## ParkService.apxc

1. //Generated by wsdl2apex
2. public class ParkService {
3. public class byCountryResponse {
4. public String[] return\_x;
5. private String[] return\_x\_type\_info = new String[]{'return','http://parks.services/',null,'0','-1','false'};
6. private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};
7. private String[] field\_order\_type\_info = new String[]{'return\_x'};
8. }
9. public class byCountry {
10. public String arg0;
11. private String[] arg0\_type\_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
12. private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};
13. private String[] field\_order\_type\_info = new String[]{'arg0'};
14. }
15. public class ParksImplPort {
16. public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
17. public Map<String,String> inputHttpHeaders\_x;
18. public Map<String,String> outputHttpHeaders\_x;
19. public String clientCertName\_x;
20. public String clientCert\_x;
21. public String clientCertPasswd\_x;
22. public Integer timeout\_x;
23. private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};
24. public String[] byCountry(String arg0) {
25. ParkService.byCountry request\_x = new ParkService.byCountry();
26. request\_x.arg0 = arg0;
27. ParkService.byCountryResponse response\_x;
28. Map<String, ParkService.byCountryResponse> response\_map\_x = new Map<String, ParkService.byCountryResponse>();
29. response\_map\_x.put('response\_x', response\_x);
30. WebServiceCallout.invoke(
31. this,
32. request\_x,
33. response\_map\_x,
34. new String[]{endpoint\_x,
35. '',
36. 'http://parks.services/',
37. 'byCountry',
38. 'http://parks.services/',
39. 'byCountryResponse',
40. 'ParkService.byCountryResponse'}
41. );
42. response\_x = response\_map\_x.get('response\_x');
43. return response\_x.return\_x;
44. }
45. }
46. }

## AsyncParkService.apxc

1. //Generated by wsdl2apex
2. public class AsyncParkService {
3. public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
4. public String[] getValue() {
5. ParkService.byCountryResponse response = (ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
6. return response.return\_x;
7. }
8. }
9. public class AsyncParksImplPort {
10. public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
11. public Map<String,String> inputHttpHeaders\_x;
12. public String clientCertName\_x;
13. public Integer timeout\_x;
14. private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};
15. public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation continuation,String arg0) {
16. ParkService.byCountry request\_x = new ParkService.byCountry();
17. request\_x.arg0 = arg0;
18. return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(
19. this,
20. request\_x,
21. AsyncParkService.byCountryResponseFuture.class,
22. continuation,
23. new String[]{endpoint\_x,
24. '',
25. 'http://parks.services/',
26. 'byCountry',
27. 'http://parks.services/',
28. 'byCountryResponse',
29. 'ParkService.byCountryResponse'}
30. );
31. }
32. }
33. }

## AnimalLocatorMock.apxc

1. @isTest
2. global class AnimalLocatorMock implements HttpCalloutMock {
3. // Implement this interface method
4. global HTTPResponse respond(HTTPRequest request) {
5. // Create a fake response
6. HttpResponse response = new HttpResponse();
7. response.setHeader('Content-Type', 'application/json');
8. response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');
9. response.setStatusCode(200);
10. return response;
11. }
12. }

## AnimalLocatorTest.apxc

1. @isTest
2. private class AnimalLocatorTest{
3. @isTest static void AnimalLocatorMock1() {
4. 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-callout.herokuapp.com/animals/' + x);
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

1. @RestResource(urlMapping='/Accounts/\*/contacts')
2. global class AccountManager {
3. @HttpGet
4. global static Account getAccount() {
5. RestRequest req = RestContext.request;
6. String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
7. Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
8. FROM Account WHERE Id = :accId];
9. return acc;
10. }
11. }

## AccountManagerTest.apxc

1. @isTest
2. private class AccountManagerTest {
3. private static testMethod void getAccountTest1() {
4. Id recordId = createTestRecord();
5. // Set up a test request
6. RestRequest request = new RestRequest();
7. request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts' ;
8. request.httpMethod = 'GET';
9. RestContext.request = request;
10. // Call the method to test
11. Account thisAccount = AccountManager.getAccount();
12. // Verify results
13. System.assert(thisAccount != null);
14. System.assertEquals('Test record', thisAccount.Name);
15. }
16. // Helper method
17. static Id createTestRecord() {
18. // Create test record
19. Account TestAcc = new Account(
20. Name='Test record');
21. insert TestAcc;
22. Contact TestCon= new Contact(
23. LastName='Test',
24. AccountId = TestAcc.id);
25. return TestAcc.Id;
26. }
27. }

