**Triggers:**

trigger myAccountTrigger on Account(before delete, before insert, before update, after delete, after insert, after update) {

if (Trigger.isBefore) {

if (Trigger.isDelete) {

// In a before delete trigger, the trigger accesses the records that will be

// deleted with the Trigger.old list.

for (Account a : Trigger.old) {

if (a.name != 'okToDelete') {

a.addError('You can\'t delete this record!');

}

}

} else {

// In before insert or before update triggers, the trigger accesses the new records

// with the Trigger.new list.

for (Account a : Trigger.new) {

if (a.name == 'bad') {

a.name.addError('Bad name');

}

}

if (Trigger.isInsert) {

for (Account a : Trigger.new) {

System.assertEquals('xxx', a.accountNumber);

System.assertEquals('industry', a.industry);

System.assertEquals(100, a.numberofemployees);

System.assertEquals(100.0, a.annualrevenue);

a.accountNumber = 'yyy';

}

// If the trigger is not a before trigger, it must be an after trigger.

} else {

if (Trigger.isInsert) {

List<Contact> contacts = new List<Contact>();

for (Account a : Trigger.new) {

if(a.Name == 'makeContact') {

contacts.add(new Contact (LastName = a.Name,

AccountId = a.Id));

}

}

insert contacts;

}

}

}}}

**Mixed DML Future**

This is the definition of the Util class, which contains the future method for inserting a user with a non-null role.

public class Util {

@future

public static void insertUserWithRole(String uname, String al, String em, String lname) {

Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];

UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];

// Create new user with a non-null user role ID

User u = new User(alias = al, email=em, emailencodingkey='UTF-8', lastname=lname,

languagelocalekey='en\_US', localesidkey='en\_US', **profileid = p.Id, userroleid = r.Id,**

timezonesidkey='America/Los\_Angeles', username=uname);

insert u;

}

}

This is the class containing the main method that calls the future method defined previously.

public class MixedDMLFuture {

public static void useFutureMethod() {

// First DML operation

Account a = new Account(Name='Acme');

insert a;

// This next operation (insert a user with a role)

// can't be mixed with the previous insert unless

// it is within a future method.

// Call future method to insert a user with a role.

Util.insertUserWithRole('mruiz@awcomputing.com', 'mruiz', 'mruiz@awcomputing.com', 'Ruiz');

}

}

Batch Apex Code

global class UpdateContactAddresses implements Database.Batchable**<sObject>**, Database.Stateful {

// instance member to retain state across transactions

global Integer recordsProcessed = 0;

static String emailAddress = 'admin@yourcompany.com';

global 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\''**

);

}

global 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;

}

}

List<Database.SaveResults> srs = Database.update(contacts, false);

for(Database.SaveResults sr: srs){

if(!sr.isSuccess()){

Database.Error err = sr.getMessages()[0];

}

}

}

global 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);

Messaging.SigleEmailMessage sem = new Messaging.SingleEmailMessage();

String [] toAddress = new String []{emailAddress};

sem.setToAddress(toAddress);

sem.setSubject(‘Batch Failed’);

sem.setPlainTextBody(‘Failed Results’ + err.getMessage());

Messaging.sendEmail(new Messaging.SigleEmailMessage[] {sem});

}

}

## Testing Batch Apex

@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;

}

static testmethod 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'**]);

}

}

## Queueable Syntax

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());

}

}

CRUD and Field-Level Security (FLS)

public with sharing class ExpenseController {

// This method is recommended.

@AuraEnabled

public static List<ns\_\_Expense\_\_c> getExpenses() {

String [] expenseAccessFields = new String [] {'Id', 'Name', 'ns\_\_Amount\_\_c', 'ns\_\_Client\_\_c', 'ns\_\_Date\_\_c',

'ns\_\_Reimbursed\_\_c', 'CreatedDate'

};

// Obtain the field name/token map for the Expense object

Map<String,Schema.SObjectField> m = Schema.SObjectType.ns\_\_Expense\_\_c.fields.getMap();

for (String fieldToCheck : expenseAccessFields) {

// Check if the user has access to view field

if (!m.get(fieldToCheck).getDescribe().isAccessible()) {

// Pass error to client

throw new System.NoAccessException();

}

}

// Query the object safely

return [SELECT Id, Name, ns\_\_Amount\_\_c, ns\_\_Client\_\_c, ns\_\_Date\_\_c,

ns\_\_Reimbursed\_\_c, CreatedDate FROM ns\_\_Expense\_\_c];

}

# [**Retrieve the RecordType which are Accessible by Profile only**](https://salesforce.stackexchange.com/questions/94041/retrieve-the-recordtype-which-are-accessible-by-profile-only)

