Schedule Jobs Using the Apex Scheduler

**Learning Objectives**

After completing this unit, you’ll know:

* When to use scheduled Apex.
* How to monitor scheduled jobs.
* Scheduled Apex syntax.
* Scheduled method best practices.

**Follow Along with Trail Together**

Want to follow along with an instructor as you work through this step? Take a look at this video, part of the Trail Together series on Trailhead Live.

(This clip starts at the 1:09:53 minute mark, in case you want to rewind and watch the beginning of the step again.)

**Scheduled Apex**

The Apex Scheduler lets you delay execution so that you can run Apex classes at a specified time. This is ideal for daily or weekly maintenance tasks using Batch Apex. To take advantage of the scheduler, write an Apex class that implements the Schedulable interface, and then schedule it for execution on a specific schedule.

**Scheduled Apex Syntax**

To invoke Apex classes to run at specific times, first implement the Schedulable interface for the class. Then, schedule an instance of the class to run at a specific time using the System.schedule method.

public class SomeClass implements Schedulable {

public void **execute**(SchedulableContext ctx) {

// awesome code here

}

}

Copy

The class implements the Schedulable interface and must implement the only method that this interface contains, which is the execute method.

The parameter of this method is a SchedulableContext object. After a class has been scheduled, a CronTrigger object is created that represents the scheduled job. It provides a getTriggerId method that returns the ID of a CronTrigger API object.

**Sample Code**

This class queries for open opportunities that should have closed by the current date, and creates a task on each one to remind the owner to update the opportunity.

public class RemindOpptyOwners implements Schedulable {

public void **execute**(SchedulableContext ctx) {

List<Opportunity> opptys = [SELECT Id, Name, OwnerId, CloseDate

FROM Opportunity

WHERE IsClosed = False AND

CloseDate < TODAY];

// Create a task for each opportunity in the list

TaskUtils.**remindOwners**(opptys);

}

}

Copy

You can schedule your class to run either programmatically or from the Apex Scheduler UI.

**Using the System.Schedule Method**

After you implement a class with the Schedulable interface, use the System.Schedule method to execute it. The System.Schedule method uses the user's timezone for the basis of all schedules, but runs in system mode—all classes are executed, whether or not the user has permission to execute the class.

Note

Use extreme care if you’re planning to schedule a class from a trigger. You must be able to guarantee that the trigger won’t add more scheduled job classes than the limit. In particular, consider API bulk updates, import wizards, mass record changes through the user interface, and all cases where more than one record can be updated at a time.

The System.Schedule method takes three arguments: a name for the job, a CRON expression used to represent the time and date the job is scheduled to run, and an instance of a class that implements the Schedulable interface.

RemindOpptyOwners reminder = new RemindOpptyOwners();

// Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

String sch = '20 30 8 10 2 ?';

String jobID = System.**schedule**('Remind Opp Owners', sch, reminder);

Copy

For more information on the CRON expression used for scheduling, see the “Using the System.Schedule Method” section in [Apex Scheduler](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_scheduler.htm).

**Scheduling a Job from the UI**

You can also schedule a class using the user interface.

1. From Setup, enter Apex in the Quick Find box, then select **Apex Classes**.
2. Click **Schedule Apex**.
3. For the job name, enter something like Daily Oppty Reminder.
4. Click the lookup button next to Apex class and enter \* for the search term to get a list of all classes that can be scheduled. In the search results, click the name of your scheduled class.
5. Select Weekly or Monthly for the frequency and set the frequency desired.
6. Select the start and end dates, and a preferred start time.
7. Click **Save**.

**Testing Scheduled Apex**

Just like with the other async methods we’ve covered so far, with Scheduled Apex you must also ensure that the scheduled job is finished before testing against the results. To do this, use startTest and stopTest again around the System.schedule method, to ensure processing finishes before continuing your test.

@isTest

private class RemindOppyOwnersTest {

// Dummy CRON expression: midnight on March 15.