## 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,
6. Map<String, Object> response,
7. String endpoint,
8. String soapAction,
9. String requestName,
10. String responseNS,
11. String responseName,
12. String responseType) {
13. // start - specify the response you want to send
14. ParkService.byCountryResponse response\_x = new ParkService.byCountryResponse();
15. response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
16. // end
17. response.put('response\_x', response\_x);
18. }
19. }

## ParkLocatorTest.apxc

1. @isTest
2. private class ParkLocatorTest {
3. @isTest static void testCallout() {
4. Test.setMock(WebServiceMock.class, new ParkServiceMock ());
5. String country = 'United States';
6. List<String> result = ParkLocator.country(country);
7. List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
8. System.assertEquals(parks, result);
9. }
10. }

## CreateDefaultData.apxc

1. public with sharing class CreateDefaultData{
2. Static Final String TYPE\_ROUTINE\_MAINTENANCE = 'Routine Maintenance';
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. //creates Default Data for How We Roll application
10. @AuraEnabled
11. public static void createDefaultData(){
12. List<Vehicle\_\_c> vehicles = createVehicles();
13. List<Product2> equipment = createEquipment();
14. List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
15. List<Equipment\_Maintenance\_Item\_\_c> joinRecords = createJoinRecords(equipment, maintenanceRequest);
16. updateCustomSetting(true);
17. }
18. public static void updateCustomSetting(Boolean isDataCreated){
19. How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();
20. customSetting.Is\_Data\_Created\_\_c = isDataCreated;
21. upsert customSetting;
22. }
23. public static List<Vehicle\_\_c> createVehicles(){
24. List<Vehicle\_\_c> vehicles = new List<Vehicle\_\_c>();
25. vehicles.add(new Vehicle\_\_c(Name = 'Toy Hauler RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Toy Hauler RV'));
26. vehicles.add(new Vehicle\_\_c(Name = 'Travel Trailer RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 2, Bedrooms\_\_c = 2, Model\_\_c = 'Travel Trailer RV'));
27. vehicles.add(new Vehicle\_\_c(Name = 'Teardrop Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Teardrop Camper'));
28. vehicles.add(new Vehicle\_\_c(Name = 'Pop-Up Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Pop-Up Camper'));
29. insert vehicles;
30. return vehicles;
31. }
32. public static List<Product2> createEquipment(){
33. List<Product2> equipments = new List<Product2>();
34. equipments.add(new Product2(Warehouse\_SKU\_\_c = '55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement\_Part\_\_c = true,Cost\_\_c = 100 ,Maintenance\_Cycle\_\_c = 100));
35. equipments.add(new Product2(name = 'Fuse 20B',Replacement\_Part\_\_c = true,Cost\_\_c = 1000, Maintenance\_Cycle\_\_c = 30 ));
36. equipments.add(new Product2(name = 'Breaker 13C',Replacement\_Part\_\_c = true,Cost\_\_c = 100 , Maintenance\_Cycle\_\_c = 15));
37. equipments.add(new Product2(name = 'UPS 20 VA',Replacement\_Part\_\_c = true,Cost\_\_c = 200 , Maintenance\_Cycle\_\_c = 60));
38. insert equipments;
39. return equipments;
40. }
41. public static List<Case> createMaintenanceRequest(List<Vehicle\_\_c> vehicles){
42. List<Case> maintenanceRequests = new List<Case>();
43. maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(1).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));
44. maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(2).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));
45. insert maintenanceRequests;
46. return maintenanceRequests;
47. }
48. public static List<Equipment\_Maintenance\_Item\_\_c> createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){
49. List<Equipment\_Maintenance\_Item\_\_c> joinRecords = new List<Equipment\_Maintenance\_Item\_\_c>();
50. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));
51. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));
52. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));
53. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));
54. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));
55. joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));
56. insert joinRecords;
57. return joinRecords;
58. }
59. }

