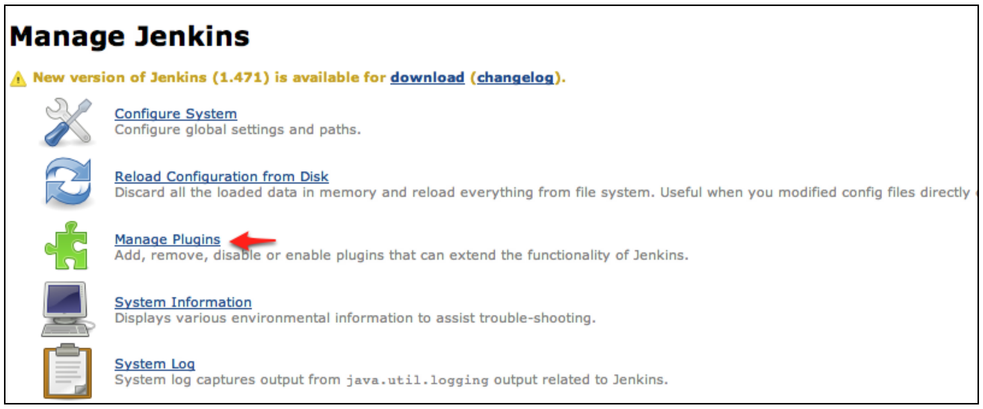
In the screenshot above, the error heading and detail text are an output of the CloudTest Jenkins Plugin, Play Composition(s) command for the test shown. The hypertext link for the CloudTest dashboard is an output of the plugin, which has already been clicked to show the CloudTest dashboard. Once the Jenkins plugin is installed, The Click here... link will appear once the Post-build action box is checked.

Clicking the link, and providing CloudTest credentials, will then display the precise failure inline in Jenkins for the given CloudTest result.

Installing the SOASTA CloudTest Jenkins/Hudson Plugin

Use the following steps to install the CloudTest Plugin from within your Jenkins instance. You must perform this installation to succeed using this tutorial.

1. In Jenkins, click Manage Jenkins, and then click Manage Plugins.



Manage Jenkins

New version of Jenkins (1.471) is available for download (changelog).

Configure System
Configure global settings and paths.

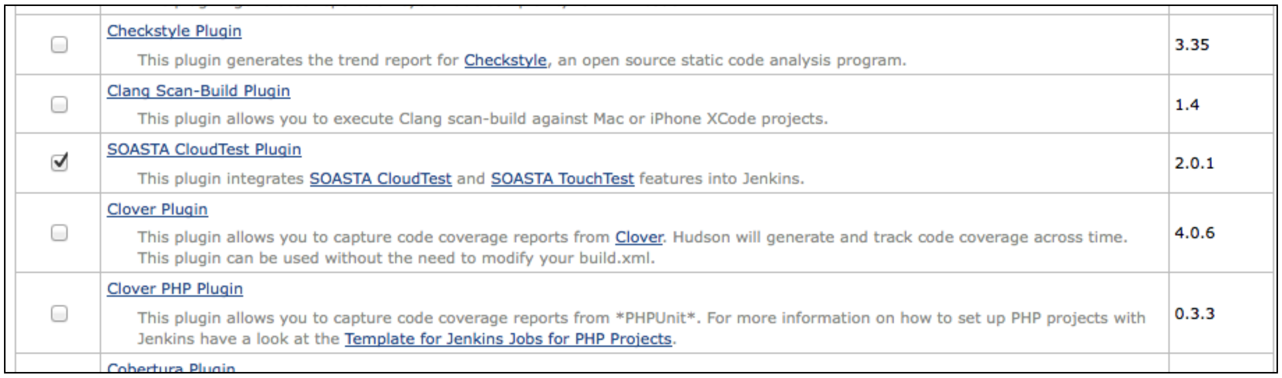
Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

System Information
Displays various environmental information to assist trouble-shooting.

System Log
System log captures output from `java.util.logging` output related to Jenkins.

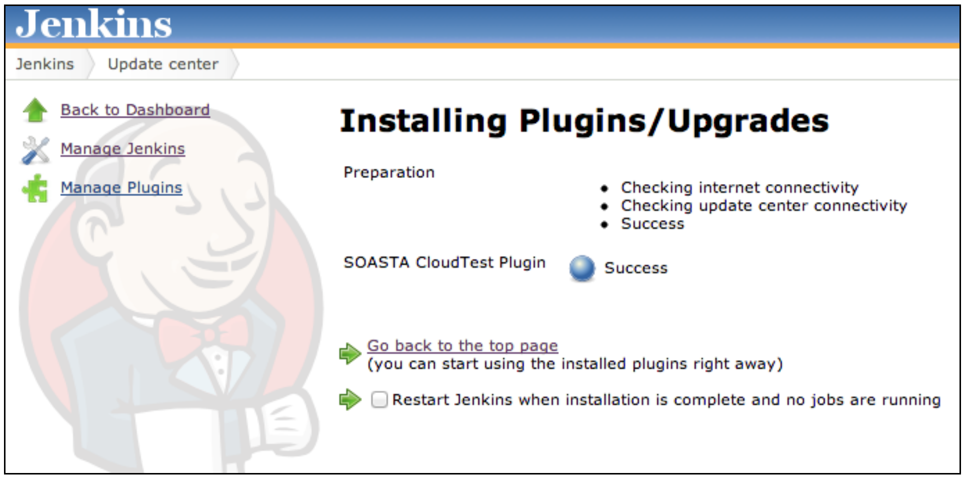
2. Click the Plugin Manager, Available tab.
3. Locate the SOASTA CloudTest Plugin in the list (using Cmd+F and "SOASTA" is a quick way to locate it).



☐	Checkstyle Plugin This plugin generates the trend report for Checkstyle, an open source static code analysis program.	3.35
☐	Clang Scan-Build Plugin This plugin allows you to execute Clang scan-build against Mac or iPhone XCode projects.	1.4
✓	SOASTA CloudTest Plugin This plugin integrates SOASTA CloudTest and SOASTA TouchTest features into Jenkins.	2.0.1
☐	Clover Plugin This plugin allows you to capture code coverage reports from Clover. Hudson will generate and track code coverage across time. This plugin can be used without the need to modify your build.xml.	4.0.6
☐	Clover PHP Plugin This plugin allows you to capture code coverage reports from *PHPUnit*. For more information on how to set up PHP projects with Jenkins have a look at the Template for Jenkins Jobs for PHP Projects.	0.3.3
	Cobertura Plugin	

4. Click the Install without Restart button at the bottom of the page.

The Installing Plugins/Upgrades page appears and indicates success once the install completes.



Installing Plugins/Upgrades

Preparation
- Checking internet connectivity
- Checking update center connectivity
- Success

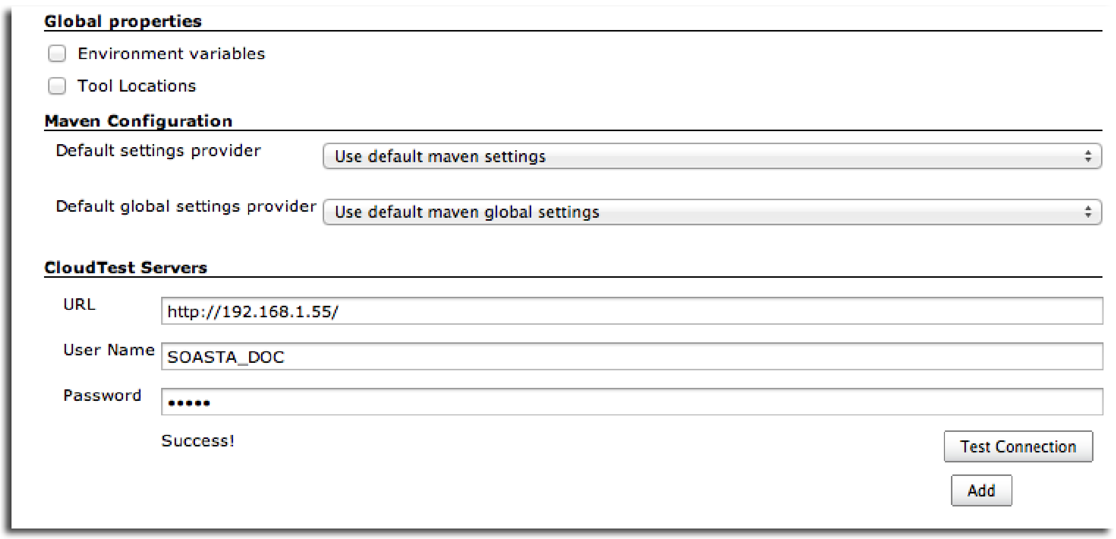
SOASTA CloudTest Plugin ● Success

Go back to the top page
(you can start using the installed plugins right away)

☐ Restart Jenkins when installation is complete and no jobs are running

Configuring the SOASTA CloudTest Jenkins/Hudson Plugin

Before using the plug-in, you will need to provide the CloudTest server information, in the Manage Jenkins > Configure System page.



Global properties

- [ ] Environment variables
- [ ] Tool Locations

Maven Configuration

Default settings provider	Use default maven settings
Default global settings provider	Use default maven global settings

CloudTest Servers

URL	http://192.168.1.55/
User Name	SOASTA_DOC
Password	•••••

Success!

1. Enter the CloudTest URL and the matching credentials. *SOASTA recommends creating a dedicated CloudTest account for Jenkins to use.*
2. Save the Configure System page before exiting.

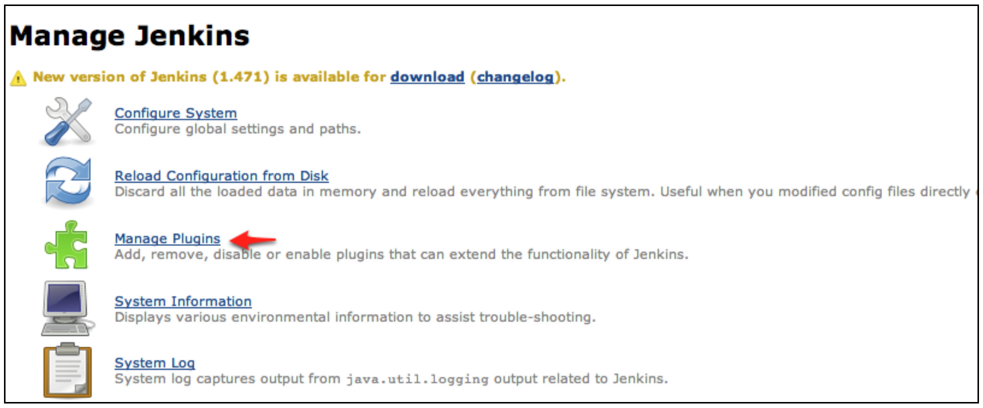
Installing the Git Plugin

Use the following steps to install the Git Plugin from within your Jenkins instance. This will allow us to use Git as a Source Code Repository to retrieve the example project, Stockfish.

TIP: Git is a distributed version control system used for software development. It is not necessary to signup or login to GitHub in order to checkout the code using the following command. In the Source Code Management section, click Git.

