#Get Started with Apex Triggers

Add the following trigger using the Developer Console (follow the steps of the HelloWorldTrigger example but use AddRelatedRecord for the trigger name).

```
trigger AddRelatedRecord on Account(after insert, after update) {
    List<Opportunity> oppList = new List<Opportunity>();

    // Get the related opportunities for the accounts in this trigger
    Map<Id,Account> acctsWithOpps = new Map<Id,Account>(
        [SELECT Id,(SELECT Id FROM Opportunities) FROM Account WHERE Id IN :Trigger.New]);

    // Add an opportunity for each account if it doesn't already have one.
    // Iterate through each account.
    for(Account a : Trigger.New) {
        System.debug('acctsWithOpps.get(a.Id).Opportunities.size()=' + acctsWithOpps.get(a.Id).Opportunities.size());
        // Check if the account already has a related opportunity.
        if (acctsWithOpps.get(a.Id).Opportunities.size() == 0) {
            // If it doesn't, add a default opportunity
            oppList.add(new Opportunity(Name=a.Name + ' Opportunity',
                                StageName='Prospecting',
                                CloseDate=System.today().addMonths(1),
                                AccountId=a.Id));
        }
    }
    if (oppList.size() > 0) {
        insert oppList;
    }
}
```

#2

Using the Developer Console, add the following trigger.

```
trigger AccountDeletion on Account (before delete) {

    // Prevent the deletion of accounts if they have related opportunities.
    for (Account a : [SELECT Id FROM Account
                      WHERE Id IN (SELECT AccountId FROM Opportunity) AND
                      Id IN :Trigger.old]) {
        Trigger.oldMap.get(a.Id).addError(
            'Cannot delete account with related opportunities.');
    }

}

public class CalloutClass {
    @future(callout=true)
    public static void makeCallout() {
        HttpRequest request = new HttpRequest();
        // Set the endpoint URL.
        String endpoint = 'http://yourHost/yourService';
        request.setEndPoint(endpoint);
        // Set the HTTP verb to GET.
        request.setMethod('GET');
        // Send the HTTP request and get the response.
        HttpResponse response = new HTTP().send(request);
```

```
    }
}'



#Bulk Apex Trigger

trigger MyTriggerNotBulk on Account(before insert) {
    Account a = Trigger.New[0];
    a.Description = 'New description';
}


trigger MyTriggerBulk on Account(before insert) {
    for(Account a : Trigger.New) {
        a.Description = 'New description';
    }
}


trigger SoqlTriggerNotBulk on Account(after update) {
    for(Account a : Trigger.New) {
        // Get child records for each account
        // Inefficient SOQL query as it runs once for each account!
        Opportunity[] opps = [SELECT Id,Name,CloseDate
                              FROM Opportunity WHERE AccountId=:a.Id];

        // Do some other processing
    }
}


trigger SoqlTriggerBulk on Account(after update) {
    // Perform SOQL query once.
    // Get the accounts and their related opportunities.
    List<Account> acctsWithOpps =
        [SELECT Id,(SELECT Id,Name,CloseDate FROM Opportunities)
         FROM Account WHERE Id IN :Trigger.New];

    // Iterate over the returned accounts
    for(Account a : acctsWithOpps) {
        Opportunity[] relatedOpps = a.Opportunities;
        // Do some other processing
    }
}

trigger SoqlTriggerBulk on Account(after update) {
    // Perform SOQL query once.
    // Get the related opportunities for the accounts in this trigger.
    List<Opportunity> relatedOpps = [SELECT Id,Name,CloseDate FROM Opportunity
        WHERE AccountId IN :Trigger.New];

    // Iterate over the related opportunities
    for(Opportunity opp : relatedOpps) {
        // Do some other processing
```

```
    }
}


trigger SoqlTriggerBulk on Account(after update) {
    // Perform SOQL query once.
    // Get the related opportunities for the accounts in this trigger,
    // and iterate over those records.
    for(Opportunity opp : [SELECT Id,Name,CloseDate FROM Opportunity
        WHERE AccountId IN :Trigger.New]) {

        // Do some other processing
    }
}


trigger DmlTriggerNotBulk on Account(after update) {
    // Get the related opportunities for the accounts in this trigger.