## CreateDefaultDataTest.apxc

1. @isTest
2. private class CreateDefaultDataTest {
3. @isTest
4. static void createData\_test(){
5. Test.startTest();
6. CreateDefaultData.createDefaultData();
7. List<Vehicle\_\_c> vehicles = [SELECT Id FROM Vehicle\_\_c];
8. List<Product2> equipment = [SELECT Id FROM Product2];
9. List<Case> maintenanceRequest = [SELECT Id FROM Case];
10. List<Equipment\_Maintenance\_Item\_\_c> joinRecords = [SELECT Id FROM Equipment\_Maintenance\_Item\_\_c];
11. System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');
12. System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');
13. System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');
14. System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items created');
15. }
16. @isTest
17. static void updateCustomSetting\_test(){
18. How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();
19. customSetting.Is\_Data\_Created\_\_c = false;
20. upsert customSetting;
21. System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be false');
22. customSetting.Is\_Data\_Created\_\_c = true;
23. upsert customSetting;
24. System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be true');
25. }
26. }

## 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. //When an existing maintenance request of type Repair or Routine Maintenance is closed,
12. //create a new maintenance request for a future routine checkup.
13. if (!validIds.isEmpty()){
14. Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle\_\_c, Equipment\_\_c, Equipment\_\_r.Maintenance\_Cycle\_\_c,
15. (SELECT Id,Equipment\_\_c,Quantity\_\_c FROM Equipment\_Maintenance\_Items\_\_r)
16. FROM Case WHERE Id IN :validIds]);
17. Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
18. //calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.
19. AggregateResult[] results = [SELECT Maintenance\_Request\_\_c,
20. MIN(Equipment\_\_r.Maintenance\_Cycle\_\_c)cycle
21. FROM Equipment\_Maintenance\_Item\_\_c
22. WHERE Maintenance\_Request\_\_c IN :ValidIds GROUP BY Maintenance\_Request\_\_c];
23. for (AggregateResult ar : results){
24. maintenanceCycles.put((Id) ar.get('Maintenance\_Request\_\_c'), (Decimal) ar.get('cycle'));
25. }
26. List<Case> newCases = new List<Case>();
27. for(Case cc : closedCases.values()){
28. Case nc = new Case (
29. ParentId = cc.Id,
30. Status = 'New',
31. Subject = 'Routine Maintenance',
32. Type = 'Routine Maintenance',
33. Vehicle\_\_c = cc.Vehicle\_\_c,
34. Equipment\_\_c =cc.Equipment\_\_c,
35. Origin = 'Web',
36. Date\_Reported\_\_c = Date.Today()
37. );
38. //If multiple pieces of equipment are used in the maintenance request,
39. //define the due date by applying the shortest maintenance cycle to today’s date.
40. //If (maintenanceCycles.containskey(cc.Id)){
41. nc.Date\_Due\_\_c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
42. //} else {
43. // nc.Date\_Due\_\_c = Date.today().addDays((Integer) cc.Equipment\_\_r.maintenance\_Cycle\_\_c);
44. //}
45. newCases.add(nc);
46. }
47. insert newCases;
48. List<Equipment\_Maintenance\_Item\_\_c> clonedList = new List<Equipment\_Maintenance\_Item\_\_c>();
49. for (Case nc : newCases){
50. for (Equipment\_Maintenance\_Item\_\_c clonedListItem : closedCases.get(nc.ParentId).Equipment\_Maintenance\_Items\_\_r){
51. Equipment\_Maintenance\_Item\_\_c item = clonedListItem.clone();
52. item.Maintenance\_Request\_\_c = nc.Id;
53. clonedList.add(item);
54. }
55. }
56. insert clonedList;
57. }
58. }
59. }

