ParkService.apxc

```
1  //Generated by wsdl2apex
2
3  public class ParkService {
4      public class byCountryResponse {
5          public String[] return_x;
6          private String[] return_x_type_info = new
   String[]{'return','http://parks.services/',null,'0','-

7          private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
8          private String[] field_order_type_info = new
   String[]{'return_x'};
9      }
10     public class byCountry {
11         public String arg0;
12         private String[] arg0_type_info = new
   String[]{'arg0','http://parks.services/',null,'0','1','fals

13         private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
14         private String[] field_order_type_info = new
   String[]{'arg0'};
15     }
16     public class ParksImplPort {
17         public String endpoint_x = 'https://th-apex-soap-

18         public Map<String,String> inputHttpHeaders_x;
19         public Map<String,String> outputHttpHeaders_x;
20         public String clientCertName_x;
21         public String clientCert_x;
22         public String clientCertPasswd_x;
23         public Integer timeout_x;
24         private String[] ns_map_type_info = new
   String[]{'http://parks.services/', 'ParkService'};
```

```
25          public String[] byCountry(String arg0) {
26              ParkService.byCountry request_x = new
    ParkService.byCountry();
27              request_x.arg0 = arg0;
28              ParkService.byCountryResponse response_x;
29              Map<String, ParkService.byCountryResponse>
    response_map_x = new Map<String,
    ParkService.byCountryResponse>();
30              response_map_x.put('response_x', response_x);
31              WebServiceCallout.invoke(
32                this,
33                request_x,
34                response_map_x,
35                new String[]{endpoint_x,
36                '',
37                'http://parks.services/',
38                'byCountry',
39                'http://parks.services/',
40                'byCountryResponse',
41                'ParkService.byCountryResponse'}
42              );
43              response_x = response_map_x.get('response_x');
44              return response_x.return_x;
45          }
46      }
47  }
```

AsyncParkService.apxc

```
1  //Generated by wsdl2apex
2
3  public class AsyncParkService {
4      public class byCountryResponseFuture extends
    System.WebServiceCalloutFuture {
```

```
5           public String[] getValue() {
6               ParkService.byCountryResponse response =
  (ParkService.byCountryResponse)System.WebServiceCallout.end

7               return response.return_x;
8           }
9       }
10      public class AsyncParksImplPort {
11          public String endpoint_x = 'https://th-apex-soap-

12          public Map<String,String> inputHttpHeaders_x;
13          public String clientCertName_x;
14          public Integer timeout_x;
15          private String[] ns_map_type_info = new
  String[]{'http://parks.services/', 'ParkService'};
16          public AsyncParkService.byCountryResponseFuture
  beginByCountry(System.Continuation continuation,String
  arg0) {
17              ParkService.byCountry request_x = new
  ParkService.byCountry();
18              request_x.arg0 = arg0;
19              return
  (AsyncParkService.byCountryResponseFuture)
  System.WebServiceCallout.beginInvoke(
20                this,
21                request_x,
22
  AsyncParkService.byCountryResponseFuture.class,
23                continuation,
24                new String[]{endpoint_x,
25                '',
26                'http://parks.services/',
27                'byCountry',
28                'http://parks.services/',
29                'byCountryResponse',
30                'ParkService.byCountryResponse'}
```

```
            );
        }
    }
}
```

AnimalLocatorMock.apxc

```
@isTest
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",


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

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new
```

```
   AnimalLocatorMock());
5          string result = AnimalLocator.getAnimalNameById(3);
6          String expectedResult = 'chicken';
7          System.assertEquals(result,expectedResult );
8      }
9  }
```

AnimalLocator.apxc

```
1  public class AnimalLocator{
2      public static String getAnimalNameById(Integer x){
3          Http http = new Http();
4          HttpRequest req = new HttpRequest();
5          req.setEndpoint('https://th-apex-http-

6          req.setMethod('GET');
7          Map<String, Object> animal= new Map<String,
   Object>();
8          HttpResponse res = http.send(req);
9              if (res.getStatusCode() == 200) {
10         Map<String, Object> results = (Map<String,
   Object>)JSON.deserializeUntyped(res.getBody());
11       animal = (Map<String, Object>) results.get('animal');
12         }
13 return (String)animal.get('name');
14     }
15 }
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId =
req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name
FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
'https://na1.salesforce.com/services/apexrest/Accounts/'+
recordId +'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record',
thisAccount.Name);
```

```
16
17      }
18
19      // Helper method
20          static Id createTestRecord() {
21          // Create test record
22          Account TestAcc = new Account(
23            Name='Test record');
24          insert TestAcc;
25          Contact TestCon= new Contact(
26          LastName='Test',
27          AccountId = TestAcc.id);
28          return TestAcc.Id;
29      }
30  }
```