    List<Opportunity> relatedOpps = [SELECT Id,Name,Probability FROM Opportunity
        WHERE AccountId IN :Trigger.New];
    // Iterate over the related opportunities
    for(Opportunity opp : relatedOpps) {
        // Update the description when probability is greater
        // than 50% but less than 100%
        if ((opp.Probability >= 50) && (opp.Probability < 100)) {
            opp.Description = 'New description for opportunity.';
            // Update once for each opportunity -- not efficient!
            update opp;
        }
    }
}



trigger DmlTriggerBulk on Account(after update) {
    // Get the related opportunities for the accounts in this trigger.
    List<Opportunity> relatedOpps = [SELECT Id,Name,Probability FROM Opportunity
        WHERE AccountId IN :Trigger.New];

    List<Opportunity> oppsToUpdate = new List<Opportunity>();
    // Iterate over the related opportunities
    for(Opportunity opp : relatedOpps) {
        // Update the description when probability is greater
        // than 50% but less than 100%
        if ((opp.Probability >= 50) && (opp.Probability < 100)) {
            opp.Description = 'New description for opportunity.';
            oppsToUpdate.add(opp);
        }
    }

    // Perform DML on a collection
    update oppsToUpdate;
}
```

```
[SELECT Id,Name FROM Account WHERE Id IN :Trigger.New AND
                                Id NOT IN (SELECT AccountId FROM Opportunity)]


for(Account a : [SELECT Id,Name FROM Account WHERE Id IN :Trigger.New AND
                                Id NOT IN (SELECT AccountId FROM Opportunity)]){
}


trigger AddRelatedRecord on Account(after insert, after update) {
    List<Opportunity> oppList = new List<Opportunity>();

    // Add an opportunity for each account if it doesn't already have one.
    // Iterate over accounts that are in this trigger but that don't have opportunities.
    for (Account a : [SELECT Id,Name FROM Account
                      WHERE Id IN :Trigger.New AND
                      Id NOT IN (SELECT AccountId FROM Opportunity)]) {
        // Add a default opportunity for this account
        oppList.add(new Opportunity(Name=a.Name + ' Opportunity',
                                    StageName='Prospecting',
                                    CloseDate=System.today().addMonths(1),
                                    AccountId=a.Id));
    }

    if (oppList.size() > 0) {
        insert oppList;
    }
}


#
Get Started with Apex Unit Tests

@isTest static void testName() {
    // code_block
}

@isTest
private class MyTestClass {
    @isTest static void myTest() {
        // code_block
    }
}

public class TemperatureConverter {
    // Takes a Fahrenheit temperature and returns the Celsius equivalent.
    public static Decimal FahrenheitToCelsius(Decimal fh) {
        Decimal cs = (fh - 32) * 5/9;
        return cs.setScale(2);
    }
}


@isTest
private class TemperatureConverterTest {
```

```
    @isTest static void testWarmTemp() {
        Decimal celsius = TemperatureConverter.FahrenheitToCelsius(70);
        System.assertEquals(21.11,celsius);
    }
    @isTest static void testFreezingPoint() {
        Decimal celsius = TemperatureConverter.FahrenheitToCelsius(32);
        System.assertEquals(0,celsius);
    }
    @isTest static void testBoilingPoint() {
        Decimal celsius = TemperatureConverter.FahrenheitToCelsius(212);
        System.assertEquals(100,celsius,'Boiling point temperature is not expected.');
    }
    @isTest static void testNegativeTemp() {
        Decimal celsius = TemperatureConverter.FahrenheitToCelsius(-10);
        System.assertEquals(-23.33,celsius);
    }
}

public class TaskUtil {
    public static String getTaskPriority(String leadState) {
        // Validate input
        if (String.isBlank(leadState) || leadState.length() > 2) {
            return null;
        }
        String taskPriority;
        if (leadState == 'CA') {
            taskPriority = 'High';
        } else {
            taskPriority = 'Normal';
        }
        return taskPriority;
    }
}

@isTest
private class TaskUtilTest {
    @isTest static void testTaskPriority() {
        String pri = TaskUtil.getTaskPriority('NY');
        System.assertEquals('Normal', pri);
    }
}

@isTest
private class TaskUtilTest {
    @isTest static void testTaskPriority() {
        String pri = TaskUtil.getTaskPriority('NY');
        System.assertEquals('Normal', pri);
    }
    @isTest static void testTaskHighPriority() {
        String pri = TaskUtil.getTaskPriority('CA');
        System.assertEquals('High', pri);
    }
    @isTest static void testTaskPriorityInvalid() {
        String pri = TaskUtil.getTaskPriority('Montana');
        System.assertEquals(null, pri);
```

```
    }
}


#Test Apex Triggers
trigger AccountDeletion on Account (before delete) {
    // Prevent the deletion of accounts if they have related opportunities.
