

APEX TRIGGERS

Get Started With Apex

AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert,before update) {

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode =account.BillingPostalCode;
        }
    }
}
```

Bulk Apex Triggers

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
    List<Task> tasklist = new List<Task>();

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

APEX TESTING

Get Started With Apex Unittest

VerifyDate.apxc

```
public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2. Otherwise use the end
of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }
}
```

```

    }
}

//method to check if date2 is within the next 30 days of date1
@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
    //check for date2 being in the past
    if( date2 < date1) { return false; }

    //check that date2 is within (>=) 30 days of date1
    Date date30Days = date1.addDays(30); //create a date 30 days away from date1
    if( date2 >= date30Days ) { return false; }
    else { return true; }
}

//method to return the end of the month of a given date
@TestVisible private static Date SetEndOfMonthDate(Date date1) {
    Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
    Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
    return lastDay;
}
}

TestVerifyDate.apxc
@Test
public class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'),D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'),D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }
}

```

```

    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }

}

```

Test Apex Triggers

RestrictContactByName.apxt

```

trigger RestrictContactByName on Contact (before insert, before update) {

    //check contacts prior to insert or update for invalid data
    For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
            c.AddError('The Last Name "' + c.LastName + '" is not allowed for DML');
        }
    }

}

```

```

}

```

TestRestrictContactByName.apxc

```

@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){

```

```

Contact cnt = new Contact();
cnt.LastName = 'INVALIDNAME';

Test.startTest();
Database.SaveResult result = Database.insert(cnt,false);
Test.stopTest();

System.assert(!result.isSuccess());
System.assert(result.getErrors().size() > 0);
System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
}
}

```

Create Test Data for Apex Tests

RandomContactFactory.apxc

```

public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt,string lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test'+i,LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}

```

ASYNCHRONOUS APEX

Use Future Methods

AccountProcessor.apxc

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new List<Account>();
        List<Account> accounts = [Select Id, Name,(Select Id from Contacts) from Account Where
        Id in :accountIds];
    }
}

```

```

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}

AccountProcessorTest.apxc
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();

    }
}

```

Use Batch Apex

LeadProcessor.apxc

```

global class LeadProcessor implements Database.Batchable<sObject>{
    global Integer count = 0;

    global Database.QueryLocator start(Database.BatchableContext bc){

```

```

        return Database.getQueryLocator('SELECT ID,LeadSource FROM Lead');
    }

    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count = '+ count);
    }
}

```

LeadProcessorTest.apxc

```

@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0; i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}

```

```
}
```

Control Processes with Queueable Apex

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }

    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName,LastName, Id from contacts)
                                from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }

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

    }

}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++){
```

```

        testAccounts.add(new Account(Name='Account'+i,BillingState='CA'));
    }
    for(Integer j=0;j<50;j++){
        testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));
    }
    insert testAccounts;

    Contact testContact = new Contact(FirstName = 'John',LastName = 'Doe');
    insert testContact;

    AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

    Test.startTest();
    system.enqueueJob(addit);
    Test.stopTest();

    system.assertEquals(50,[Select count() from Contact where accountId in (Select Id from
Account where BillingState='CA')]);
    }
}

```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc

```

global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
            }
            update lList;
        }
    }
}

```

DailyLeadProcessorTest.apxc

```

@isTest
public class DailyLeadProcessorTest {
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 4 ? 2023';
}

```



```

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();
}
}

```

APEX INTEGRATION SERVICES

Apex REST Callouts

AnimalLocator.apxc

```

public class AnimalLocator {
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, object> animal= new Map<String, Object>();
        HttpResponse res = http.send(req);
        if (res.getStatusCode() == 200){
            Map<String, Object> results = (Map<String,
            Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) results.get('animal');

```

```

    }
    return (String)animal.get('name');
}
}

```

AnimalLocatorTest.apxc

```

@Test
private class AnimalLocatorTest {
    @Test static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult);
    }
}

```

AnimalLocatorMock.apxc

```

@Test
global class AnimalLocatorMock 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;
    }
}

```

Apex SOAP Callouts

ParkService.apxc

//Generated by wsdl2apex

```

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-
1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
    }
}

```

```

        private String[] field_order_type_info = new String[]{return_x};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{"arg0","http://parks.services/",null,"0","1","false"};
        private String[] apex_schema_type_info = new String[]{"http://parks.services/","false","false"};
        private String[] field_order_type_info = new String[]{"arg0"};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{"http://parks.services/", 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
                this,
                request_x,
                response_map_x,
                new String[]{endpoint_x,
                    ",
                    'http://parks.services/',
                    'byCountry',
                    'http://parks.services/',
                    'byCountryResponse',
                    'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}

```

ParkLocator.apxc

```
public class ParkLocator {  
    Public static string[] country(String country){  
        parkService.parksImplPort park = new parkService.parksImplPort();  
        return park.byCountry(country);  
    }  
}
```

ParkLocatorTest.apxc

```
@isTest  
private class ParkLocatorTest {  
    @isTest  
    static void testCallout(){  
        Test.setMock(WebServiceMock.class, new ParkServiceMock());  
        String country = 'USA';  
        System.assertEquals(new List<String>{'Me', 'You', 'Her'}, ParkLocator.country(country));  
    }  
}
```

ParkServiceMock.apxc

```
@isTest  
global class ParkServiceMock 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){  
        parkService.byCountryResponse response_x = new parkService.byCountryResponse();  
        response_x.return_x = new List<String>{'Me', 'You', 'Her'};  
        response.put('response_x', response_x);  
    }  
}
```

AsyncParksServices.apxc

//Generated by wsdl2apex

```
public class AsyncParkService {  
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
```

```

        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }

    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
                this,
                request_x,
                AsyncParkService.byCountryResponseFuture.class,
                continuation,
                new String[]{endpoint_x,
                    "",
                    'http://parks.services/',
                    'byCountry',
                    'http://parks.services/',
                    'byCountryResponse',
                    'ParkService.byCountryResponse'}
                );
        }
    }
}

```

Apex Web Services

AccountManager.apxc

```

@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {

```

```

@HttpGet
global static Account getAccount(){
    RestRequest request = RestContext.request;
    string accountId = request.requestURI.substringBetween('Accounts/', '/contacts');
    Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account where
Id=:accountId Limit 1];
    return result;
}
}

```

AccountManagerTest.apxc

```

@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
        + recordId + '/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }

    static Id createTestRecord(){
        Account accountTest = new Account(
            Name = 'Test record');
        insert accountTest;
        Contact contactTest = new Contact(
            FirstName='John',
            LastName = 'Doe',
            AccountId = accountTest.Id
        );
        insert contactTest;

        return accountTest.Id;
    }
}

```