ParkLocator.apxc

```
1  public class ParkLocator {
2      public static string[] country(string theCountry) {
3          ParkService.ParksImplPort  parkSvc = new
   ParkService.ParksImplPort(); // remove space
4          return parkSvc.byCountry(theCountry);
5      }
6  }
```

ParkServiceMock.apxc

```
1  @isTest
2  global class ParkServiceMock implements WebServiceMock {
3     global void doInvoke(
4              Object stub,
5              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
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
        response_x.return_x = new
List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
        // end
        response.put('response_x', response_x);
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new
ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new
List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

CreateDefaultData.apxc

```
1  public with sharing class CreateDefaultData{
2      Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine

3      //gets value from custom metadata
   How_We_Roll_Settings__mdt to know if Default data was
   created
4      @AuraEnabled
5      public static Boolean isDataCreated() {
6          How_We_Roll_Settings__c customSetting =
   How_We_Roll_Settings__c.getOrgDefaults();
7          return customSetting.Is_Data_Created__c;
8      }
9
10     //creates Default Data for How We Roll application
11     @AuraEnabled
12     public static void createDefaultData(){
13         List<Vehicle__c> vehicles = createVehicles();
14         List<Product2> equipment = createEquipment();
15         List<Case> maintenanceRequest =
   createMaintenanceRequest(vehicles);
16         List<Equipment_Maintenance_Item__c> joinRecords =
   createJoinRecords(equipment, maintenanceRequest);
17
18         updateCustomSetting(true);
19     }
20
21
22     public static void updateCustomSetting(Boolean
   isDataCreated){
23         How_We_Roll_Settings__c customSetting =
   How_We_Roll_Settings__c.getOrgDefaults();
24         customSetting.Is_Data_Created__c = isDataCreated;
25         upsert customSetting;
26     }
```

```
27
28      public static List<Vehicle__c> createVehicles(){
29          List<Vehicle__c> vehicles = new List<Vehicle__c>();
30          vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV',
    Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
    1, Model__c = 'Toy Hauler RV'));
31          vehicles.add(new Vehicle__c(Name = 'Travel Trailer

    Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));
32          vehicles.add(new Vehicle__c(Name = 'Teardrop

    Bedrooms__c = 1, Model__c = 'Teardrop Camper'));
33          vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper',
    Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c =
    1, Model__c = 'Pop-Up Camper'));
34          insert vehicles;
35          return vehicles;
36      }
37
38      public static List<Product2> createEquipment(){
39          List<Product2> equipments = new List<Product2>();
40          equipments.add(new Product2(Warehouse_SKU__c =
    '55d66226726b611100aaf741',name = 'Generator 1000 kW',
    Replacement_Part__c = true,Cost__c = 100
    ,Maintenance_Cycle__c = 100));
41          equipments.add(new Product2(name = 'Fuse

    Maintenance_Cycle__c = 30  ));
42          equipments.add(new Product2(name = 'Breaker

    Maintenance_Cycle__c = 15));
43          equipments.add(new Product2(name = 'UPS 20

    Maintenance_Cycle__c = 60));
44          insert equipments;
45          return equipments;
```

```
46
47      }
48
49      public static List<Case>
    createMaintenanceRequest(List<Vehicle__c> vehicles){
50          List<Case> maintenanceRequests = new List<Case>();
51          maintenanceRequests.add(new Case(Vehicle__c =
    vehicles.get(1).Id, Type = TYPE_ROUTINE_MAINTENANCE,
    Date_Reported__c = Date.today()));
52          maintenanceRequests.add(new Case(Vehicle__c =
    vehicles.get(2).Id, Type = TYPE_ROUTINE_MAINTENANCE,
    Date_Reported__c = Date.today()));
53          insert maintenanceRequests;
54          return maintenanceRequests;
55      }
56
57      public static List<Equipment_Maintenance_Item__c>
    createJoinRecords(List<Product2> equipment, List<Case>
    maintenanceRequest){
58          List<Equipment_Maintenance_Item__c> joinRecords =
    new List<Equipment_Maintenance_Item__c>();
59          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(0).Id, Maintenance_Request__c =
    maintenanceRequest.get(0).Id));
60          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(1).Id, Maintenance_Request__c =
    maintenanceRequest.get(0).Id));
61          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(2).Id, Maintenance_Request__c =
    maintenanceRequest.get(0).Id));
62          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(0).Id, Maintenance_Request__c =
    maintenanceRequest.get(1).Id));
```