If you are using your own app, you can skip this requirement and then substitute the SCM tool and repository to use in the place of Git.

1. In Jenkins, click Manage Jenkins, and then click Manage Plugins.



Manage Jenkins

New version of Jenkins (1.471) is available for download (changelog).

Configure System
Configure global settings and paths.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly

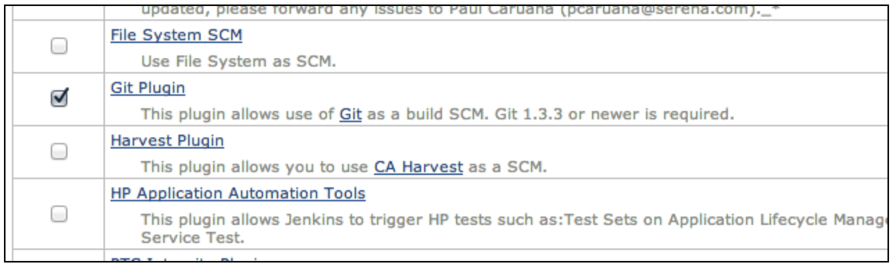
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

System Information
Displays various environmental information to assist trouble-shooting.

System Log
System log captures output from java.util.logging output related to Jenkins.

2. Click the Plugin Manager, Available tab

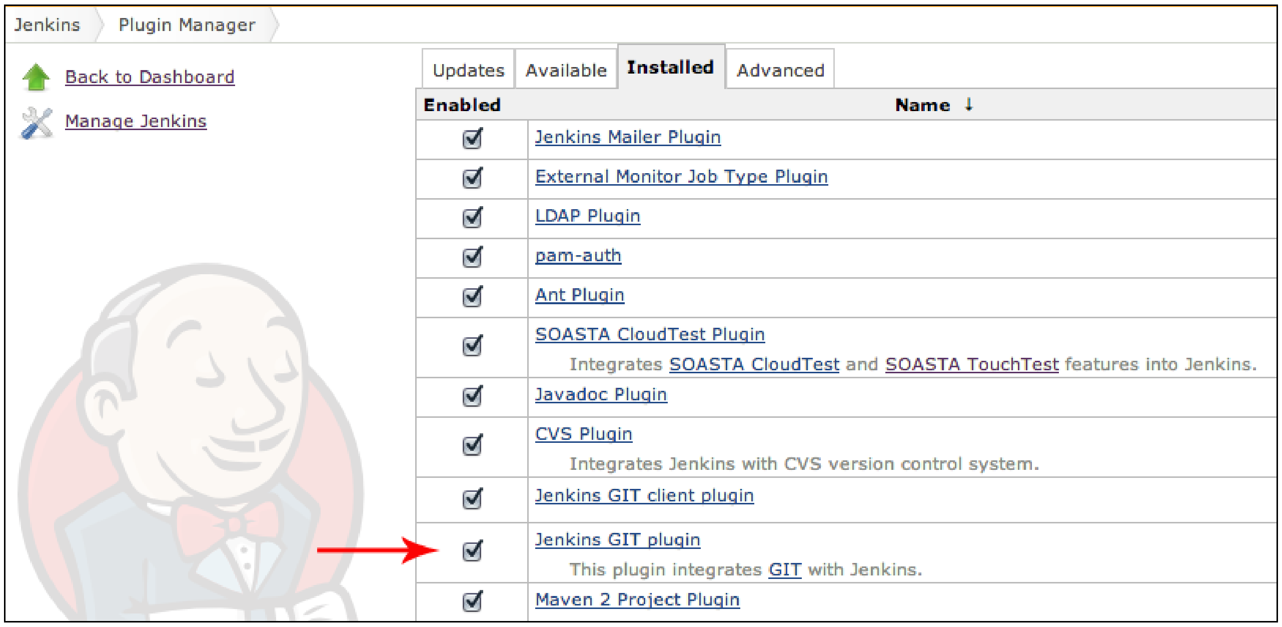
3. Locate the Git Plugin and check it as well.



updated, please forward any issues to Paul Caruana (pcaruana@serena.com)._*

☐	File System SCM Use File System as SCM.
☑	Git Plugin This plugin allows use of Git as a build SCM. Git 1.3.3 or newer is required.
☐	Harvest Plugin This plugin allows you to use CA Harvest as a SCM.
☐	HP Application Automation Tools This plugin allows Jenkins to trigger HP tests such as:Test Sets on Application Lifecycle Manag Service Test.

You can verify plugin installation on the Manage Plugins, Installed tab:



Enabled	Name ↓
✓	Jenkins Mailer Plugin
✓	External Monitor Job Type Plugin
✓	LDAP Plugin
✓	pam-auth
✓	Ant Plugin
✓	SOASTA CloudTest Plugin Integrates SOASTA CloudTest and SOASTA TouchTest features into Jenkins.
✓	Javadoc Plugin
✓	CVS Plugin Integrates Jenkins with CVS version control system.
✓	Jenkins GIT client plugin
✓	Jenkins GIT plugin This plugin integrates GIT with Jenkins.
✓	Maven 2 Project Plugin

Note: Some users may encounter a "Failed to connect to repository" error when using the Jenkins Git Plugin. If this occurs, set the plugin's path (i.e. /usr/local/git/bin/git or wherever the executable is located) to your git executable in the Manage Jenkins > Configure System > Git > Git Installations > Installation directory field.

Alternately, you can use an Execute Shell step in place of installing the plugin that will call the git command; however, this also requires the correct path to the executable. The script in the step would be:

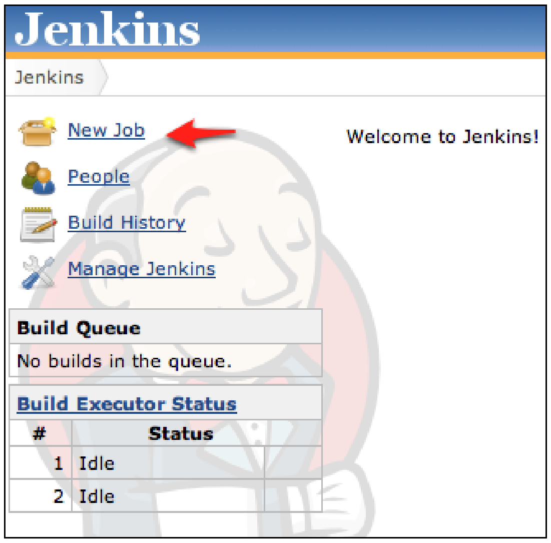
```
#Download Stockfish from GitHub

/usr/local/git/bin/git clone
https://github.com/elitecoder/stockfishchess-ios
```

Creating a Jenkins Job

The Jenkins job created below will run on the Mac machine with Xcode and one or more properly provisioned USB-attached devices. The job will have build steps that the CloudTest Jenkins Plug-in Command to download a project from source, run the MakeAppTouchTestable utility on that project, build and deploy using the deploy script, and call SCommand to silently play a list of compositions.

TIP: Your Jenkins workflow will differ based on the MATT instrumentation methods you have decided to use. You can review them a second time in the TouchTest iOS Tutorial

1. In top level Jenkins dashboard, click New Job.



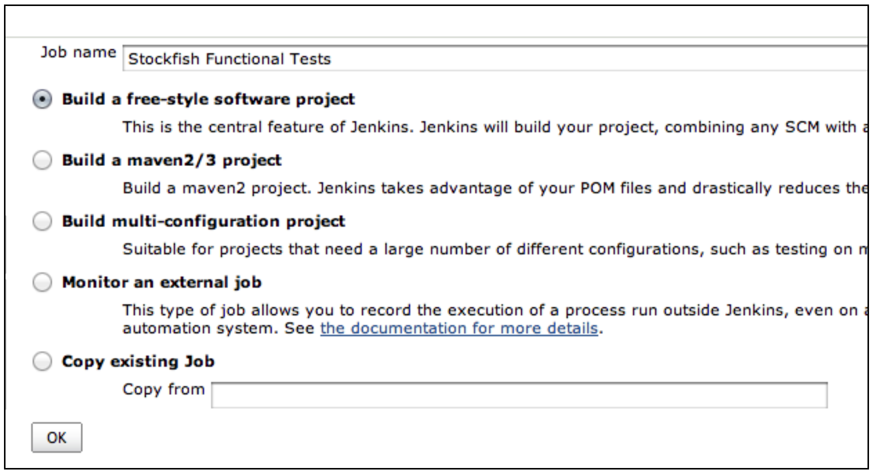
Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

The Job name page appears.



Job name Stockfish Functional Tests

[✓ Build a free-style software project]

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with a

Build a maven2/3 project

Build a maven2 project. Jenkins takes advantage of your POM files and drastically reduces the

Build multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on m

Monitor an external job

This type of job allows you to record the execution of a process run outside Jenkins, even on a automation system. See the documentation for more details.

Copy existing Job

Copy from

OK

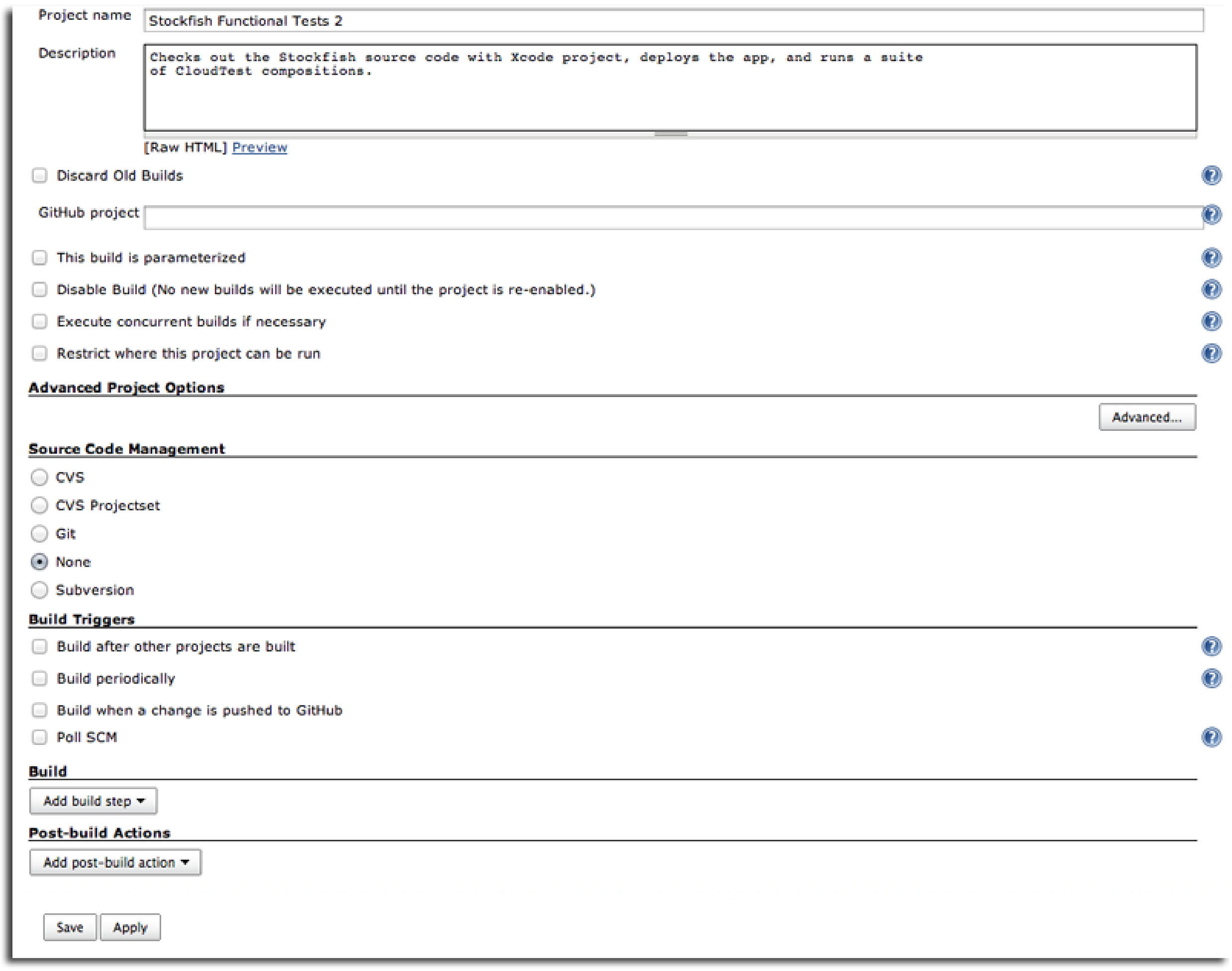
1. Enter a job name and check the first option, "Build a free-style software project".