## MaintenanceRequestHelperTest.apxc

1. @isTest
2. public with sharing class MaintenanceRequestHelperTest {
3. // createVehicle
4. private static Vehicle\_\_c createVehicle(){
5. Vehicle\_\_c vehicle = new Vehicle\_\_C(name = 'Testing Vehicle');
6. return vehicle;
7. }
8. // createEquipment
9. private static Product2 createEquipment(){
10. product2 equipment = new product2(name = 'Testing equipment',
11. lifespan\_months\_\_c = 10,
12. maintenance\_cycle\_\_c = 10,
13. replacement\_part\_\_c = true);
14. return equipment;
15. }
16. // createMaintenanceRequest
17. private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
18. case cse = new case(Type='Repair',
19. Status='New',
20. Origin='Web',
21. Subject='Testing subject',
22. Equipment\_\_c=equipmentId,
23. Vehicle\_\_c=vehicleId);
24. return cse;
25. }
26. // createEquipmentMaintenanceItem
27. private static Equipment\_Maintenance\_Item\_\_c createEquipmentMaintenanceItem(id equipmentId,id requestId){
28. Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = new Equipment\_Maintenance\_Item\_\_c(
29. Equipment\_\_c = equipmentId,
30. Maintenance\_Request\_\_c = requestId);
31. return equipmentMaintenanceItem;
32. }
33. @isTest
34. private static void testPositive(){
35. Vehicle\_\_c vehicle = createVehicle();
36. insert vehicle;
37. id vehicleId = vehicle.Id;
38. Product2 equipment = createEquipment();
39. insert equipment;
40. id equipmentId = equipment.Id;
41. case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
42. insert createdCase;
43. Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = createEquipmentMaintenanceItem(equipmentId,createdCase.id);
44. insert equipmentMaintenanceItem;
45. test.startTest();
46. createdCase.status = 'Closed';
47. update createdCase;
48. test.stopTest();
49. Case newCase = [Select id,
50. subject,
51. type,
52. Equipment\_\_c,
53. Date\_Reported\_\_c,
54. Vehicle\_\_c,
55. Date\_Due\_\_c
56. from case
57. where status ='New'];
58. Equipment\_Maintenance\_Item\_\_c workPart = [select id
59. from Equipment\_Maintenance\_Item\_\_c
60. where Maintenance\_Request\_\_c =:newCase.Id];
61. list<case> allCase = [select id from case];
62. system.assert(allCase.size() == 2);
63. system.assert(newCase != null);
64. system.assert(newCase.Subject != null);
65. system.assertEquals(newCase.Type, 'Routine Maintenance');
66. SYSTEM.assertEquals(newCase.Equipment\_\_c, equipmentId);
67. SYSTEM.assertEquals(newCase.Vehicle\_\_c, vehicleId);
68. SYSTEM.assertEquals(newCase.Date\_Reported\_\_c, system.today());
69. }
70. @isTest
71. private static void testNegative(){
72. Vehicle\_\_C vehicle = createVehicle();
73. insert vehicle;
74. id vehicleId = vehicle.Id;
75. product2 equipment = createEquipment();
76. insert equipment;
77. id equipmentId = equipment.Id;
78. case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
79. insert createdCase;
80. Equipment\_Maintenance\_Item\_\_c workP = createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
81. insert workP;
82. test.startTest();
83. createdCase.Status = 'Working';
84. update createdCase;
85. test.stopTest();
86. list<case> allCase = [select id from case];
87. Equipment\_Maintenance\_Item\_\_c equipmentMaintenanceItem = [select id
88. from Equipment\_Maintenance\_Item\_\_c
89. where Maintenance\_Request\_\_c = :createdCase.Id];
90. system.assert(equipmentMaintenanceItem != null);
91. system.assert(allCase.size() == 1);
92. }
93. @isTest
94. private static void testBulk(){
95. list<Vehicle\_\_C> vehicleList = new list<Vehicle\_\_C>();
96. list<Product2> equipmentList = new list<Product2>();
97. list<Equipment\_Maintenance\_Item\_\_c> equipmentMaintenanceItemList = new list<Equipment\_Maintenance\_Item\_\_c>();
98. list<case> caseList = new list<case>();
99. list<id> oldCaseIds = new list<id>();
100. for(integer i = 0; i < 300; i++){
101. vehicleList.add(createVehicle());
102. equipmentList.add(createEquipment());
103. }
104. insert vehicleList;
105. insert equipmentList;
106. for(integer i = 0; i < 300; i++){
107. caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
108. }
109. insert caseList;
110. for(integer i = 0; i < 300; i++){
111. equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id, caseList.get(i).id));
112. }
113. insert equipmentMaintenanceItemList;
114. test.startTest();
115. for(case cs : caseList){
116. cs.Status = 'Closed';
117. oldCaseIds.add(cs.Id);
118. }
119. update caseList;
120. test.stopTest();
121. list<case> newCase = [select id
122. from case
123. where status ='New'];
124. list<Equipment\_Maintenance\_Item\_\_c> workParts = [select id
125. from Equipment\_Maintenance\_Item\_\_c
126. where Maintenance\_Request\_\_c in: oldCaseIds];
127. system.assert(newCase.size() == 300);
128. list<case> allCase = [select id from case];
129. system.assert(allCase.size() == 600);
130. }
131. }