```
        joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(1).Id, Maintenance_Request__c =
    maintenanceRequest.get(1).Id));
        joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c =
    equipment.get(2).Id, Maintenance_Request__c =
    maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest.apxc

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM
    Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM
    Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM
    Case];
        List<Equipment_Maintenance_Item__c> joinRecords =
    [SELECT Id FROM Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There

        System.assertEquals(4, equipment.size(), 'There

        System.assertEquals(2, maintenanceRequest.size(),
    'There should have been 2 maintenance request created');
```

```
15          System.assertEquals(6, joinRecords.size(), 'There

16
17      }
18
19      @isTest
20      static void updateCustomSetting_test(){
21          How_We_Roll_Settings__c customSetting =
    How_We_Roll_Settings__c.getOrgDefaults();
22          customSetting.Is_Data_Created__c = false;
23          upsert customSetting;
24
25          System.assertEquals(false,
    CreateDefaultData.isDataCreated(), 'The custom setting


26
27          customSetting.Is_Data_Created__c = true;
28          upsert customSetting;
29
30          System.assertEquals(true,
    CreateDefaultData.isDataCreated(), 'The custom setting


31
32      }
33  }
```

MaintenanceRequestHelper.apxc

```
1  public with sharing class MaintenanceRequestHelper {
2      public static void updateworkOrders(List<Case>
    updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
3          Set<Id> validIds = new Set<Id>();
```

```
4           For (Case c : updWorkOrders){
5               if (nonUpdCaseMap.get(c.Id).Status != 'Closed'
  && c.Status == 'Closed'){
6                   if (c.Type == 'Repair' || c.Type ==
  'Routine Maintenance'){
7                       validIds.add(c.Id);
8                   }
9               }
10          }
11
12          //When an existing maintenance request of type
  Repair or Routine Maintenance is closed,
13          //create a new maintenance request for a future
  routine checkup.
14          if (!validIds.isEmpty()){
15              Map<Id,Case> closedCases = new
  Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
  Equipment__r.Maintenance_Cycle__c,
16
  (SELECT Id,Equipment__c,Quantity__c FROM
  Equipment_Maintenance_Items__r)
17
  FROM Case WHERE Id IN :validIds]);
18              Map<Id,Decimal> maintenanceCycles = new
  Map<ID,Decimal>();
19
20              //calculate the maintenance request due dates
  by using the maintenance cycle defined on the related
  equipment records.
21              AggregateResult[] results = [SELECT
  Maintenance_Request__c,
22
  MIN(Equipment__r.Maintenance_Cycle__c)cycle
23                                          FROM
  Equipment_Maintenance_Item__c
24                                          WHERE
  Maintenance_Request__c IN :ValidIds GROUP BY
```

```
Maintenance_Request__c];

            for (AggregateResult ar : results){
                maintenanceCycles.put((Id)
ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
            }

            List<Case> newCases = new List<Case>();
            for(Case cc : closedCases.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                    Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()
                );

                //If multiple pieces of equipment are used
in the maintenance request,
                //define the due date by applying the
shortest maintenance cycle to today's date.
                //If
(maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c =
Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
                //} else {
                //    nc.Date_Due__c =
Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                //}

                newCases.add(nc);
```

```
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> clonedList
  = new List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c
  clonedListItem :
  closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r
  ){
                    Equipment_Maintenance_Item__c item =
  clonedListItem.clone();
                    item.Maintenance_Request__c = nc.Id;
                    clonedList.add(item);
                }
            }
            insert clonedList;
        }
    }
}
```

MaintenanceRequestHelperTest.apxc

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing

        return vehicle;
    }