The Job Details page appears.

2. Enter a description for this job:

- For example, "Checks out the Stockfish source code with Xcode project, deploys the app, and runs a suite of CloudTest compositions."

Leave the Job open and continue with the following section.



Project name: Stockfish Functional Tests 2

Description:
```
Checks out the Stockfish source code with Xcode project, deploys the app, and runs a suite
of CloudTest compositions.
```
[Raw HTML] Preview

- [ ] Discard Old Builds

GitHub project:

- [ ] This build is parameterized
- [ ] Disable Build (No new builds will be executed until the project is re-enabled.)
- [ ] Execute concurrent builds if necessary
- [ ] Restrict where this project can be run

Advanced Project Options

Advanced...

Source Code Management

- ○ CVS
- ○ CVS Projectset
- ○ Git
- ● None
- ○ Subversion

Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Build when a change is pushed to GitHub
- [ ] Poll SCM

Build

Add build step

Post-build Actions

Add post-build action

Save Apply

Jenkins Workflows for TouchTest

The CloudTest Jenkins Plugin is used to do all tasks in the Jenkins job, with the exception of Build tasks, which are performed using xcodebuild and xcrun.

Because there are different possible Jenkins job workflows, the *possible* steps are presented *a la carte.* For each Jenkins job there will be (minimally):

- o A First step; used to retrieve the source project (all workflows)
- o In between the first and last steps, each Jenkins job will have:
    - ▪ A CloudTest Plugin, MakeAppTouchTestable step;
    - ▪ with one or more Build steps,
    - ▪ and one or more Install/Run steps
- o A Last step; to Play the Composition

Note: Provisioning and codesigning can be performed either using the CloudTest Jenkins Plugin, MATT's Advanced steps to enter the APP/IPA optional parameters (refer to the relevant sections). Or, these steps can be done using xcrun.

There is a one-time provisioning profile step that should be run prior to running the complete Jenkins job.

As discussed in the TouchTest iOS Tutorial,

- Static instrumentation applies MATT to the Xcode project file
- Dynamic instrumentation applies MATT to the APP or IPA

In general, you will delay the MATT step only until the last logical point (i.e., only after the GIT if you're using static instrumentation, after the APP step if for dynamic using a simulator(s), and after the IPA step for dynamic using physical devices.

Static Instrumentation of an Xcode Project

In this workflow, you'll apply MATT to the Xcode project itself, using the `project` parameter. After which, you'll add the steps necessary to build and install to your simulators and devices.

- *Apply MATT* using the `project` parameter (static only)
- *Build the APP* with an Execute Shell step using xcodebuild
- (optional) *Build the IPA* with an Execute Shell step using xcrun
- Either *Run App in iOS Simulator* or *Install iOS App on Device* (using IPA)

Dynamic Instrumentation of an APP file

In this workflow, you'll apply MATT to the compiled APP file using the `appbundle` parameter. If you're also deploying to physical devices, you'll need to mix and match steps to do both, keeping in mind to build the APP first.

- *Build the APP* with an Execute Shell step using xcodebuild
- *Apply MATT* to that compiled APP using `iOS APP Bundle`
- *Run App on iOS Simulator* command

Dynamic Instrumentation of an IPA file

In this workflow, you'll delay applying MATT until the IPA is created

- *Build the APP* with an Execute Shell step using xcodebuild
- *Build the IPA* file with an Execute Shell step using xcrun,
- *Apply MATT* to that compiled IPA using `ipa`
- Install App on iOS Device command.

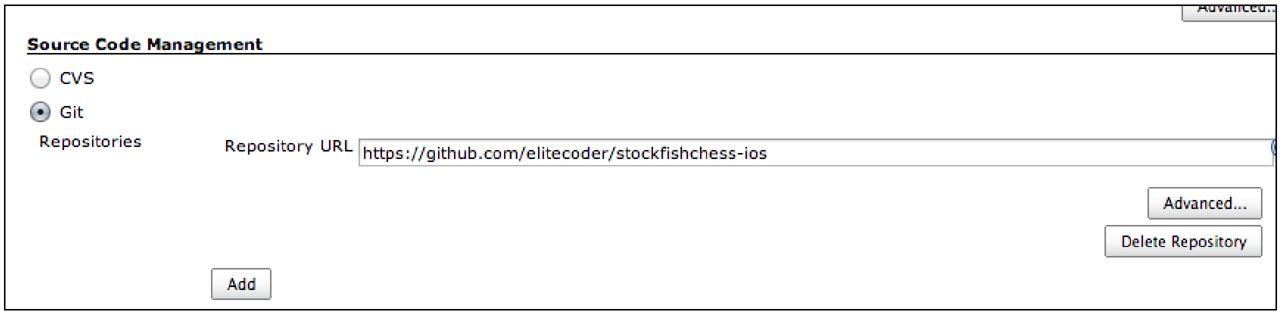
In the remainder of the Job Creation steps, mix and match the tasks that you need to build your Jenkins job.

Get Stockfish using the Git Plugin

Next, we will add a step that will get the Stockfish Xcode project that will be used in the remainder of this tutorial. The following instructions use the Jenkins Git Plugin (installed above). If you are using your own Source Code Management system simply select its type and enter its repository URL, as you would normally do.

With the Git Plugin installed in our Jenkins instance, we will add the Source Code Management step as we would with any SCM tool.

1. In the Source Code Management section, check the Git radio button.



Source Code Management

CVS [✓ Git]

Repositories — Repository URL: https://github.com/elitecoder/stockfishchess-ios

Note: You can specify your own SCM tool and Repository URL here.

2. Enter the repository URL in the entry field:

```
https://github.com/elitecoder/stockfishchess-ios
```

Using CloudTest Plugin, MakeAppTouchTestable

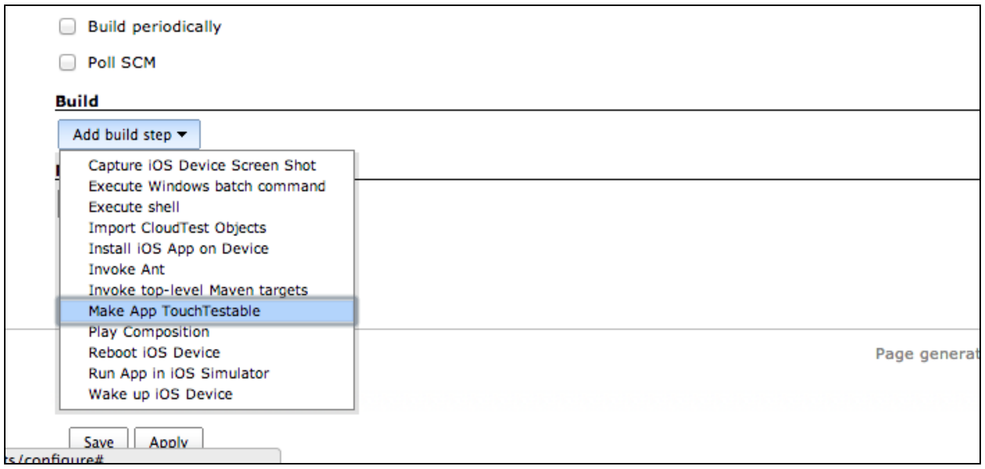
The CloudTest Jenkins Plugin wraps the MakeAppTouchTestable commands into a convenient set of commands that automate the instrumentation steps. Use these commands. The MATT section is presented first, although it can be anywhere between the first and last steps in your job as described above.

In the following sections, we will use the CloudTest Jenkins Plugin's MakeAppTouchTestable module to make an iOS project, a compiled APP file, and finally an IPA file, TouchTestable. The plugin's MATT forms also provide a means to use existing MATT commands via the command line.

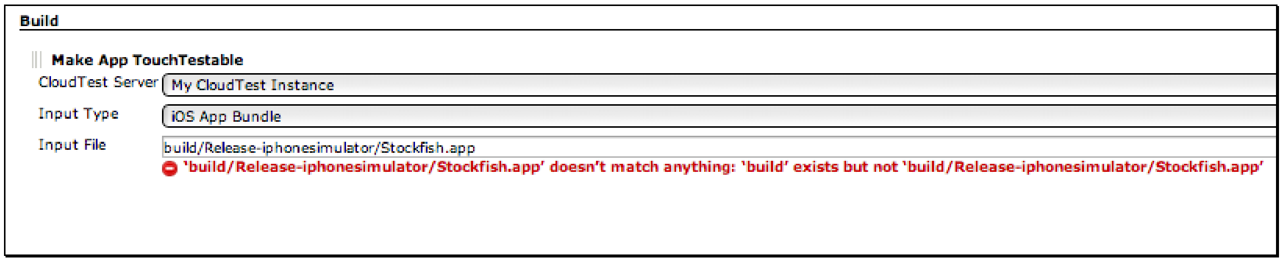
Using MATT on an APP (Dynamic Instrumentation for Simulator)

Use the following steps to dynamically instrument an APP file. This is typically done before it is run on a simulator. For example, you can use this step after an Execute Shell step using xcodebuild.

1. Add a MakeAppTouchTestable step to the job.



2. Apply MATT to the compiled APP by selecting the Input Type, iOS App Bundle (e.g. the MATT equivalent is the `appbundle` parameter).