    for (Account a : [SELECT Id FROM Account
                      WHERE Id IN (SELECT AccountId FROM Opportunity) AND
                      Id IN :Trigger.old]) {
        Trigger.oldMap.get(a.Id).addError(
            'Cannot delete account with related opportunities.');
    }
}

@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create an account with an opportunity, and then try to delete it
        Account acct = new Account(Name='Test Account');
        insert acct;
        Opportunity opp = new Opportunity(Name=acct.Name + ' Opportunity',
                                          StageName='Prospecting',
                                          CloseDate=System.today().addMonths(1),
                                          AccountId=acct.Id);
        insert opp;
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(acct, false);
        Test.stopTest();
        // Verify
        // In this case the deletion should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related opportunities.',
                             result.getErrors()[0].getMessage());
    }
}



#Create Test Data for Apex Tests
@isTest
public class TestDataFactory {
    public static List<Account> createAccountsWithOpps(Integer numAccts, Integer numOppsPerAcct) {
        List<Account> accts = new List<Account>();
        for(Integer i=0;i<numAccts;i++) {
            Account a = new Account(Name='TestAccount' + i);
            accts.add(a);
        }
        insert accts;
        List<Opportunity> opps = new List<Opportunity>();
        for (Integer j=0;j<numAccts;j++) {
```

```
            Account acct = accts[j];
            // For each account just inserted, add opportunities
            for (Integer k=0;k<numOppsPerAcct;k++) {
                opps.add(new Opportunity(Name=acct.Name + ' Opportunity ' + k,
                                         StageName='Prospecting',
                                         CloseDate=System.today().addMonths(1),
                                         AccountId=acct.Id));
            }
        }
        // Insert all opportunities for all accounts.
        insert opps;
        return accts;
    }
}


        // Test data setup
        // Create one account with one opportunity by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,1);



@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create one account with one opportunity by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,1);
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(accts[0], false);
        Test.stopTest();
        // Verify that the deletion should have been stopped by the trigger,
        // so check that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related opportunities.',
                             result.getErrors()[0].getMessage());
    }
}


@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create one account with one opportunity by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,1);
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(accts[0], false);
        Test.stopTest();
        // Verify that the deletion should have been stopped by the trigger,
        // so check that we got back an error.
        System.assert(!result.isSuccess());
```

```
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related opportunities.',
                             result.getErrors()[0].getMessage());
    }
    @isTest static void TestDeleteAccountWithNoOpportunities() {
        // Test data setup
        // Create one account with no opportunities by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(1,0);
        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(accts[0], false);
        Test.stopTest();
        // Verify that the deletion was successful
        System.assert(result.isSuccess());
    }
    @isTest static void TestDeleteBulkAccountsWithOneOpportunity() {
        // Test data setup
        // Create accounts with one opportunity each by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(200,1);
        // Perform test
        Test.startTest();
        Database.DeleteResult[] results = Database.delete(accts, false);
        Test.stopTest();
        // Verify for each record.