## 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. //Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
4. //The callout’s JSON response returns the equipment records that you upsert in Salesforce.
5. @future(callout=true)
6. public static void runWarehouseEquipmentSync(){
7. System.debug('go into runWarehouseEquipmentSync');
8. Http http = new Http();
9. HttpRequest request = new HttpRequest();
10. request.setEndpoint(WAREHOUSE\_URL);
11. request.setMethod('GET');
12. HttpResponse response = http.send(request);
13. List<Product2> product2List = new List<Product2>();
14. System.debug(response.getStatusCode());
15. if (response.getStatusCode() == 200){
16. List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
17. System.debug(response.getBody());
18. //class maps the following fields:
19. //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
20. for (Object jR : jsonResponse){
21. Map<String,Object> mapJson = (Map<String,Object>)jR;
22. Product2 product2 = new Product2();
23. //replacement part (always true),
24. product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');
25. //cost
26. product2.Cost\_\_c = (Integer) mapJson.get('cost');
27. //current inventory
28. product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');
29. //lifespan
30. product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');
31. //maintenance cycle
32. product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');
33. //warehouse SKU
34. product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');
35. product2.Name = (String) mapJson.get('name');
36. product2.ProductCode = (String) mapJson.get('\_id');
37. product2List.add(product2);
38. }
39. if (product2List.size() > 0){
40. upsert product2List;
41. System.debug('Your equipment was synced with the warehouse one');
42. }
43. }
44. }
45. public static void execute (QueueableContext context){
46. System.debug('start runWarehouseEquipmentSync');
47. runWarehouseEquipmentSync();
48. System.debug('end runWarehouseEquipmentSync');
49. }
50. }

## WarehouseCalloutServiceMock.apxc

1. @isTest
2. global class WarehouseCalloutServiceMock implements HttpCalloutMock {
3. // implement http mock callout
4. global static HttpResponse respond(HttpRequest request) {
5. HttpResponse response = new HttpResponse();
6. response.setHeader('Content-Type', 'application/json');
7. response.setBody('[{"\_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"\_id":"55d66226726b611100aaf742","repeperiod":0,"lifespan":0,"cost":30espan":0,"cost":22,"sku":"100005"}]');
8. response.setStatusCode(200);
9. return response;
10. }
11. }

## WarehouseCalloutServiceTest.apxc

1. @IsTest
2. private class WarehouseCalloutServiceTest {
3. // implement your mock callout test here
4. @isTest
5. static void testWarehouseCallout() {
6. test.startTest();
7. test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
8. WarehouseCalloutService.execute(null);
9. test.stopTest();
10. List<Product2> product2List = new List<Product2>();
11. product2List = [SELECT ProductCode FROM Product2];
12. System.assertEquals(3, product2List.size());
13. System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
14. System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
15. System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
16. }
17. }

## WarehouseSyncScheduleTest.apxc

1. @isTest
2. public with sharing class WarehouseSyncScheduleTest {
3. // implement scheduled code here
4. //
5. @isTest static void test() {
6. String scheduleTime = '00 00 00 \* \* ? \*';
7. Test.startTest();
8. Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
9. String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new WarehouseSyncSchedule());
10. CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
11. System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');
12. Test.stopTest();
13. }
14. }