```

```
    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing

lifespan_months__c = 10,
maintenance_cycle__c = 10,
replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id
vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }

    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c
equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }
```

```
37
38      @isTest
39      private static void testPositive(){
40          Vehicle__c vehicle = createVehicle();
41          insert vehicle;
42          id vehicleId = vehicle.Id;
43
44          Product2 equipment = createEquipment();
45          insert equipment;
46          id equipmentId = equipment.Id;
47
48          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
49          insert createdCase;
50
51          Equipment_Maintenance_Item__c
    equipmentMaintenanceItem =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
52          insert equipmentMaintenanceItem;
53
54          test.startTest();
55          createdCase.status = 'Closed';
56          update createdCase;
57          test.stopTest();
58
59          Case newCase = [Select id,
60                          subject,
61                          type,
62                          Equipment__c,
63                          Date_Reported__c,
64                          Vehicle__c,
65                          Date_Due__c
66                         from case
67                         where status ='New'];
68
69          Equipment_Maintenance_Item__c workPart = [select id
70                                                    from
```

```
   Equipment_Maintenance_Item__c
71                                                  where
   Maintenance_Request__c =:newCase.Id];
72          list<case> allCase = [select id from case];
73          system.assert(allCase.size() == 2);
74
75          system.assert(newCase != null);
76          system.assert(newCase.Subject != null);
77          system.assertEquals(newCase.Type, 'Routine

78          SYSTEM.assertEquals(newCase.Equipment__c,
   equipmentId);
79          SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
80          SYSTEM.assertEquals(newCase.Date_Reported__c,
   system.today());
81      }
82
83      @isTest
84      private static void testNegative(){
85          Vehicle__C vehicle = createVehicle();
86          insert vehicle;
87          id vehicleId = vehicle.Id;
88
89          product2 equipment = createEquipment();
90          insert equipment;
91          id equipmentId = equipment.Id;
92
93          case createdCase =
   createMaintenanceRequest(vehicleId,equipmentId);
94          insert createdCase;
95
96          Equipment_Maintenance_Item__c workP =
   createEquipmentMaintenanceItem(equipmentId,
   createdCase.Id);
97          insert workP;
98
99          test.startTest();
```

```
100         createdCase.Status = 'Working';
101         update createdCase;
102         test.stopTest();
103 
104         list<case> allCase = [select id from case];
105 
106         Equipment_Maintenance_Item__c
  equipmentMaintenanceItem = [select id
107                                                     from
  Equipment_Maintenance_Item__c
108                                                     where
  Maintenance_Request__c = :createdCase.Id];
109 
110         system.assert(equipmentMaintenanceItem != null);
111         system.assert(allCase.size() == 1);
112     }
113 
114     @isTest
115     private static void testBulk(){
116         list<Vehicle__C> vehicleList = new
  list<Vehicle__C>();
117         list<Product2> equipmentList = new
  list<Product2>();
118         list<Equipment_Maintenance_Item__c>
  equipmentMaintenanceItemList = new
  list<Equipment_Maintenance_Item__c>();
119         list<case> caseList = new list<case>();
120         list<id> oldCaseIds = new list<id>();
121 
122         for(integer i = 0; i < 300; i++){
123             vehicleList.add(createVehicle());
124             equipmentList.add(createEquipment());
125         }
126         insert vehicleList;
127         insert equipmentList;
128 
```

```
129         for(integer i = 0; i < 300; i++){
130 
  caseList.add(createMaintenanceRequest(vehicleList.get(i).i

131         }
132         insert caseList;
133 
134         for(integer i = 0; i < 300; i++){
135 
  equipmentMaintenanceItemList.add(createEquipmentMaintenance

136         }
137         insert equipmentMaintenanceItemList;
138 
139         test.startTest();
140         for(case cs : caseList){
141             cs.Status = 'Closed';
142             oldCaseIds.add(cs.Id);
143         }
144         update caseList;
145         test.stopTest();
146 
147         list<case> newCase = [select id
148                                 from case
149                                 where status ='New'];
150 
151 
152 
153         list<Equipment_Maintenance_Item__c> workParts =
  [select id
154 
  from Equipment_Maintenance_Item__c
155 
  where Maintenance_Request__c in: oldCaseIds];
156 
157         system.assert(newCase.size() == 300);
```

```
158
159         list<case> allCase = [select id from case];
160         system.assert(allCase.size() == 600);
161     }
162 }
```