        // In this case the deletion should have been stopped by the trigger,
        // so check that we got back an error.
        for(Database.DeleteResult dr : results) {
            System.assert(!dr.isSuccess());
            System.assert(dr.getErrors().size() > 0);
            System.assertEquals('Cannot delete account with related opportunities.',
                                 dr.getErrors()[0].getMessage());
        }
    }
    @isTest static void TestDeleteBulkAccountsWithNoOpportunities() {
        // Test data setup
        // Create accounts with no opportunities by calling a utility method
        Account[] accts = TestDataFactory.createAccountsWithOpps(200,0);
        // Perform test
        Test.startTest();
        Database.DeleteResult[] results = Database.delete(accts, false);
        Test.stopTest();
        // For each record, verify that the deletion was successful
        for(Database.DeleteResult dr : results) {
            System.assert(dr.isSuccess());
        }
    }
}


#apex rest call out
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('GET');
HttpResponse response = http.send(request);
```

```
// If the request is successful, parse the JSON response.
if(response.getStatusCode() == 200) {
    // Deserialize the JSON string into collections of primitive data types.
    Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
    // Cast the values in the 'animals' key as a list
    List<Object> animals = (List<Object>) results.get('animals');
    System.debug('Received the following animals:');
    for(Object animal: animals) {
        System.debug(animal);
    }
}
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
request.setMethod('POST');
request.setHeader('Content-Type', 'application/json;charset=UTF-8');
// Set the body as a JSON object
request.setBody('{"name":"mighty moose"}');
HttpResponse response = http.send(request);
// Parse the JSON response
if(response.getStatusCode() != 201) {
    System.debug('The status code returned was not expected: ' + response.getStatusCode() + ' ' + response.getStatus()
);
} else {
    System.debug(response.getBody());
}
public class AnimalsCallouts {
    public static HttpResponse makeGetCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
            // Deserializes the JSON string into collections of primitive data types.
            Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
            List<Object> animals = (List<Object>) results.get('animals');
            System.debug('Received the following animals:');
            for(Object animal: animals) {
                System.debug(animal);
            }
        }
        return response;
    }
    public static HttpResponse makePostCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals');
        request.setMethod('POST');
        request.setHeader('Content-Type', 'application/json;charset=UTF-8');
        request.setBody('{"name":"mighty moose"}');
        HttpResponse response = http.send(request);
        // Parse the JSON response
```

```
        if(response.getStatusCode() != 201) {
            System.debug('The status code returned was not expected: ' +
                response.getStatusCode() + ' ' + response.getStatus());
        } else {
            System.debug(response.getBody());
        }
        return response;
    }
}
```

```
{"animals": ["pesky porcupine", "hungry hippo", "squeaky squirrel"]}
```

```
@isTest
private class AnimalsCalloutsTest {
    @isTest static  void testGetCallout() {
        // Create the mock response based on a static resource
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('GetAnimalResource');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json;charset=UTF-8');
        // Associate the callout with a mock response
        Test.setMock(HttpCalloutMock.class, mock);
        // Call method to test
        HttpResponse result = AnimalsCallouts.makeGetCallout();
        // Verify mock response is not null
        System.assertNotEquals(null,result, 'The callout returned a null response.');
        // Verify status code
        System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
        // Verify content type
        System.assertEquals('application/json;charset=UTF-8',
          result.getHeader('Content-Type'),
          'The content type value is not expected.');
        // Verify the array contains 3 items
        Map<String, Object> results = (Map<String, Object>)
            JSON.deserializeUntyped(result.getBody());
        List<Object> animals = (List<Object>) results.get('animals');
        System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
    }
}
```

```
Test.setMock(HttpCalloutMock.class, new AnimalsHttpCalloutMock());
```

```
@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');
```

```
        response.setStatusCode(200);
        return response;
    }
}


@isTest
static void testPostCallout() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalsHttpCalloutMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    HttpResponse response = AnimalsCallouts.makePostCallout();
    // Verify that the response received contains fake values
    String contentType = response.getHeader('Content-Type');
    System.assert(contentType == 'application/json');
    String actualValue = response.getBody();
    System.debug(response.getBody());
    String expectedValue = '{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]
}';
    System.assertEquals(expectedValue, actualValue);
    System.assertEquals(200, response.getStatusCode());
}

#Apex SOAP Callouts