Build

Make App TouchTestable

CloudTest Server: My CloudTest Instance

Input Type: iOS App Bundle

Input File: build/Release-iphonesimulator/Stockfish.app

'build/Release-iphonesimulator/Stockfish.app' doesn't match anything: 'build' exists but not 'build/Release-iphonesimulator/Stockfish.app'

1. Enter the APP name as the Input File (from the workspace root).

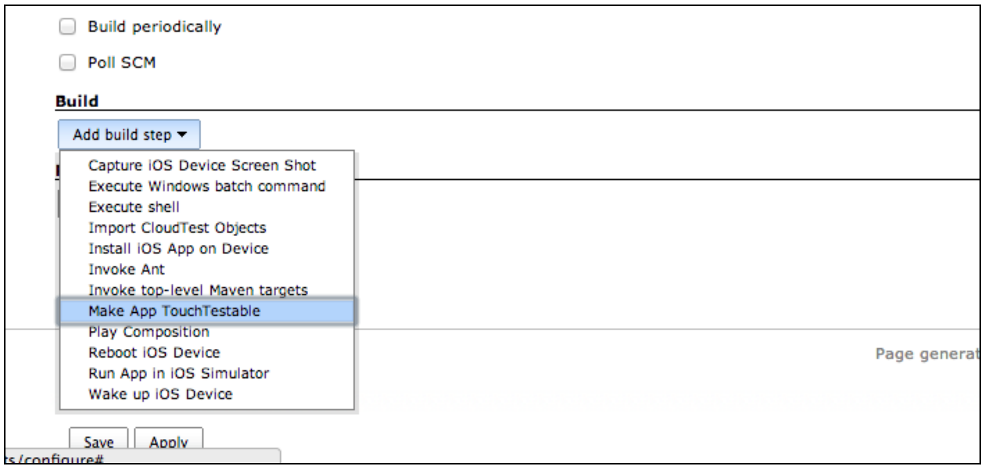
2. Optionally, click Advanced to display additional MATT configuration fields.

3. Save the Jenkins job.

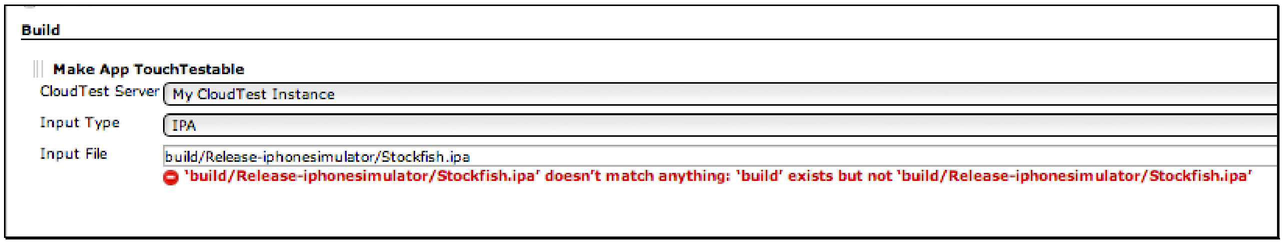
Using MATT on an IPA (Dynamic Instrumentation for Device)

Use the following steps to dynamically instrument an IPA file. This is typically done as the last step before it is installed on a device. For example, you can use this step after an Execute Shell step using xcrun.

3. Add a MakeAppTouchTestable step to the job.



4. Apply MATT to the compiled IPA by selecting the Input Type, IPA (e.g. the MATT equivalent is the `ipa` parameter).



Build

Make App TouchTestable

CloudTest Server: My CloudTest Instance

Input Type: IPA

Input File: build/Release-iphonesimulator/Stockfish.ipa

'build/Release-iphonesimulator/Stockfish.ipa' doesn't match anything: 'build' exists but not 'build/Release-iphonesimulator/Stockfish.ipa'

4. Enter the IPA name as the Input File (from the workspace root).

5. Optionally, click Advanced to display additional MATT configuration fields.

- **Launch URL** – Same as MATT `launchurl`. For example: `my-app://launch`

TIP: If you are specifying a custom Launch URL here, be sure to avoid spaces and underscores, as they will cause an error.

- **Back up modified files** – Check this to keep backups in the project).
- **Additional options** - Enter any additional MATT command line parameters.

  Most notably, you can use MATT to provision and code sign the dynamically instrumented IPA file (code signing and provisioning can, of course, be done using xcrun, which is discussed in the Execute Shell step, Building the IPA for a Device).

  Use the following MATT optional IPA parameters

  - `-provisioningprofile <profilepath>` – Path of the Provisioning profile to be used for building IPA file. The provisioning profile you input MUST to be a Distribution profile.
  - `-signingidentity <signingidentityname>` – Name of the signing identity to be used for codesigning the application. (e.g. "iOS Distribution: Developer Name")
  - `-entitlementsfile <entitlementsfilepath>` - path of the entitlements file to be used for codesigning the application

For more about using additional MATT parameters, use:

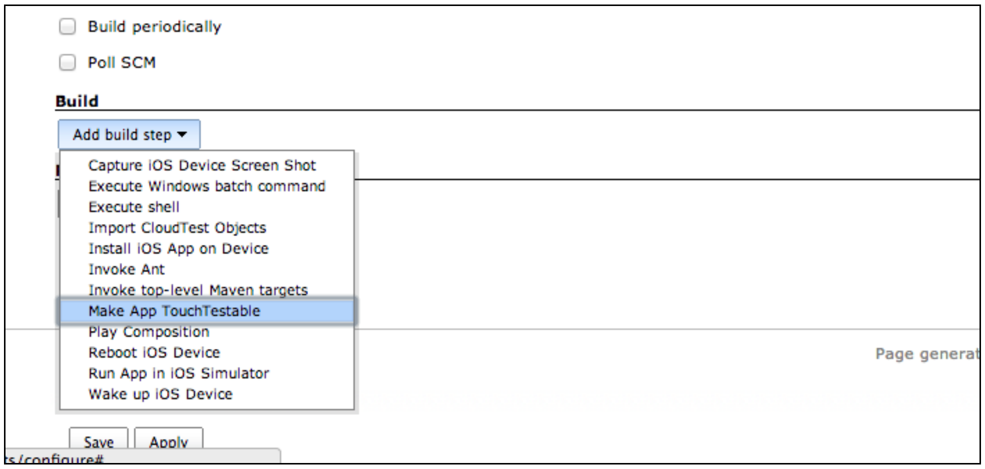
```
java sh MakeAppTouchTestable/bin/MakeAppTouchTestable -help
```

6. Save the Jenkins job.

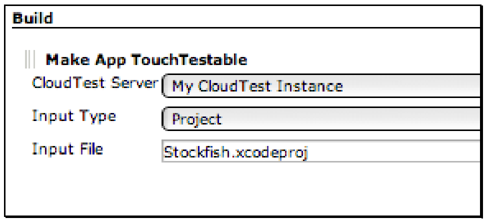
Using MATT on a Project (Static Instrumentation for Simulators/Devices)

This utility is automatically downloaded by the SOASTA CloudTest Jenkins Plugin and can be easily specified in your Jenkins job using the following steps.

7. Click Add a Build Step, and select MakeAppTouchTestable from the drop-down list. This MakeAppTouchTestable step will always be the first step in a Static Instrumentation workflow, but will come after the APP or IPA is built where those files are in use on a simulator(s) or device(s).



The Make App TouchTestable form appears.



CloudTest Server	My CloudTest Instance
Input Type	Project
Input File	Stockfish.xcodeproj

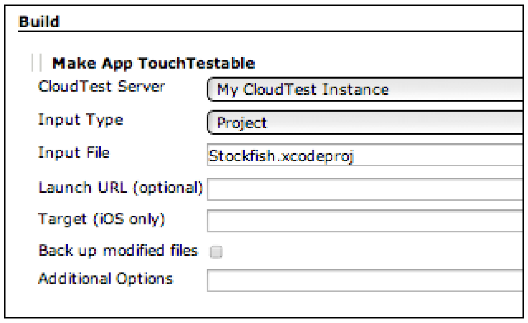
8. Enter the Xcode project name as the Input File.

9. Optionally, click Advanced to display additional MATT configuration fields.

- **Launch URL** – Same as MATT `launchurl`. For example: `my-app://launch`

TIP: If you are specifying a custom Launch URL here, be sure to avoid spaces and underscores, as they will cause an error.

- **Target** – Same as MATT `target`. For example: `Stockfish copy`


Build

Make App TouchTestable

CloudTest Server	My CloudTest Instance
Input Type	Project
Input File	Stockfish.xcodeproj
Launch URL (optional)	
Target (iOS only)	
Back up modified files	☐
Additional Options	

- **Back up modified files** – Check this to keep backups in the project folder (where .xcodeproj resides).

- **Additional options** - Enter any additional MATT command line parameters. For more about using additional MATT parameters, use:

```
java sh MakeAppTouchTestable/bin/MakeAppTouchTestable -help
```

10. Save the Jenkins job.

Build an APP or IPA using Execute Shell

In the following sections, we will build the app, and generate an IPA.

Note: If an app for simulator is also included in your Jenkins job alongside one for a real device, it should be built and run *prior* to building the app for a device (described in the following steps). This ordering is necessary to prevent the example mobile app for a simulator from being overwritten by the subsequent version of the app intended for the physical device.

Using the Jenkins $WORKSPACE Variable

In the Execute Shell sections below, we will build the mobile app for both a simulator and a device using the Jenkins $WORKSPACE variable. This variable is a shortcut to the Jenkins folder's workspace sub-folder. For example:

/Users/username/.jenkins/jobs/Stockfish Functional Tests/workspace

The Jenkins $WORKSPACE variable is used as a matter of convenience. It is not required. When using $WORKSPACE in an Execute Shell step it is a good idea to place it into double-quotes (e.g. "$WORKSPACE"). This is in case the path includes spaces. For example, if the name of your job includes spaces so will your workspace.

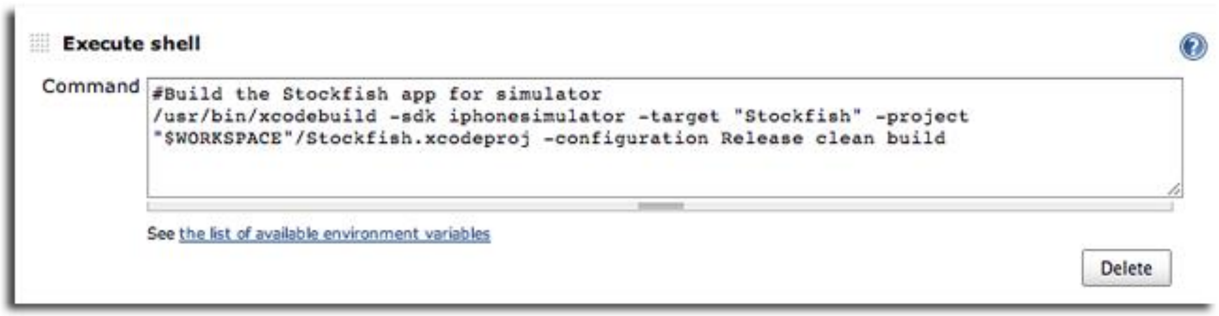
Build the APP for Simulator (or Before Building an IPA)

The app file is a requirement for deployment to all iOS simulators and devices. For iPhone and iPad devices, the app file is then used to generate an IPA archive for deployment.

In order to build the app, the command line `xcodebuild` command is used.