List<SelectOption> recordTypes = new List<SelectOption>();

for(RecordTypeInfo info: Account.SObjectType.getDescribe().getRecordTypeInfos()) {

if(info.isAvailable()) {

recordTypes.add(new SelectOption(info.getRecordTypeId(), info.getName()));

}

}

// Include these next 3 lines to output the results during development and debugging

for( SelectOption selopt: recordTypes ) {

System.debug( selopt.getLabel() + ',' + selopt.getValue() );

}

Returning Apex Objects

public class SimpleAccount {

@AuraEnabled public String Id { get; set; }

@AuraEnabled public String Name { get; set; }

public String Phone { get; set; }

// Trivial constructor, for server-side Apex -> client-side JavaScript

public SimpleAccount(String id, String name, String phone) {

this.Id = id;

this.Name = name;

this.Phone = phone;

}

// Default, no-arg constructor, for client-side -> server-side

public SimpleAccount() {}

}

Here’s an example of a controller that returns a collection of custom Apex objects.

public with sharing class SimpleAccountController {

@AuraEnabled

public static List<SimpleAccount> getAccounts() {

// Perform isAccessible() check here

// SimpleAccount is a simple "wrapper" Apex class for transport

List<SimpleAccount> simpleAccounts = new List<SimpleAccount>();

List<Account> accounts = [SELECT Id, Name, Phone FROM Account LIMIT 5];

for (Account acct : accounts) {

simpleAccounts.add(new SimpleAccount(acct.Id, acct.Name, acct.Phone));

}

return simpleAccounts;

}

}

**Aura**

**ContactList Controller**

**({**

**doInit : function(component, event, helper) {**

**// Retrieve contacts during component initialization**

**helper.loadContacts(component);**

**},**

**})**

**ContactList Helper:**

**({**

**loadContacts : function(cmp) {**

**// Load all contact data**

**var action = cmp.get("c.getContacts");**

**action.setCallback(this, function(response) {**

**var state = response.getState();**

**if (state === "SUCCESS") {**

**cmp.set("v.contacts", response.getReturnValue());**

**cmp.set("v.contactList", response.getReturnValue());**

**this.updateTotal(cmp);**

**}**

**// Display toast message to indicate load status**

**var toastEvent = $A.get("e.force:showToast");**

**if (state === 'SUCCESS'){**

**toastEvent.setParams({**

**"title": "Success!",**

**"message": " Your contacts have been loaded successfully."**

**});**

**}**

**else {**

**toastEvent.setParams({**

**"title": "Error!",**

**"message": " Something has gone wrong."**

**});**

**}**

**toastEvent.fire();**

**});**

**$A.enqueueAction(action);**

**},**

**updateTotal: function(cmp) {**

**var contacts = cmp.get("v.contacts");**

**cmp.set("v.totalContacts", contacts.length);**

**}**

**})**

**Aura Component – Card:**

<aura:component>

<lightning:card footer="Card Footer" title="Hello">

<aura:set attribute="actions">

<lightning:button label="New"/>

</aura:set>

<p class="slds-p-horizontal\_small">

Card Body (custom component)

</p>

</lightning:card>

</aura:component>

**Aura - Select**

<aura:component>

<aura:attribute name="colors" type="String[]" default="Red,Green,Blue"/>

<lightning:select name="select" label="Select a Color" required="true">

<aura:iteration items="{!v.colors}" var="color">

<option text="{!color}"></option>

</aura:iteration>

</lightning:select>

</aura:component>

**Aura - if**

<aura:component>

<aura:if isTrue="{!v.truthy}">

True

<aura:set attribute="else">

False

</aura:set>

</aura:if>

</aura:component>

**DataTable**

<aura:component>

*<!-- Imports -->*

<aura:import library="lightningcomponentdemo:mockdataFaker" property="mockdataLibrary"/>

*<!-- attributes -->*

<aura:attribute name="data" type="Object"/>

<aura:attribute name="columns" type="List"/>

*<!-- handlers-->*

<aura:handler name="init" value="{! this }" action="{! c.init }"/>

*<!-- the container element determine the height of the datatable -->*

<div style="height: 300px">

<lightning:datatable

keyField="id"

data="{! v.data }"

columns="{! v.columns }"

hideCheckboxColumn="true"/>

</div>

</aura:component>

({

init: function (cmp, event, helper) {

cmp.set('v.columns', [

{label: 'Opportunity name', fieldName: 'opportunityName', type: 'text'},

{label: 'Account name', fieldName: 'accountName', type: 'text'},

{label: 'Close date', fieldName: 'closeDate', type: 'date'},

{label: 'Confidence', fieldName: 'confidence', type: 'percentage'},

{label: 'Amount', fieldName: 'amount', type: 'currency', typeAttributes: { currencyCode: 'EUR', maximumSignificantDigits: 5}},