## WarehouseSyncSchedule.apxc

1. global with sharing class WarehouseSyncSchedule implements Schedulable {
2. // implement scheduled code here
3. global void execute (SchedulableContext ctx){
4. System.enqueueJob(new WarehouseCalloutService());
5. }
6. }

## MaintenanceRequest.apxt

1. trigger MaintenanceRequest on Case (before update, after update) {
2. if(Trigger.isUpdate && Trigger.isAfter){
3. MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
4. }
5. }

## AddPrimaryContact.apxc

1. public class AddPrimaryContact implements Queueable
2. {
3. private Contact c;
4. private String state;
5. public AddPrimaryContact(Contact c, String state)
6. {
7. this.c = c;
8. this.state = state;
9. }
10. public void execute(QueueableContext context)
11. {
12. List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
13. List<Contact> lstContact = new List<Contact>();
14. 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. if(lstContact.size() >0 )
21. {
22. insert lstContact;
23. }
24. }
25. }

## 

## 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. }
11. update updatedAccounts;
12. }
13. }

## VerifyDate.apxc

1. public class VerifyDate {
2. //method to handle potential checks against two dates
3. public static Date CheckDates(Date date1, Date date2) {
4. //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month
5. if(DateWithin30Days(date1,date2)) {
6. return date2;
7. } else {
8. return SetEndOfMonthDate(date1);
9. }
10. }
11. //method to check if date2 is within the next 30 days of date1
12. private static Boolean DateWithin30Days(Date date1, Date date2) {
13. //check for date2 being in the past
14. if( date2 < date1) { return false; }
15. //check that date2 is within (>=) 30 days of date1
16. Date date30Days = date1.addDays(30); //create a date 30 days away from date1
17. if( date2 >= date30Days ) { return false; }
18. else { return true; }
19. }
20. //method to return the end of the month of a given date
21. private static Date SetEndOfMonthDate(Date date1) {
22. Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
23. Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
24. return lastDay;
25. }
26. }

## 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. for(Integer i=0;i<numContactsToGenerate;i++) {
6. Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
7. contactList.add(c);
8. System.debug(c);
9. }
10. //insert contactList;
11. System.debug(contactList.size());
12. return contactList;
13. }
14. }

## DailyLeadProcessor.apxc

1. public class DailyLeadProcessor implements Schedulable {
2. Public void execute(SchedulableContext SC){
3. List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
4. for(Lead l:LeadObj){
5. l.LeadSource='Dreamforce';
6. update l;
7. }
8. }
9. }

## DailyLeadProcessorTest.apxc

1. @isTest
2. private class DailyLeadProcessorTest {
3. static testMethod void testDailyLeadProcessor() {
4. String CRON\_EXP = '0 0 1 \* \* ?';
5. List<Lead> lList = new List<Lead>();
6. for (Integer i = 0; i < 200; i++) {
7. lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.', Status='Open - Not Contacted'));
8. }
9. insert lList;
11. Test.startTest();
12. String jobId = System.schedule('DailyLeadProcessor', CRON\_EXP, new DailyLeadProcessor());
13. }
14. }

## LeadProcessor.apxc

1. public class LeadProcessor implements Database.Batchable<sObject> {
2. public Database.QueryLocator start(Database.BatchableContext bc) {
3. // collect the batches of records or objects to be passed to execute
4. return Database.getQueryLocator([Select LeadSource From Lead ]);
5. }
6. public void execute(Database.BatchableContext bc, List<Lead> leads){
7. // process each batch of records
8. for (Lead Lead : leads) {
9. lead.LeadSource = 'Dreamforce';
10. }
11. update leads;
12. }
13. public void finish(Database.BatchableContext bc){
14. }
15. }

## TestVerifyDate.apxc

1. @isTest
2. public class TestVerifyDate
3. {
4. static testMethod void testMethod1()
5. {
6. Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
7. Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
8. }
9. }