// Because this is a test, job executes

// immediately after Test.stopTest().

public static String CRON\_EXP = '0 0 0 15 3 ? 2022';

static testmethod void **testScheduledJob**() {

// Create some out of date Opportunity records

List<Opportunity> opptys = new List<Opportunity>();

Date closeDate = Date.**today**().**addDays**(-7);

for (Integer i=0; i<10; i++) {

Opportunity o = new Opportunity(

Name = 'Opportunity ' + i,

CloseDate = closeDate,

StageName = 'Prospecting'

);

opptys.**add**(o);

}

insert opptys;

// Get the IDs of the opportunities we just inserted

Map<Id, Opportunity> opptyMap = new Map<Id, Opportunity>(opptys);

List<Id> opptyIds = new List<Id>(opptyMap.**keySet**());

Test.**startTest**();

// Schedule the test job

String jobId = System.**schedule**('ScheduledApexTest',

CRON\_EXP,

new RemindOpptyOwners());

// Verify the scheduled job has not run yet.

List<Task> lt = [SELECT Id

FROM Task

WHERE WhatId IN :opptyIds];

System.**assertEquals**(0, lt.**size**(), 'Tasks exist before job has run');

// Stopping the test will run the job synchronously

Test.**stopTest**();

// Now that the scheduled job has executed,

// check that our tasks were created

lt = [SELECT Id

FROM Task

WHERE WhatId IN :opptyIds];

System.**assertEquals**(opptyIds.**size**(),

lt.**size**(),

'Tasks were not created');

}

}

Copy

**Things to Remember**

Scheduled Apex has a number of items you need to be aware of (see Apex Scheduler in the Resources section for a complete list when you have time), but in general:

* You can only have 100 scheduled Apex jobs at one time and there are maximum number of scheduled Apex executions per a 24-hour period. See Execution Governors and Limits in the Resources section for details.
* Use extreme care if you’re planning to schedule a class from a trigger. You must be able to guarantee that the trigger won’t add more scheduled jobs than the limit.
* Synchronous Web service callouts are not supported from scheduled Apex. To be able to make callouts, make an asynchronous callout by placing the callout in a method annotated with @future(callout=true) and call this method from scheduled Apex. However, if your scheduled Apex executes a batch job, callouts are supported from the batch class.

**Resources**

* [Apex Scheduler](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_scheduler.htm" \t "_blank)
* [Execution Governors and Limits](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_gov_limits.htm)

Top of Form

**Hands-on Challenge**

**+500 points**

**GET READY**

You’ll be completing this unit in your own hands-on org. Click **Launch** to get started, or click the name of your org to choose a different one.

If you use Trailhead in a language other than English, make sure that your hands-on org is set to the same language as the challenge instructions. Otherwise you may run into issues passing this unit. Want to find out more about using hands-on orgs on Trailhead? Check out [Trailhead Playground Management](https://trailhead.salesforce.com/en/content/learn/modules/trailhead_playground_management).

**YOUR CHALLENGE**

**Create an Apex class that uses Scheduled Apex to update Lead records.**

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

* Create an Apex class:
  + Name: DailyLeadProcessor
  + Interface: Schedulable
  + The execute method must find the first 200 Lead records with a blank LeadSource field and update them with the LeadSource value of Dreamforce
* Create an Apex test class:
  + Name: DailyLeadProcessorTest
  + In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly
  + The unit tests must cover all lines of code included in the **DailyLeadProcessor** class, resulting in 100% code coverage.
* Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

**swamy2**

Last used on 6/27/2022

Bottom of Form

<https://github.com/Mulodo-Salesforce-Training/trailhead-salesforce-challenge/blob/master/Scheduler.md>

global class DailyLeadProcessor implements Schedulable{

global void execute(SchedulableContext ctx){

List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

if(leads.size() > 0){

List<Lead> newLeads = new List<Lead>();

for(Lead lead : leads){

lead.LeadSource = 'DreamForce';

newLeads.add(lead);

}

update newLeads;

}

}

}

@isTest

private class DailyLeadProcessorTest{

//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year

public static String CRON\_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){

List<Lead> leads = new List<Lead>();

for(Integer i = 0; i < 200; i++){

Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');

leads.add(lead);

}

insert leads;

Test.startTest();

// Schedule the test job

String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());

// Stopping the test will run the job synchronously

Test.stopTest();

}

}

<https://developer.salesforce.com/forums/?id=906F0000000DAGEIA4>