1. Click Add a Build Step, and select Execute Shell from the drop-down list. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.
2. Enter the following lines in the end of the Execute Shell field (revise the paths to match that of your own environment):

```
#Build the Stockfish app for simulator
/usr/bin/xcodebuild -sdk iphonesimulator -target "Stockfish" -project
"$WORKSPACE"/Stockfish.xcodeproj" -configuration Release clean build
```



```
#Build the Stockfish app for simulator
/usr/bin/xcodebuild -sdk iphonesimulator -target "Stockfish" -project
"$WORKSPACE"/Stockfish.xcodeproj -configuration Release clean build
```

where:

- `<sdk>` Either iphonesimulator or iphoneos.
- `<target>` is the name of the target in the ".xcodeproj" file representing your project.

Note: When building an app for a simulator, specify the SDK as an argument (recommended).

- `<project>` is the name of the ".xcodeproj" file to build.
- `<configuration>` is the type of build. Refer to `/usr/bin/xcodebuild -help` for more information.

Next Steps:

- After the Stockfish.app has been successfully created, make it TouchTestable, if that is the last step before running it on a simulator.
- You can also build the Xcode project *before* applying MATT, in which case you will use Add Build Step, MakeAppTouchTestable, iOS APP Bundle to make it TouchTestable before you install to a simulator.

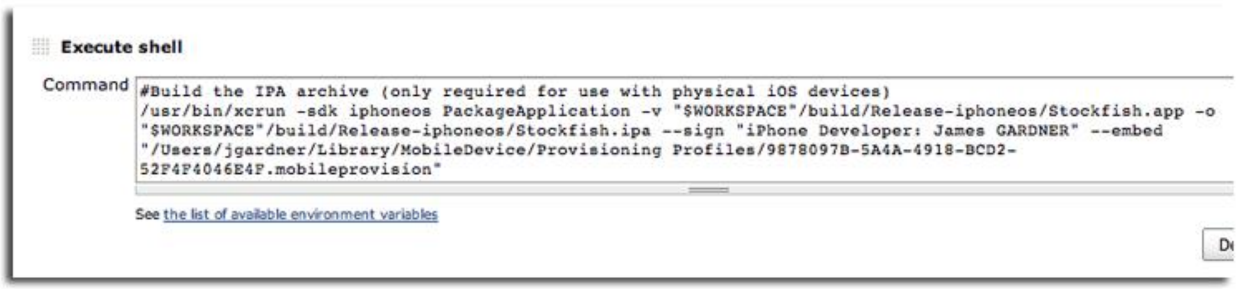
Build the IPA for a Device (Static or Dynamic)

Now, we'll add an Execute Shell step to build the IPA archive file using the Xcode Tools command, xcrun. In the command below, we will also sign the application using a provisioning profile. You'll need the correct provisioning profile and iPhone Developer or iPhone Distribution developer ID to succeed here.

Building the IPA requires the built app (see above steps) and the following command (be sure to revise the command for your own environment):

1. In the Execute Shell step, enter the following (be sure to revise this example to use your own paths).

```
#Build the IPA archive (only required for use with physical iOS devices)

/usr/bin/xcrun -sdk iphoneos PackageApplication -v
"$WORKSPACE"/build/Release-iphoneos/Stockfish.app -o
"$WORKSPACE"/build/Release-iphoneos/Stockfish.ipa --sign "iPhone
Distribution: Username" --embed
"/Users/username/Library/MobileDevice/Provisioning Profiles/20454FCB-7B8C-
4996-BDD3-E660DB279261.mobileprovision"
```



```
#Build the IPA archive (only required for use with physical iOS devices)
/usr/bin/xcrun -sdk iphoneos PackageApplication -v "$WORKSPACE"/build/Release-iphoneos/Stockfish.app -o
"$WORKSPACE"/build/Release-iphoneos/Stockfish.ipa --sign "iPhone Developer: James GARDNER" --embed
"/Users/jgardner/Library/MobileDevice/Provisioning Profiles/9878097B-5A4A-4918-BCD2-
52F4F4046E4F.mobileprovision"
```

where:

- `<sdk>` in this case the SDK to use is 'iphoneos'.
- `<output>` or `-o`. The output path and filename are where the IPA archive is created.
- `<sign>` with the Code Signing Identity found in the Xcode Organizer's Teams node. When deploying to a device, the iPhone Distribution identity is generally used (required for App Store submissions, etc.). You can also open your Xcode project and navigate to the Build Settings tab, Code Signing section to view the Code Signing Identity.

Note: In general, you want one and only one provisioning profile each for the iPhone Developer and iPhone Distribution identities. Apple warns against multiple provisioning profiles, which is known to cause Xcode conflicts, and which may in some cases prevent your Jenkins job from succeeding. If you run into Code Signing problems it may be necessary to resolve your setup before continuing. Refer to the Best Practices for Maintaining Certificates and Provisioning Profiles article in the iOS Developer Library for more information.

- `<embed>` the file name and extension of the mobile provision to use. The provisioning profiles are found in the Xcode Organizer, Library, Provisioning Profiles node. Select the profile in the Organizer window's Library section and then right-click to choose "Reveal Profile in Finder." Specify the path and file name with extension in your xcrun command. You can also locate the profile using Finder's Go To Folder command using: "~/Library/MobileDevice/Provisioning Profiles".

Note: In some cases, usually on the occasion of an Xcode update, users have encountered the following error while using xcrun to sign the app:

```
/var/folders/d9/swbmcb491lzcnl2kk1rlbhx80000gn/T/xzFYPWzzYi/Payl
oad/Stockfish.app: object file format unrecognized, invalid, or
unsuitable
```

If this error is encountered, try adding the following symbolic link (it should only be necessary to do so once):

```
export CODESIGN_ALLOCATE="/Applications/Xcode.app/Contents/Devel
oper/usr/bin/codesign_allocate"
```

Run `/usr/bin/xcrun -help` for more information about this Xcode command-line utility.

2. Click Apply on the Job page.

Once the IPA archive has been created, we are ready to deploy our mobile app to any tethered, provisioned iPad or iPhone.

Next Steps:

- After the Stockfish.ipa has been successfully created, make it TouchTestable, before installing it on a device.
- If the IPA was static instrumented many steps before now, you are ready to install it to a device.

Install to a Simulator or Device

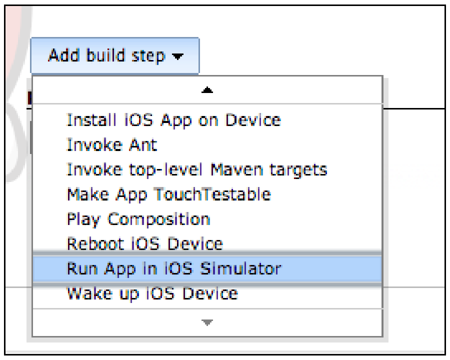
In the following two sections, we will use the CloudTest Jenkins Plugin, Run App in iOS Simulator and Install iOS App on Device commands to deploy the newly TouchTestable apps.

TIP: In Jenkins jobs where an app will be generated for BOTH a simulator and a physical device, it is necessary to generate and run the simulator app first to prevent it from being overwritten by the subsequent version of the app intended for the physical iOS device.

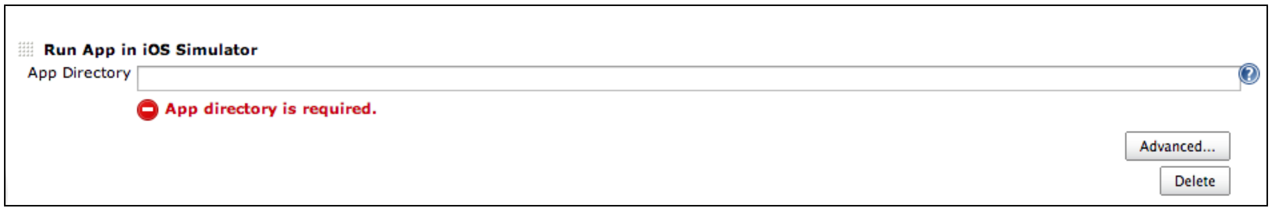
Run the App File on a Simulator

Once an App file has been created, and been made TouchTestable, whether by the static project or dynamic instrumentation using the iOS App Bundle command, use the following steps to run it on a simulator.

1. Click Add a Build Step, and select Execute Shell from the drop-down list. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.
2. Select Run app in iOS Simulator.



The Run App in iOS Simulator form appears.

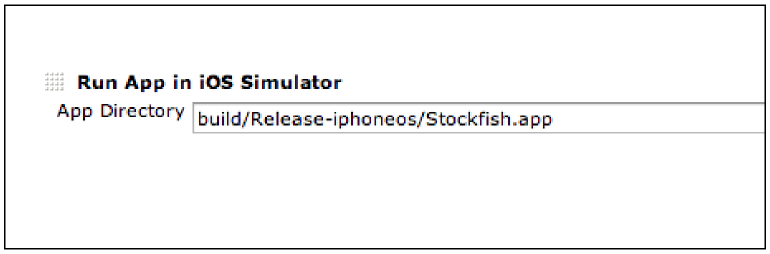


Run App in iOS Simulator

App Directory

App directory is required.

3. Enter the app folder path. In this case, use `build/Release-iphoneos/Stockfish.app`.


Run App in iOS Simulator

App Directory: build/Release-iphoneos/Stockfish.app

TIP: The four types possible here are Release-iphoneos, Release-iphonesimulator, Debug-iphoneos, Debug-iphonesimulator.

4. Click Apply on the Job page.

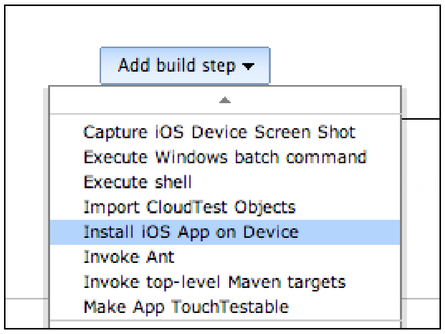
Next Steps:

- Once your mobile app is on the Simulator, the Play Composition step is added.

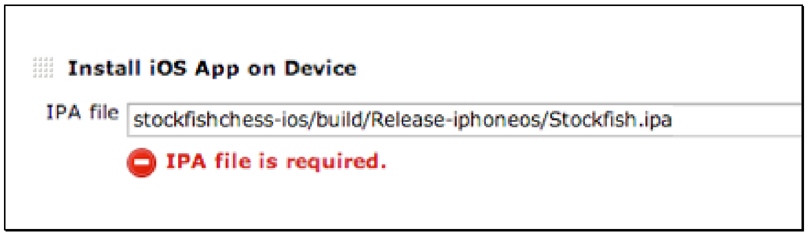
Install the IPA on a Device (Static or Dynamic)

Once an IPA file has been built, make it TouchTestable using CloudTest Jenkins Plugin, MATT, IPA. After which, you can use the following CloudTest Plugin steps to install the TouchTestable IPA onto a physical device.

1. Click "Add Build Step". Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.
2. Select Install iOS App on Device.


Add build step
Capture iOS Device Screen Shot
Execute Windows batch command
Execute shell
Import CloudTest Objects
Install iOS App on Device
Invoke Ant
Invoke top-level Maven targets
Make App TouchTestable

The Install App in iOS Device form appears.