calculatorServices.CalculatorImplPort calculator = new  calculatorServices.CalculatorImplPort();
Double x = 1.0;
Double y = 2.0;
Double result = calculator.doAdd(x,y);
System.debug(result);


Test.setMock(WebServiceMock.class, new MyWebServiceMockImpl());

public class AwesomeCalculator {
    public static Double add(Double x, Double y) {
        calculatorServices.CalculatorImplPort calculator =
            new calculatorServices.CalculatorImplPort();
        return calculator.doAdd(x,y);
    }
}


@isTest
global class CalculatorCalloutMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
```

```
        // start - specify the response you want to send
        calculatorServices.doAddResponse response_x =
            new calculatorServices.doAddResponse();
        response_x.return_x = 3.0;
        // end
        response.put('response_x', response_x);
    }
}


@isTest
private class AwesomeCalculatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new CalculatorCalloutMock());
        // Call the method that invokes a callout
        Double x = 1.0;
        Double y = 2.0;
        Double result = AwesomeCalculator.add(x, y);
        // Verify that a fake result is returned
        System.assertEquals(3.0, result);
    }
}



#


@RestResource(urlMapping='/Account/*')
global with sharing class MyRestResource {
    @HttpGet
    global static Account getRecord() {
        // Add your code
    }
}

global with sharing class MySOAPWebService {
    webservice static Account getRecord(String id) {
        // Add your code
    }
}

@RestResource(urlMapping='/Cases/*')
global with sharing class CaseManager {
    @HttpGet
    global static Case getCaseById() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
        String caseId = request.requestURI.substring(
          request.requestURI.lastIndexOf('/')+1);
        Case result =  [SELECT CaseNumber,Subject,Status,Origin,Priority
                        FROM Case
                        WHERE Id = :caseId];
        return result;
```

```
    }
    @HttpPost
    global static ID createCase(String subject, String status,
        String origin, String priority) {
        Case thisCase = new Case(
            Subject=subject,
            Status=status,
            Origin=origin,
            Priority=priority);
        insert thisCase;
        return thisCase.Id;
    }
    @HttpDelete
    global static void deleteCase() {
        RestRequest request = RestContext.request;
        String caseId = request.requestURI.substring(
            request.requestURI.lastIndexOf('/')+1);
        Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
        delete thisCase;
    }
    @HttpPut
    global static ID upsertCase(String subject, String status,
        String origin, String priority, String id) {
        Case thisCase = new Case(
                Id=id,
                Subject=subject,
                Status=status,
                Origin=origin,
                Priority=priority);
        // Match case by Id, if present.
        // Otherwise, create new case.
        upsert thisCase;
        // Return the case ID.
        return thisCase.Id;
    }
    @HttpPatch
    global static ID updateCaseFields() {
        RestRequest request = RestContext.request;
        String caseId = request.requestURI.substring(
            request.requestURI.lastIndexOf('/')+1);
        Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
        // Deserialize the JSON string into name-value pairs
        Map<String, Object> params = (Map<String, Object>)JSON.deserializeUntyped(request.requestbody.tostring()
);
        // Iterate through each parameter field and value
        for(String fieldName : params.keySet()) {
            // Set the field and value on the Case sObject
            thisCase.put(fieldName, params.get(fieldName));
        }
        update thisCase;
        return thisCase.Id;
    }
}

{
```

```
  "subject" : "Bigfoot Sighting!",
  "status" : "New",
  "origin" : "Phone",
  "priority" : "Low"
}

HTTP/1.1 200 OK
Date: Wed, 07 Oct 2015 14:18:20 GMT
Set-Cookie: BrowserId=F1wxIhHPQHCXp6wrvqToXA;Path=/;Domain=.salesforce.com;Expires=Sun, 06-Dec-201
5 14:18:20 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json;charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked
"50061000000t7kYAAQ"

HTTP/1.1 200 OK
Date: Wed, 07 Oct 2015 14:28:20 GMT
Set-Cookie: BrowserId=j5qAnPDdRxSu8eHGqaRVLQ;Path=/;Domain=.salesforce.com;Expires=Sun, 06-Dec-2015
 14:28:20 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json;charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked
{
  "attributes" : {
    "type" : "Case",
    "url" : "/services/data/v34.0/sobjects/Case/50061000000t7kYAAQ"
  },
  "CaseNumber" : "00001026",
  "Subject" : "Bigfoot Sighting!",
  "Status" : "New",
  "Origin" : "Phone",
  "Priority" : "Low",
  "Id" : "50061000000t7kYAAQ"
}


curl -v https://login.salesforce.com/services/oauth2/token -d "grant_type=password" -d "client_id=<your_consumer_
key>" -d "client_secret=<your_consumer_secret>" -d "username=<your_username>" -d "password=<your_passwor
d_and_security_token>" -H "X-PrettyPrint:1"


curl https://yourInstance.my.salesforce.com/services/apexrest/Cases/<Record_ID> -H "Authorization: Bearer <your
_session_id>" -H "X-PrettyPrint:1"


@HttpPut
global static ID upsertCase(String subject, String status,
    String origin, String priority, String id) {
    Case thisCase = new Case(
        Id=id,
        Subject=subject,
        Status=status,
        Origin=origin,
```

```
        Priority=priority);
    // Match case by Id, if present.