## AccountProcessorTest.apxc

1. @isTest
2. public class AccountProcessorTest {
3. @isTest
4. public static void testNoOfContacts(){
5. Account a = new Account();
6. a.Name = 'Test Account';
7. Insert a;
8. Contact c = new Contact();
9. c.FirstName = 'Bob';
10. c.LastName = 'Willie';
11. c.AccountId = a.Id;
12. Contact c2 = new Contact();
13. c2.FirstName = 'Tom';
14. c2.LastName = 'Cruise';
15. c2.AccountId = a.Id;
16. List<Id> acctIds = new List<Id>();
17. acctIds.add(a.Id);
18. Test.startTest();
19. AccountProcessor.countContacts(acctIds);
20. Test.stopTest();
21. }
22. }

## 

## 

## TestRestrictContactByName.apxc

1. @isTest
2. private class TestRestrictContactByName {
3. static testMethod void metodoTest()
4. {
5. List<Contact> listContact= new List<Contact>();
6. Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' , email='Test@test.com');
7. Contact c2 = new Contact(FirstName='Francesco1', LastName = 'INVALIDNAME',email='Test@test.com');
8. listContact.add(c1);
9. listContact.add(c2);
10. Test.startTest();
11. try
12. {
13. insert listContact;
14. }
15. catch(Exception ee)
16. {
17. }
18. Test.stopTest();
19. }
20. }

## 

## 

## AddPrimaryContactTest.apxc

1. @isTest
2. public class AddPrimaryContactTest
3. {
4. @isTest static void TestList()
5. {
6. List<Account> Teste = new List <Account>();
7. for(Integer i=0;i<50;i++)
8. {
9. Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
10. }
11. for(Integer j=0;j<50;j++)
12. {
13. Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
14. }
15. insert Teste;
16. Contact co = new Contact();
17. co.FirstName='demo';
18. co.LastName ='demo';
19. insert co;
20. String state = 'CA';
21. AddPrimaryContact apc = new AddPrimaryContact(co, state);
22. Test.startTest();
23. System.enqueueJob(apc);
24. Test.stopTest();
25. }
26. }

## LeadProcessorTest.apxc

1. @isTest
2. public class LeadProcessorTest {
3. @testSetup
4. static void setup() {
5. List<Lead> leads = new List<Lead>();
6. for(Integer counter=0 ;counter <200;counter++){
7. Lead lead = new Lead();
8. lead.FirstName ='FirstName';
9. lead.LastName ='LastName'+counter;
10. lead.Company ='demo'+counter;
11. leads.add(lead);
12. }
13. insert leads;
14. }
15. @isTest static void test() {
16. Test.startTest();
17. LeadProcessor leadProcessor = new LeadProcessor();
18. Id batchId = Database.executeBatch(leadProcessor);
19. Test.stopTest();
20. }
21. }

## RestrictContactByName.apxt

1. trigger RestrictContactByName on Contact (before insert, before update) {
2. //check contacts prior to insert or update for invalid data
3. For (Contact c : Trigger.New) {
4. if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
5. c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
6. }
7. }
8. }

## ClosedOpportunityTrigger.apxt

1. trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
2. List<Task> taskList = new List<Task>();
3. for(Opportunity opp : Trigger.new) {
4. //Only create Follow Up Task only once when Opp StageName is to 'Closed Won' on Create
5. if(Trigger.isInsert) {
6. if(Opp.StageName == 'Closed Won') {
7. taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
8. }
9. }
10. //Only create Follow Up Task only once when Opp StageName changed to 'Closed Won' on Update
11. if(Trigger.isUpdate) {
12. if(Opp.StageName == 'Closed Won'
13. && Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {
14. taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
15. }
16. }
17. }
18. if(taskList.size()>0) {
19. insert taskList;
20. }
21. }

## AccountAddressTrigger.apxt

1. trigger AccountAddressTrigger on Account (before insert,before update) {
2. List<Account> acclst=new List<Account>();
3. for(account a:trigger.new){
4. if(a.Match\_Billing\_Address\_\_c==true && a.BillingPostalCode!=null){
5. a.ShippingPostalCode=a.BillingPostalCode;
6. }
7. }
8. }