Install iOS App on Device

IPA file: stockfishchess-ios/build/Release-iphoneos/Stockfish.ipa

IPA file is required.

3. Enter the app folder path. For example, `stockfishchess-ios/build/Release-iphoneos/Stockfish.ipa`.

Note: The alert indicating that the IPA path doesn't match appears in those cases where the directory doesn't exist before the build actually happens. Since this job builds before installing the app, the folder will exist when the Install iOS App on Device step runs, and the alert can be ignored.

4. Click Apply on the Job page.

5. Save the Jenkins job.

Next Steps:

- Once your mobile app is on the Simulator, the Play Composition step is added.

TouchTest Agent for iOS 9+

Those using iOS 9 or later will need to use the TouchTest Agent app. This app facilitates continuous integration on iOS 9 or later devices.

There are **two ways** to get the TouchTest Agent app.

Get the TouchTest Agent IPA by Running MakeAppTouchTestable Utility

Use the following command:

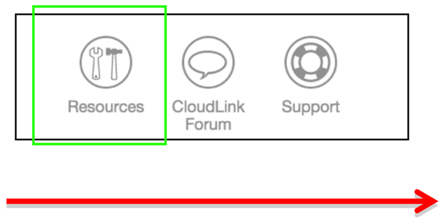
```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -touchtestagent -
output ~/tmp -provisioningprofile
~/Library/MobileDevice/Provisioning\
Profiles/example.mobileprovision -signingidentity "My Identity" -
appidprefix DW4ETSG5SA
```

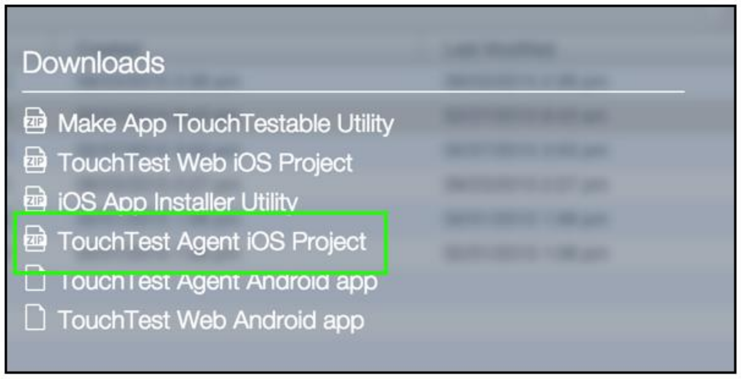
Parameters:

- `-provisioningprofile <profilepath>` - Path of the Provisioning profile to be used for building IPA file. The provisioning profile you input MUST to be a Distribution profile.
- `-signingidentity <signingidentityname>` - Name of the signing identity to be used for codesigning the application. (e.g. "iOS Distribution: Developer Name")
- `-appidprefix <Application Identifier Prefix>` - Application Identifier Prefix of the Provisioning Profile (or Team Identifier).

Get the TouchTest Agent Project from Resources Page

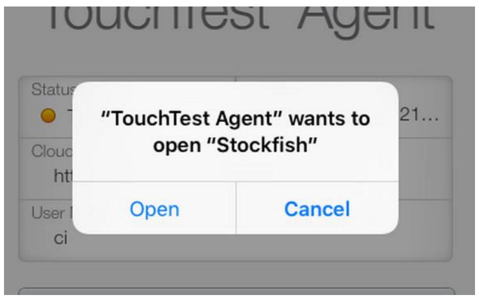
Click **TouchTest Agent iOS Project** under TouchTest Resources page, Dowloads. Once this downloads, open the TouchTest Agent iOS Project in Xcode, select your device, and hit play.

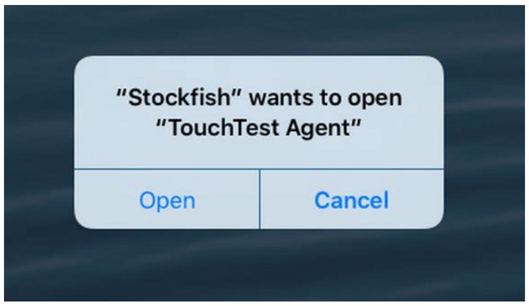



Downloads
Make App TouchTestable Utility
TouchTest Web iOS Project
iOS App Installer Utility
TouchTest Agent iOS Project
TouchTest Agent Android app
TouchTest Web Android app

Note: For iOS 9 or later, any time your device launches a new application (first time only), you need to tap **Open** twice; first, to allow TouchTest Agent to open the new app and second, to allow the app to open TouchTest Agent at the end of the session.

These actions will resemble the pop-ups below:

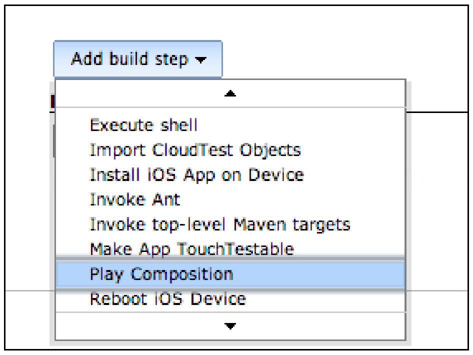




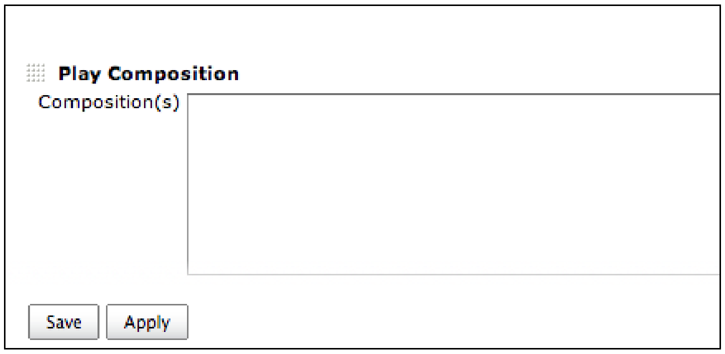
Playing the Composition

Finally, we will add a build step using the CloudTest Jenkins Plugin, Play Composition command.

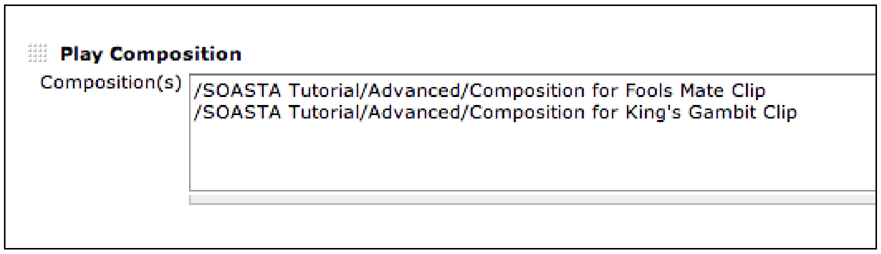
1. Add a build step and select Play Composition from the drop down.



TIP: If you are adding more than one composition, click the down arrow to expand the entry field before entering the first composition path.



2. Enter each composition to play using its full SOASTA Repository path (shown below).



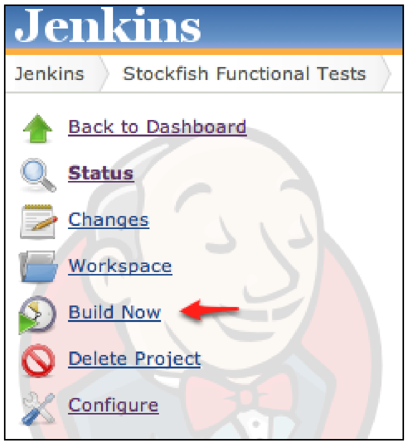
```
/SOASTA Tutorial/Advanced/Composition for Fools Mate Clip
/SOASTA Tutorial/Advanced/Composition for King's Gambit Clip
```

Note: Ensure that you have dragged all the steps into the right order before building. The order should be the same as in this tutorial, but may vary slightly depending on whether you're using devices, simulators, or both.

3. Click Save to complete the Jenkins job.

Building the Project

1. To build the project, click the "Build Now" link.



The build will start. After a short delay, you should see a progress bar appear on the left side of the page. Click this progress bar to watch the build process "live" in the Console view.

You should see the following happen:

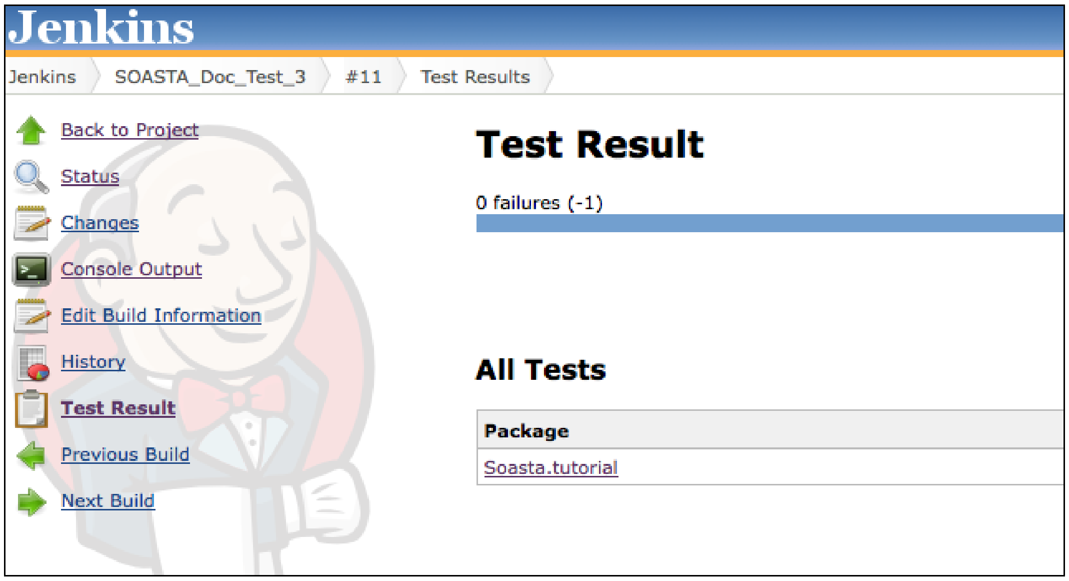
a. Jenkins checks out the source code from Git, and runs the MakeAppTouchTestable utility.

b. Jenkins performs whatever Execute Shell steps you've placed to invoke Xcode to build an app for Simulator, an app and IPA for a physical device, or both. The Run steps for either a device or simulator run consecutively after the shell that pertains to either one.

c. Jenkins plays the CloudTest compositions using SCommand. On the tethered device, you should see the Stockfish app launch and run through the test steps. When the test finishes, Stockfish will exit, and the SOASTA TouchTest Agent page will re-open.

Inspecting Test Results in Jenkins

For a successful test with no failures, the Test Result page merely lists the All Tests section with the given package (i.e. in this case the package equates to a CloudTest repository folder).