    // Otherwise, create new case.
    upsert thisCase;
    // Return the case ID.
    return thisCase.Id;
}


@HttpPatch
global static ID updateCaseFields() {
    RestRequest request = RestContext.request;
    String caseId = request.requestURI.substring(
        request.requestURI.lastIndexOf('/')+1);
    Case thisCase = [SELECT Id FROM Case WHERE Id = :caseId];
    // Deserialize the JSON string into name-value pairs
    Map<String, Object> params = (Map<String, Object>)JSON.deserializeUntyped(request.requestbody.tostring());
    // Iterate through each parameter field and value
    for(String fieldName : params.keySet()) {
        // Set the field and value on the Case sObject
        thisCase.put(fieldName, params.get(fieldName));
    }
    update thisCase;
    return thisCase.Id;
}

// Set up a test request
RestRequest request = new RestRequest();
// Set request properties
request.requestUri =
    'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
    + recordId;
request.httpMethod = 'GET';
// Set other properties, such as parameters
request.params.put('status', 'Working');
// more awesome code here....
// Finally, assign the request to RestContext if used
RestContext.request = request;


@IsTest
private class CaseManagerTest {
    @isTest static void testGetCaseById() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
            + recordId;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Case thisCase = CaseManager.getCaseById();
        // Verify results
        System.assert(thisCase != null);
```

```
    System.assertEquals('Test record', thisCase.Subject);
}
@isTest static void testCreateCase() {
    // Call the method to test
    ID thisCaseId = CaseManager.createCase(
        'Ferocious chipmunk', 'New', 'Phone', 'Low');
    // Verify results
    System.assert(thisCaseId != null);
    Case thisCase = [SELECT Id,Subject FROM Case WHERE Id=:thisCaseId];
    System.assert(thisCase != null);
    System.assertEquals(thisCase.Subject, 'Ferocious chipmunk');
}
@isTest static void testDeleteCase() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
        + recordId;
    request.httpMethod = 'DELETE';
    RestContext.request = request;
    // Call the method to test
    CaseManager.deleteCase();
    // Verify record is deleted
    List<Case> cases = [SELECT Id FROM Case WHERE Id=:recordId];
    System.assert(cases.size() == 0);
}
@isTest static void testUpsertCase() {
    // 1. Insert new record
    ID case1Id = CaseManager.upsertCase(
            'Ferocious chipmunk', 'New', 'Phone', 'Low', null);
    // Verify new record was created
    System.assert(Case1Id != null);
    Case case1 = [SELECT Id,Subject FROM Case WHERE Id=:case1Id];
    System.assert(case1 != null);
    System.assertEquals(case1.Subject, 'Ferocious chipmunk');
    // 2. Update status of existing record to Working
    ID case2Id = CaseManager.upsertCase(
            'Ferocious chipmunk', 'Working', 'Phone', 'Low', case1Id);
    // Verify record was updated
    System.assertEquals(case1Id, case2Id);
    Case case2 = [SELECT Id,Status FROM Case WHERE Id=:case2Id];
    System.assert(case2 != null);
    System.assertEquals(case2.Status, 'Working');
}
@isTest static void testUpdateCaseFields() {
    Id recordId = createTestRecord();
    RestRequest request = new RestRequest();
    request.requestUri =
        'https://yourInstance.my.salesforce.com/services/apexrest/Cases/'
        + recordId;
    request.httpMethod = 'PATCH';
    request.addHeader('Content-Type', 'application/json');
    request.requestBody = Blob.valueOf('{"status": "Working"}');
    RestContext.request = request;
```

```
        // Update status of existing record to Working
        ID thisCaseId = CaseManager.updateCaseFields();
        // Verify record was updated
        System.assert(thisCaseId != null);
        Case thisCase = [SELECT Id,Status FROM Case WHERE Id=:thisCaseId];
        System.assert(thisCase != null);
        System.assertEquals(thisCase.Status, 'Working');
    }
    // Helper method
    static Id createTestRecord() {
        // Create test record
        Case caseTest = new Case(
            Subject='Test record',
            Status='New',
            Origin='Phone',
            Priority='Medium');
        insert caseTest;
        return caseTest.Id;
    }
}


#Use Future Methods
public class SomeClass {
  @future
  public static void someFutureMethod(List<Id> recordIds) {
    List<Account> accounts = [Select Id, Name from Account Where Id IN :recordIds];
    // process account records to do awesome stuff
  }
}


public class SMSUtils {
    // Call async from triggers, etc, where callouts are not permitted.