{label: 'Contact Email', fieldName: 'contact', type: 'email'},

{label: 'Contact Phone', fieldName: 'phone', type: 'phone'},

{label: 'Website', fieldName: 'website', type: 'url', typeAttributes: { target: '\_self'}},

{label: 'Address', fieldName: 'address', type: 'location'}

]);

})

**Passing List of SObjects to Apex Method (@AuraEnabled) from Lightning JavaScript Controller (Client-side controller):**

**component:**

<aura:component controller="Saveaccesscontroller">

<aura:attribute name="AccountItems" type="Account[]" />

<aura:attribute name="Accountval" type="Account" default="{ 'sobjectType': 'Account',

'Name': 'test'}"/>

<button type="button" onclick="{!c.SaveAccessRequest}">Click Me!</button>

</aura:component>

Pass your params as string using JSON.stringify function.

SaveAccessRequest : function(component, event, helper) {

var ARLIlist = component.get("v.AccessRequestLineItems");

var action = component.get("c.getSaveRecord");

var accessreq = component.get("v.AccessRequest");

var ARLIlistAsString = JSON.stringify(ARLIlist);

var accessreqAsString = JSON.stringify(accessreq);

action.setParams({"ARAsString":accessreqAsString,

"ARLIAsString":ARLIlistAsString});

$A.enqueueAction(action);

}

Update your apex method like this: Take the inputs as string and use JSON.deserialize to convert it back to Object or List.

@AuraEnabled

public static void getSaveRecord(String ARAsString, String ARLIAsString){

Access\_Request\_\_c AR = JSON.deserialize(ARAsString, Access\_Request\_\_c.class);

List<Access\_Request\_Line\_Item\_\_c> ARLI = JSON.deserialize(ARLIAsString, List<Access\_Request\_Line\_Item\_\_c>.class);

Insert AR;

system.debug('====>'+ARLI);

for(Integer i=0; i<ARLI.size(); i++){

ARLI[i].Access\_Request\_ID\_\_c = AR.id;

}

Insert ARLI;

system.debug('====>'+ARLI[0].id);

}

Instead of passing of type List I am passing JSON string to method. Convert your list attribute into JSON string in lightning code (JSON.stringify()) and pass it to apex method as method parameter (String). In apex method deserialize the JSON string into required List.

This is working fine for me. In the same way I am passing List of object from component to apex methods. JSON string > Apex method > deserialize.

**component:**

<aura:component controller="Saveaccesscontroller">

<aura:attribute name="AccountItems" type="Account[]" />

<aura:attribute name="Accountval" type="Account" default="{ 'sobjectType': 'Account',

'Name': 'test'}"/>

<button type="button" onclick="{!c.SaveAccessRequest}">Click Me!</button>

</aura:component>

**controller:** I have queried the account list and passed it to the getReturnValue method in the server side controller.

({

SaveAccessRequest : function(component, event, helper) {

var ARLIlist = component.get("v.AccountItems");

var accessreq = component.get("v.Accountval");

var queryaction = component.get("c.queryRecord");

queryaction.setCallback(this, function(response) {

var state = response.getState();

if (state === "SUCCESS") {

component.set("v.AccountItems",response.getReturnValue());

var action = component.get("c.getSaveRecord");

action.setParams({"AR":$A.util.json.encode(accessreq),

"ARLI":$A.util.json.encode(component.get("v.AccountItems"))});

$A.enqueueAction(action);

}

});

$A.enqueueAction(queryaction);

}

})

**Apex controller:**

Define the param as string and use the convertJSONToListOfSObject method to from the post above to convert the string to list

public class Saveaccesscontroller {

@AuraEnabled

public static void getSaveRecord(String AR,String ARLI){

system.debug('====>'+ARLI);

List<SObject> newSObjectsList = convertJSONToListOfSObject(ARLI);

system.debug('====>'+newSObjectsList);

system.debug('====>'+AR);

}

@AuraEnabled

public static List<account> queryRecord(){

return [select id,name from Account limit 5];

}

private static List<SObject> convertJSONToListOfSObject(String json) {

object[] values = (object[])System.JSON.deserializeUntyped(json);

List<SObject> newSObjectsList = new List<SObject>();

for (Object v : values) {

Map<String, Object> m = (Map<String, Object>)v;

Schema.SObjectType targetType = Schema.getGlobalDescribe().get((String)m.get('sobjectType'));

SObject o = targetType.newSObject();

Map<String, Schema.SObjectField> fields = targetType.getDescribe().fields.getMap();

for (String fieldName : m.keySet()) {

// Filter out any psuedo fields such as LastNameLocal

Schema.SObjectField fi = fields.get(fieldName);

if (fi != null) {

if (fi.getDescribe().isCreateable() && fi.getDescribe().isUpdateable()) {

o.put(fieldName, m.get(fieldName));

}

}

}

newSObjectsList.add(o);

}

return newSObjectsList;

}

}