Jenkins

Jenkins › SOASTA_Doc_Test_3 › #11 › Test Results

- Back to Project
- Status
- Changes
- Console Output
- Edit Build Information
- History
- Test Result
- Previous Build
- Next Build

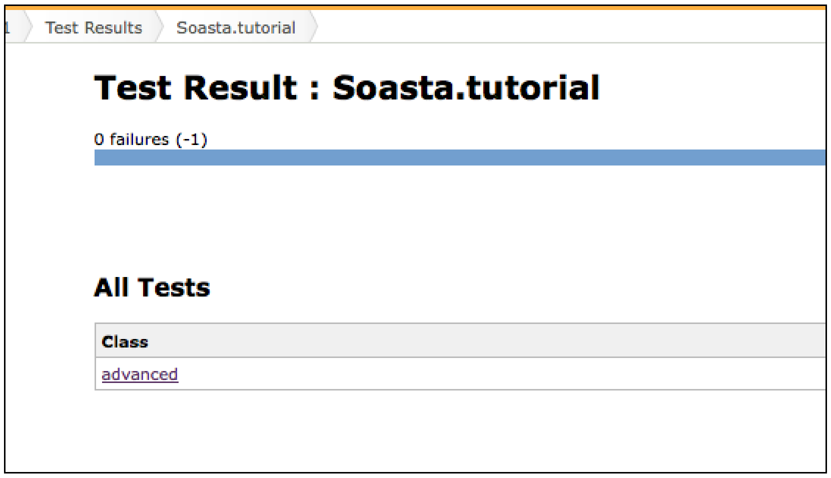
Test Result

0 failures (-1)

All Tests

Package
Soasta.tutorial

- Clicking the Package link opens the subsequent CloudTest folder.



Test Results › Soasta.tutorial

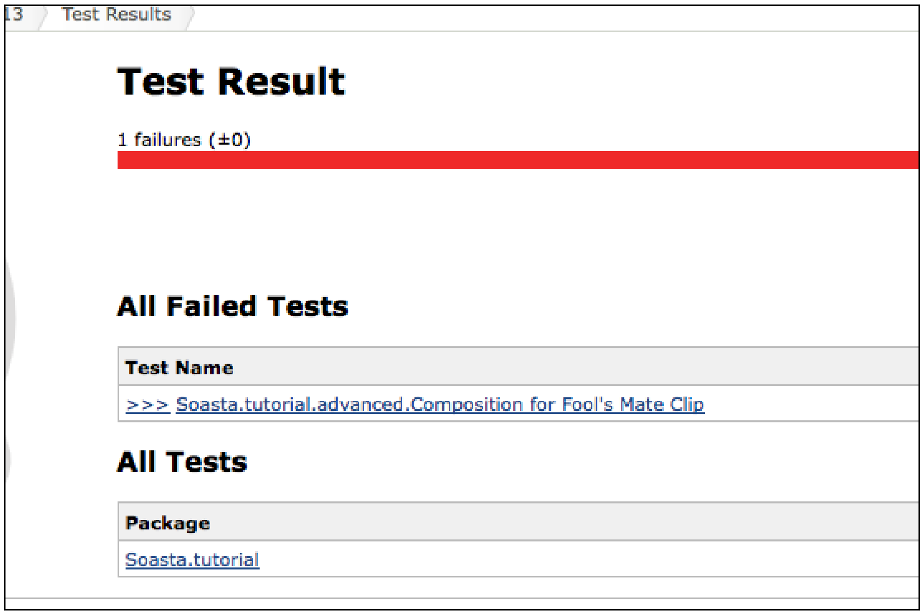
Test Result : Soasta.tutorial

0 failures (-1)

All Tests

Class
advanced

Things get more interesting when an error in the test occurs. The subsequent SCommand output is displayed (in text) on the Jenkins Test Result page (as discussed above).



Test Result

1 failures (±0)

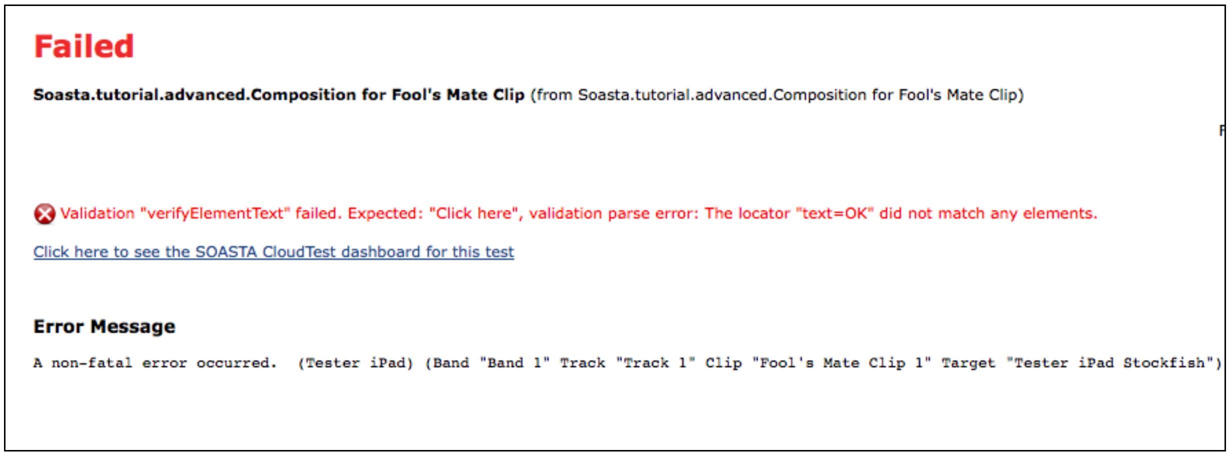
All Failed Tests

Test Name
>>> Soasta.tutorial.advanced.Composition for Fool's Mate Clip

All Tests

Package
Soasta.tutorial

In this case, the All Failed Tests section is added with the name of the test listed with a link to more of the SCommand details. Clicking the link under the Test Name section where the composition is named displays an Error detail page (for the given error).



Failed

Soasta.tutorial.advanced.Composition for Fool's Mate Clip (from Soasta.tutorial.advanced.Composition for Fool's Mate Clip)

Validation "verifyElementText" failed. Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

Click here to see the SOASTA CloudTest dashboard for this test

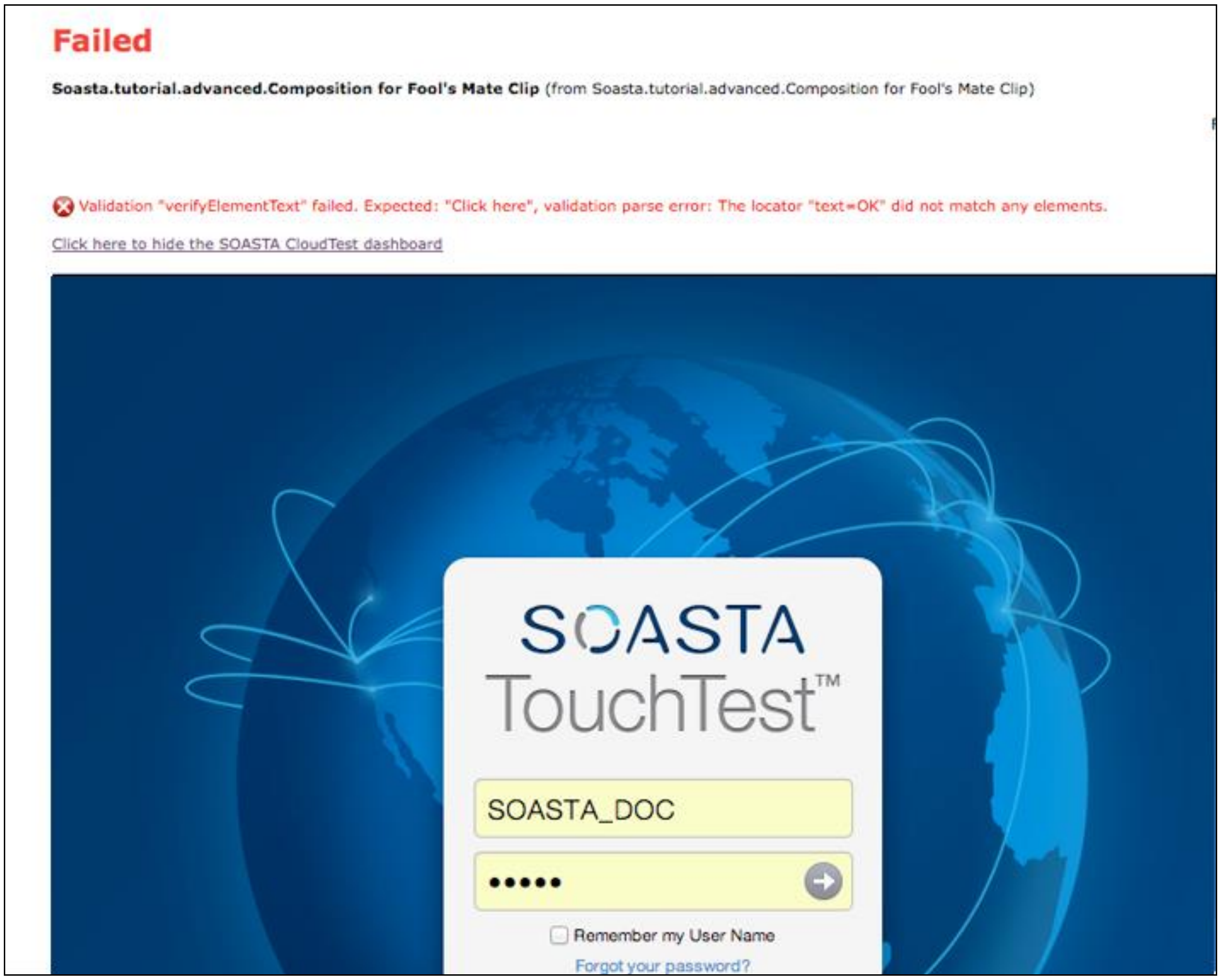
Error Message

```
A non-fatal error occurred.  (Tester iPad) (Band "Band 1" Track "Track 1" Clip "Fool's Mate Clip 1" Target "Tester iPad Stockfish")
```

In the error above a validation in the Composition for Fool's Mate has failed.

1. To view this error in the CloudTest, Result Details dashboard, click the plugin link provided (i.e. "Click here to see the SOASTA CloudTest dashboard for this test."

After the plugin link is clicked, enter CloudTest credentials whenever required.