WarehouseCalloutService.apxc

```
1  public with sharing class WarehouseCalloutService
   implements Queueable {
2      private static final String WAREHOUSE_URL =
   'https://th-superbadge-apex.herokuapp.com/equipment';
3
4      //Write a class that makes a REST callout to an
   external warehouse system to get a list of equipment that
   needs to be updated.
5      //The callout's JSON response returns the equipment
   records that you upsert in Salesforce.
6
7      @future(callout=true)
8      public static void runWarehouseEquipmentSync(){
9          System.debug('go into runWarehouseEquipmentSync');
10         Http http = new Http();
11         HttpRequest request = new HttpRequest();
12
13         request.setEndpoint(WAREHOUSE_URL);
14         request.setMethod('GET');
15         HttpResponse response = http.send(request);
16
17         List<Product2> product2List = new List<Product2>();
18         System.debug(response.getStatusCode());
19         if (response.getStatusCode() == 200){
20             List<Object> jsonResponse =
   (List<Object>)JSON.deserializeUntyped(response.getBody());
```

```
21              System.debug(response.getBody());
22
23              //class maps the following fields:
24              //warehouse SKU will be external ID for
    identifying which equipment records to update within
    Salesforce
25              for (Object jR : jsonResponse){
26                  Map<String,Object> mapJson =
    (Map<String,Object>)jR;
27                  Product2 product2 = new Product2();
28                  //replacement part (always true),
29                  product2.Replacement_Part__c = (Boolean)
    mapJson.get('replacement');
30                  //cost
31                  product2.Cost__c = (Integer)
    mapJson.get('cost');
32                  //current inventory
33                  product2.Current_Inventory__c = (Double)
    mapJson.get('quantity');
34                  //lifespan
35                  product2.Lifespan_Months__c = (Integer)
    mapJson.get('lifespan');
36                  //maintenance cycle
37                  product2.Maintenance_Cycle__c = (Integer)
    mapJson.get('maintenanceperiod');
38                  //warehouse SKU
39                  product2.Warehouse_SKU__c = (String)
    mapJson.get('sku');
40
41                  product2.Name = (String)
    mapJson.get('name');
42                  product2.ProductCode = (String)
    mapJson.get('_id');
43                  product2List.add(product2);
44              }
45
```

```
            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced

            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request)
{

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","repla
```

```
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceTest.apxc

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
 @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}
```

WarehouseSyncScheduleTest.apxc

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to

WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger
WHERE Id =: jobId];
        System.assertEquals('WAITING',
String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```

WarehouseSyncSchedule.apxc

```
global with sharing class WarehouseSyncSchedule implements
Schedulable {
    // implement scheduled code here
    global void execute (SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

MaintenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
```

```
15          {
16                  Contact cont =
  c.clone(false,false,false,false);
17                  cont.AccountId =  acc.id;
18                  lstContact.add( cont );
19          }
20
21          if(lstContact.size() >0 )
22          {
23              insert lstContact;
24          }
25
26      }
27
28  }
```

AccountProcessor.apxc

```
1  public class AccountProcessor {
2      @future
3      public static void countContacts(List<Id> accountIds){
4          List<Account> accounts = [Select Id, Name from
  Account Where Id IN : accountIds];
5          List<Account> updatedAccounts = new
  List<Account>();
6          for(Account account : accounts){
7             account.Number_of_Contacts__c = [Select count()
  from Contact Where AccountId =: account.Id];
8              System.debug('No Of Contacts = ' +
  account.Number_of_Contacts__c);
9              updatedAccounts.add(account);
10         }
```

```
        update updatedAccounts;
    }

}
```

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
  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; }
  }
```

```
23
24   //method to return the end of the month of a given date
25   private static Date SetEndOfMonthDate(Date date1) {
26     Integer totalDays = Date.daysInMonth(date1.year(),
   date1.month());
27     Date lastDay = Date.newInstance(date1.year(),
   date1.month(), totalDays);
28     return lastDay;
29   }
30
31 }
```

RandomContactFactory.apxc