    @future(callout=true)
    public static void sendSMSAsync(String fromNbr, String toNbr, String m) {
        String results = sendSMS(fromNbr, toNbr, m);
        System.debug(results);
    }
    // Call from controllers, etc, for immediate processing
    public static String sendSMS(String fromNbr, String toNbr, String m) {
        // Calling 'send' will result in a callout
        String results = SmsMessage.send(fromNbr, toNbr, m);
        insert new SMS_Log__c(to__c=toNbr, from__c=fromNbr, msg__c=results);
        return results;
    }
}


@isTest
public class SMSCalloutMock implements HttpCalloutMock {
    public HttpResponse respond(HttpRequest req) {
        // Create a fake response
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
```

```
        res.setBody('{"status":"success"}');
        res.setStatusCode(200);
        return res;
    }
}

@IsTest
private class Test_SMSUtils {
  @IsTest
  private static void testSendSms() {
    Test.setMock(HttpCalloutMock.class, new SMSCalloutMock());
    Test.startTest();
      SMSUtils.sendSMSAsync('111', '222', 'Greetings!');
    Test.stopTest();
    // runs callout and check results
    List<SMS_Log__c> logs = [select msg__c from SMS_Log__c];
    System.assertEquals(1, logs.size());
    System.assertEquals('success', logs[0].msg__c);
  }
}

#Use Batch Apex

public class MyBatchClass implements Database.Batchable<sObject> {
    public (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
    }
    public void execute(Database.BatchableContext bc, List<P> records){
        // process each batch of records
    }
    public void finish(Database.BatchableContext bc){
        // execute any post-processing operations
    }
}

MyBatchClass myBatchObject = new MyBatchClass();
Id batchId = Database.executeBatch(myBatchObject);

Id batchId = Database.executeBatch(myBatchObject, 100);


AsyncApexJob job = [SELECT Id, Status, JobItemsProcessed, TotalJobItems, NumberOfErrors FROM AsyncApexJ
ob WHERE ID = :batchId ];

public class UpdateContactAddresses implements
    Database.Batchable<sObject>, Database.Stateful {
    // instance member to retain state across transactions
    public Integer recordsProcessed = 0;
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID, BillingStreet, BillingCity, BillingState, ' +
            'BillingPostalCode, (SELECT ID, MailingStreet, MailingCity, ' +
            'MailingState, MailingPostalCode FROM Contacts) FROM Account ' +
            'Where BillingCountry = \'USA\''
        );
```

```
    }
    public void execute(Database.BatchableContext bc, List<Account> scope){
        // process each batch of records
        List<Contact> contacts = new List<Contact>();
        for (Account account : scope) {
            for (Contact contact : account.contacts) {
                contact.MailingStreet = account.BillingStreet;
                contact.MailingCity = account.BillingCity;
                contact.MailingState = account.BillingState;
                contact.MailingPostalCode = account.BillingPostalCode;
                // add contact to list to be updated
                contacts.add(contact);
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;
            }
        }
        update contacts;
    }
    public void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');
        AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
            JobItemsProcessed,
            TotalJobItems, CreatedBy.Email
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()];
        // call some utility to send email
        EmailUtils.sendMessage(job, recordsProcessed);
    }
}


@isTest
private class UpdateContactAddressesTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        List<Contact> contacts = new List<Contact>();
        // insert 10 accounts
        for (Integer i=0;i<10;i++) {
            accounts.add(new Account(name='Account '+i,
                billingcity='New York', billingcountry='USA'));
        }
        insert accounts;
        // find the account just inserted. add contact for each
        for (Account account : [select id from account]) {
            contacts.add(new Contact(firstname='first',