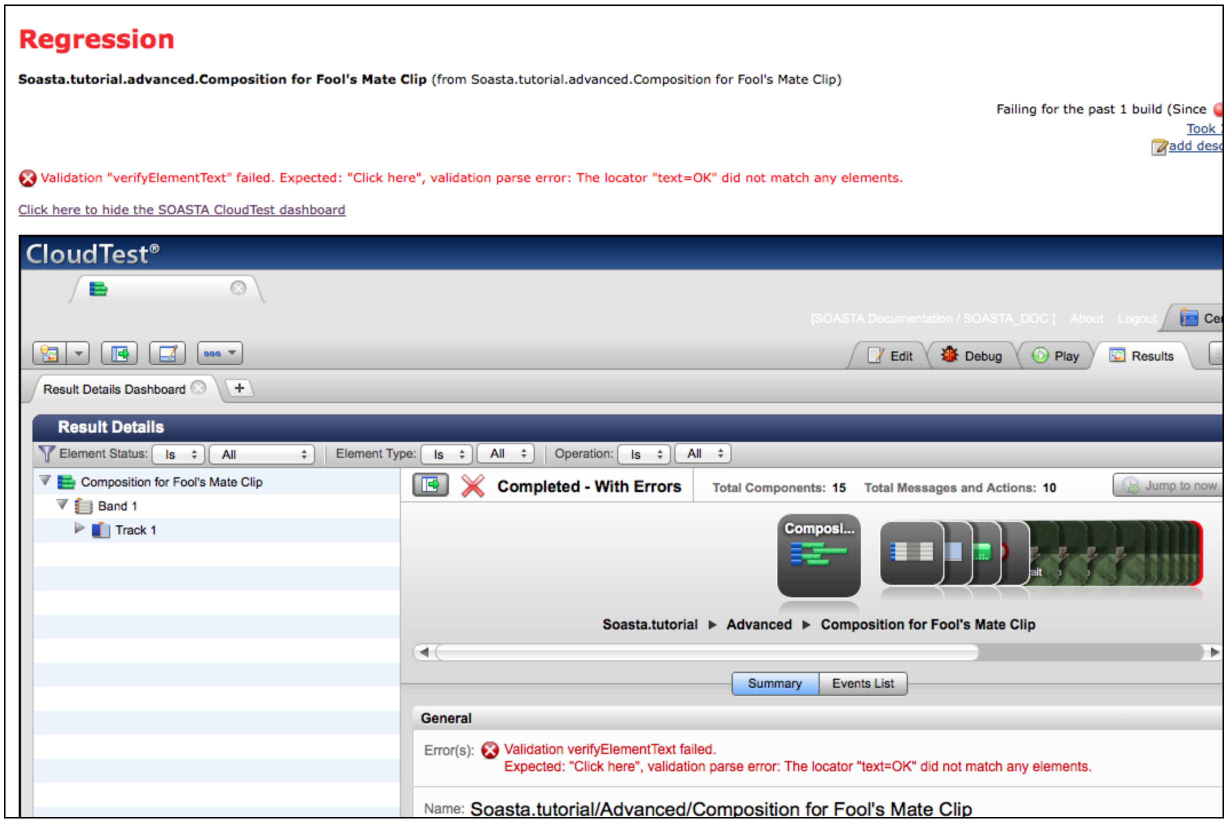

Failed

Soasta.tutorial.advanced.Composition for Fool's Mate Clip (from Soasta.tutorial.advanced.Composition for Fool's Mate Clip)

Validation "verifyElementText" failed. Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

Click here to hide the SOASTA CloudTest dashboard

- After credentials are entered, the dashboard tab opens, displays the test result, and jumps to the relevant error.


Regression

Soasta.tutorial.advanced.Composition for Fool's Mate Clip (from Soasta.tutorial.advanced.Composition for Fool's Mate Clip)

Failing for the past 1 build (Since

Took

add desc

Validation "verifyElementText" failed. Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

Click here to hide the SOASTA CloudTest dashboard

Element Status: Is All Element Type: Is All Operation: Is All

Composition for Fool's Mate Clip
Band 1
Track 1

Completed - With Errors Total Components: 15 Total Messages and Actions: 10

Jump to now
Composi...
Soasta.tutorial ▶ Advanced ▶ Composition for Fool's Mate Clip
Summary
Events List
General

Error(s): Validation verifyElementText failed.
Expected: "Click here", validation parse error: The locator "text=OK" did not match any elements.

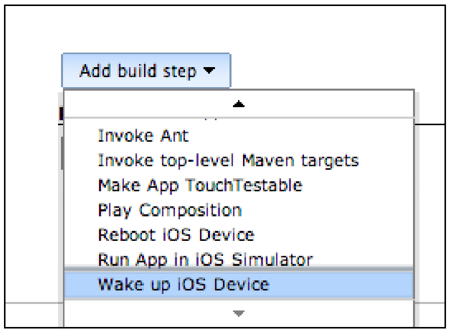
Name: Soasta.tutorial/Advanced/Composition for Fool's Mate Clip

From here, the Result Details dashboard can be navigated as within any CloudTest dashboard. Refer to Result Details Dashboard for a quick review of Result Details features.

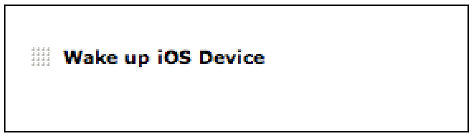
Appendix I: Waking Up an iOS Device

The SOASTA CloudTest Jenkins Plugin includes the ability to wake up a device. This technique is useful in mobile device test labs as an energy saving measure and can be used in coordination with the ability to reboot a device. Note that the TouchTest Agent should be the Safari page running on the sleeping device for test execution to succeed.

1. Add a build step using Wake Up iOS Device. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.



The Wake up iOS Device form appears. This command will wake up any device tethered on the Mac where the Jenkins job runs (for example on a slave node assigned to the job).



Note: Ensure that the device(s) do not have passcodes in effect.

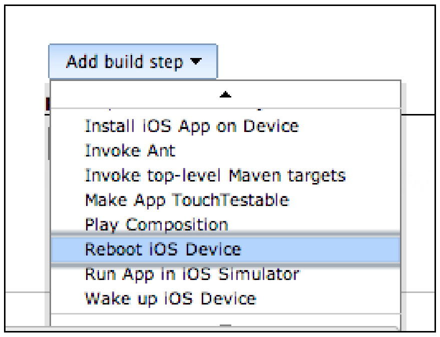
2. Drag the build step into the correct position. In general, at the beginning of a job and before Xcode build steps.

3. Click Apply on the Job page.

Appendix II: Rebooting an iOS Device

The SOASTA CloudTest Jenkins Plugin includes the ability to reboot a device as part of test lab setup. This technique is useful in mobile device test labs to refresh the iOS device state, which can be subject to degradation over time.

1. Add a build step using Reboot iOS Device. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.



The Reboot iOS Device form appears. This command will reboot any device tethered on the Mac where the Jenkins job runs (for example on a slave node assigned to the job).

Note: Ensure that the device(s) do not have passcodes in effect.

2. Drag the build step into the correct position. In general, at the beginning of a job and before Xcode build steps.

3. Click Apply on the Job page.

Appendix III: Importing Source Controlled SOASTA XML

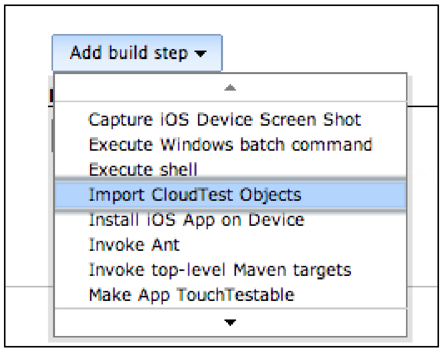
The SOASTA CloudTest Jenkins Plugin includes the ability to import SOASTA objects such as test clips from exported XML. This technique is useful in test environments where more than one tester is working as part of a team.

For example, a team might have access to the following:

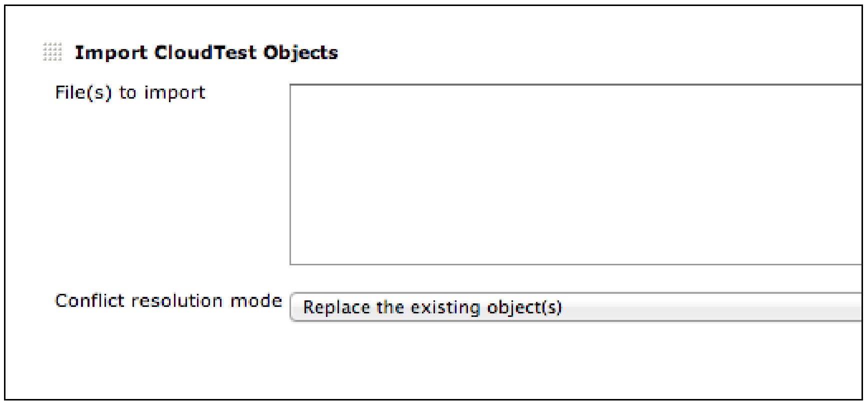
- 1 CloudTest Pro environment being used for CI, with Jenkins pointed at it.
- 1 version control system.
- 3 CloudTest Lite users, working on their own tests, do the following:
  - The CloudTest Lite users export their compositions and clips using the non-zipped XML, and check them into SVN, on a regular basis.
  - The Jenkins job checks them out from SVN, imports the XML into the CloudTest Pro environment, and then runs the compositions on that same environment. The exported objects can include dependent objects but the export should be in XML format and not zipped. Refer to Export and Import SOASTA Objects for steps.

Importing CloudTest Objects

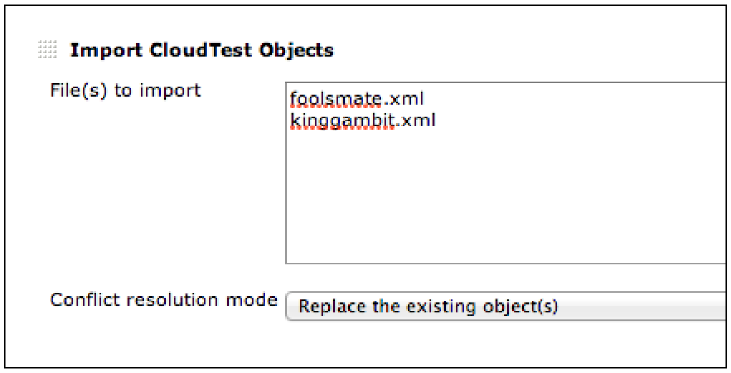
1. Export the object(s) to import and check them into the repository that also has the Xcode project.
2. Add a build step using Import CloudTest Objects. Note that after the step is created you can drag it to the correct place in the workflow of your Jenkins job.


Add build step
Capture iOS Device Screen Shot
Execute Windows batch command
Execute shell
Import CloudTest Objects
Install iOS App on Device
Invoke Ant
Invoke top-level Maven targets
Make App TouchTestable

The Import CloudTest Objects form appears. If you will be exporting more than one XML file click the down arrow to expand the form before proceeding.



4. Specify the file name of each XML file to import (one per line).



File(s) to import:

```
foolsmate.xml
kinggambit.xml
```

Conflict resolution mode: Replace the existing object(s)

TIP: You can also use wildcards here. For example, if the CloudTest objects were checked into a "testobjects" folder, then entering `testobjects/*.xml` here would import them all.

5. For this scenario, accept the default conflict resolution mode, which is "Replace the existing object(s)."

6. Drag the build step into the correct position. In general, before any Play Composition steps.

7. Click Apply on the Job page.

Appendix IV: Jenkins Plugin, Transaction Thresholds

The Transaction Threshold command—which was made available in the SOASTA CloudTest Plugin, version 2.17—provides users with the ability to automate the validation of Transactions in a composition using CloudTest metrics, such as Average Response Time or Errors per Transaction.

For more details, please visit Setting Transaction Thresholds in Jenkins.

SOASTA, Inc.
444 Castro St.
Mountain View, CA 94041
866.344.8766
http://www.soasta.com