```
1  //@isTest
2  public class RandomContactFactory {
3      public static List<Contact>
   generateRandomContacts(Integer numContactsToGenerate,
   String FName) {
4          List<Contact> contactList = new List<Contact>();
5
6          for(Integer i=0;i<numContactsToGenerate;i++) {
7              Contact c = new Contact(FirstName=FName + ' ' +
   i, LastName = 'Contact '+i);
8              contactList.add(c);
9              System.debug(c);
10         }
11         //insert contactList;
12         System.debug(contactList.size());
13         return contactList;
14     }
15
16 }
```

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable  {
    Public void execute(SchedulableContext SC){
        List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
         for(Lead l:LeadObj){
             l.LeadSource='Dreamforce';
             update l;
         }
    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
  static testMethod void testDailyLeadProcessor() {
       String CRON_EXP = '0 0 1 * * ?';
       List<Lead> lList = new List<Lead>();
      for (Integer i = 0; i < 200; i++) {
            lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open - Not Contacted'));
       }
       insert lList;

       Test.startTest();
       String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new DailyLeadProcessor());
  }
}
```

LeadProcessor.apxc

```
public class LeadProcessor implements Database.Batchable<sObject> {

     public Database.QueryLocator start(Database.BatchableContext bc) {
         // collect the batches of records or objects to be passed to execute
          return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
          // process each batch of records
             for (Lead Lead : leads) {
                 lead.LeadSource = 'Dreamforce';
             }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
      }

}
```

TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate
{
    static testMethod void testMethod1()
    {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
        Date d1 =
```

```
    VerifyDate.CheckDates(System.today(),System.today()+60);
    }
}
```

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
    @isTest
    public static void testNoOfContacts(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }

}
```

TestRestrictContactByName.apxc

```
@isTest
private class TestRestrictContactByName {

    static testMethod void  metodoTest()
    {

        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='Francesco',
LastName='Riggio' , email='Test@test.com');
        Contact c2 = new Contact(FirstName='Francesco1',
LastName = 'INVALIDNAME',email='Test@test.com');
        listContact.add(c1);
        listContact.add(c2);

        Test.startTest();
            try
            {
                insert listContact;
            }
            catch(Exception ee)
            {
            }

        Test.stopTest();

    }

}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
     @isTest static void TestList()
     {
         List<Account> Teste = new List <Account>();
         for(Integer i=0;i<50;i++)
         {
             Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
         }
         for(Integer j=0;j<50;j++)
         {
             Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
         }
         insert Teste;

         Contact co = new Contact();
         co.FirstName='demo';
         co.LastName ='demo';
         insert co;
         String state = 'CA';

          AddPrimaryContact apc = new AddPrimaryContact(co, state);
          Test.startTest();
            System.enqueueJob(apc);
          Test.stopTest();
      }
```

```
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

        @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company ='demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }

    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }

}
```

RestrictContactByName.apxt

```
1  trigger RestrictContactByName on Contact (before insert,
   before update) {
2
3    //check contacts prior to insert or update for invalid
   data
4    For (Contact c : Trigger.New) {
5      if(c.LastName == 'INVALIDNAME') {  //invalidname is
   invalid
6        c.AddError('The Last Name "'+c.LastName+'" is not

7      }
8
9    }
10
11
12
13 }
```

ClosedOpportunityTrigger.apxt

```
1  trigger ClosedOpportunityTrigger on Opportunity (after
   insert, after update) {
2
3      List<Task> taskList = new List<Task>();
4
5      for(Opportunity opp : Trigger.new) {
6
7      //Only create Follow Up Task only once when Opp
   StageName is to 'Closed Won' on Create
8      if(Trigger.isInsert) {
9        if(Opp.StageName == 'Closed Won') {
10         taskList.add(new Task(Subject = 'Follow Up Test
```

```
      }
    }

    //Only create Follow Up Task only once when Opp StageName changed to 'Closed Won' on Update
    if(Trigger.isUpdate) {
      if(Opp.StageName == 'Closed Won'
      && Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {
        taskList.add(new Task(Subject = 'Follow Up Test

      }
    }
    }

    if(taskList.size()>0) {
        insert taskList;
    }
}
```

AccountAddressTrigger.apxt

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


List<Account> acclst=new List<Account>();
  for(account a:trigger.new){
    if(a.Match_Billing_Address__c==true && a.BillingPostalCode!=null){
    a.ShippingPostalCode=a.BillingPostalCode;

    }

```

```
11 }
12 }
```