                lastname='last', accountId=account.id));
        }
        insert contacts;
    }
    @isTest static void test() {
        Test.startTest();
        UpdateContactAddresses uca = new UpdateContactAddresses();
        Id batchId = Database.executeBatch(uca);
        Test.stopTest();
```

```
        // after the testing stops, assert records were updated properly
        System.assertEquals(10, [select count() from contact where MailingCity = 'New York']);
    }
}
```

#Control Processes with Queueable Apex

```
@future
static void myFutureMethod(List<String> params) {
    // call synchronous method
    mySyncMethod(params);
}

public class SomeClass implements Queueable {
    public void execute(QueueableContext context) {
        // awesome code here
    }
}


public class UpdateParentAccount implements Queueable {
    private List<Account> accounts;
    private ID parent;
    public UpdateParentAccount(List<Account> records, ID id) {
        this.accounts = records;
        this.parent = id;
    }
    public void execute(QueueableContext context) {
        for (Account account : accounts) {
          account.parentId = parent;
          // perform other processing or callout
        }
        update accounts;
    }
}


// find all accounts in ‘NY’
List<Account> accounts = [select id from account where billingstate = ‘NY’];
// find a specific parent account for all records
Id parentId = [select id from account where name = 'ACME Corp'][0].Id;
// instantiate a new instance of the Queueable class
UpdateParentAccount updateJob = new UpdateParentAccount(accounts, parentId);
// enqueue the job for processing
ID jobID = System.enqueueJob(updateJob);



SELECT Id, Status, NumberOfErrors FROM AsyncApexJob WHERE Id = :jobID
```

```
@isTest
public class UpdateParentAccountTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        // add a parent account
        accounts.add(new Account(name='Parent'));
        // add 100 child accounts
        for (Integer i = 0; i < 100; i++) {
            accounts.add(new Account(
                name='Test Account'+i
            ));
        }
        insert accounts;
    }
    static testmethod void testQueueable() {
        // query for test data to pass to queueable class
        Id parentId = [select id from account where name = 'Parent'][0].Id;
        List<Account> accounts = [select id, name from account where name like 'Test Account%'];
        // Create our Queueable instance
        UpdateParentAccount updater = new UpdateParentAccount(accounts, parentId);
        // startTest/stopTest block to force async processes to run
        Test.startTest();
        System.enqueueJob(updater);
        Test.stopTest();
        // Validate the job ran. Check if record have correct parentId now
        System.assertEquals(100, [select count() from account where parentId = :parentId]);
    }
}


public class FirstJob implements Queueable {
    public void execute(QueueableContext context) {
        // Awesome processing logic here
        // Chain this job to next job by submitting the next job
        System.enqueueJob(new SecondJob());
    }
}


#Schedule Jobs Using the Apex Scheduler

public class SomeClass implements Schedulable {
    public void execute(SchedulableContext ctx) {
        // awesome code here
    }
}

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);
    }
}


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);


@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');
    }
}
```

#

```
AsyncApexJob jobInfo = [SELECT Status, NumberOfErrors
   FROM AsyncApexJob WHERE Id = :jobID];

CronTrigger ct = [SELECT TimesTriggered, NextFireTime FROM CronTrigger WHERE Id = :jobID];


public class DoAwesomeStuff implements Schedulable {
   public void execute(SchedulableContext sc) {
      // some awesome code
      CronTrigger ct = [SELECT TimesTriggered, NextFireTime FROM CronTrigger WHERE Id = :sc.getTriggerId
()];
   }
}
CronTrigger job = [SELECT Id, CronJobDetail.Id, CronJobDetail.Name, CronJobDetail.JobType FROM CronTrigg
er ORDER BY CreatedDate DESC LIMIT 1];


SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.JobType = '7’
```